Федеральное государственное бюджетное учреждение науки
ИНСТИТУТ СИСТЕМНОГО ПРОГРАММИРОВАНИЯ
им. В.П. ИВАННИКОВА
Российской академии наук

На правах рукописи

Черных Андрей Николаевич

Методы и алгоритмы решения задач оптимизации ресурсов в нестационарных распределенных гетерогенных вычислительных средах

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей.

ДИССЕРТАЦИЯ

на соискание ученой степени
доктора физико-математических наук

Научный консультант:
Академик РАН, профессор РАН,
д.ф.-м.н.,
Аветисян Арутюн Ишханович

Москва – 2021

ВВЕДЕНИЕ

Актуальность темы исследования. Теория оптимизации в планировании вычислений находит широкое применение для эффективного решения фундаментальных и прикладных задач, обеспечивающих конкурентное преимущество в научной, производственной, экономической и других сферах человеческой деятельности. Современные распределенные компьютерные системы открыли принципиально новые возможности по увеличению вычислительных мощностей для решения сложных задач. Их характеризуют хорошая масштабируемость, возможность гибкого управления нагрузкой, надежность и отказоустойчивость, расширяемость, безопасность и живучесть в условиях кибератак, природных катастроф и др.

Существует множество различных парадигм реализации распределенных вычислений, и, как следствие, огромное разнообразие научных и технологических направлений, от кластеров, добровольных вычислений (англ., volunteer computing), одноранговых децентрализованных и пи́ринговых (англ., Peer-to-Peer – P2P) сетей до грид- (англ., grid computing) и облачных технологий (англ., cloud computing). В рамках облачных вычислений пул абстрактных, виртуализованных, динамически-масштабируемых вычислительных ресурсов, систем хранения данных, платформ и сервисов предоставляется внешним пользователям по запросу через Интернет. Хотя распределенные вычисления обладают многими неоспоримыми преимуществами, они все еще имеют много узких мест, особенно в области эффективности, безопасности, надежности и т. п.

Подавляющее большинство исследований в области планирования распределенных вычислений предполагает наличие полной информации о работах и среде их выполнения. Однако облачные вычисления подвержены значительной нестационарности во время обеспечения доступа к ресурсам и их использования. Это создает дополнительные сложности для конечных

пользователей, поставщиков ресурсов, сервисных провайдеров и систем планирования.

Объектом исследования в данной диссертации являются задачи планирования распределенных вычислений, которые, в свою очередь, являются подклассом более широкого класса, как правило, NP-трудных задач комбинаторной оптимизации. Ключевое положение планирования вычислений состоит в том, чтобы сопоставить набор работ с набором ресурсов, рассматривая один или несколько критериев оптимизации. Как правило, используются два способа планирования: статическое и динамическое. При статическом подходе полная информация о работах, характеристиках процессоров, операционных системах, а также топологии вычислительной системы известна заранее. Производительность же, например, облачных ресурсов предсказать сложно, т. к. эти ресурсы независимы, гетерогенны и разделяются множеством пользователей. Их архитектура, топология, каналы связи и вычислительные характеристики заранее неизвестны.

При виртуализации ресурсов трудно получить точные знания о состоянии системы. Кроме того, провайдеры постоянно ищут новые пути модернизации инфраструктуры и методы управления ресурсами для улучшения характеристик виртуальной среды и поддержания качества обслуживания (англ., Quality of Service – QoS) ее пользователей.

Динамическое планирование вычислений, обеспечивающее как эффективность использования ресурсов, так и преодоление негативных последствий нестационарности, является одним из ключевых требований к современным вычислительным системам, что постоянно усиливает внимание к этой проблематике.

Недостаточная изученность такого рода ситуационного планирования, несовершенство методов и моделей приводят к существенному недоиспользованию возможностей вычислительных систем.

Решение для современных гетерогенных распределенных вычислительных сред (в дальнейшем именуемых ГРИД) видится в переходе от традиционного

планирования к новым адаптивным подходам, совершенствующим существующие стратегии распределения ресурсов и учитывающим стремительную эволюцию вычислительных средств.

В диссертации рассматриваются роль неопределенности в планировании ресурсов вычислительных систем и алгоритмы, работающие в условиях нестационарности, к которой приводят: требование эластичности вычислений, изменение характеристик машин, виртуализация вычислений при слабой связи работ с инфраструктурой, на которой они выполняются, динамическая миграция работ, изменяемость времени предоставления ресурсов, неточность оценки времени выполнения работ, изменяемость времени обработки и передачи данных в зависимости от использования ресурсов другими пользователями, вариативность рабочей нагрузки, разнообразие временных ограничений обработки (директивные сроки), изменение пропускной способности, сбои, кибератаки и другие явления.

Оптимизация в таких сложных системах должна учитывать не только производительность и эффективность использования ресурсов, а также требования пользователей по качеству обслуживания, стоимости услуг, затратам провайдеров, среднему времени окончания работ, среднему времени ожидания начала их выполнения и др.

Актуальной научной проблемой, на решение которой направлена данная работа, является выработка фундаментальных основ планирования нестационарных ресурсов, их анализ, и разработка новых адаптивных алгоритмов для различных сценариев.

Использование частных решений, разработанных как для стационарных, так и для динамических систем, приводит к необходимости существенной доработки алгоритмов для каждой существующей системы под конкретные условия использования, что увеличивает расходы и сроки внедрения этих систем.

Необходимо рассматривать подходы к решению с разных сторон: онлайн алгоритмы, планирование работ в отсутствии достоверной информации о параметрах системы, самих работах и т. п.

Анализ современных тенденций развития планирования ресурсов, базирующийся на теоретических и практических исследованиях ведущих российских и зарубежных ученых, позволяет сделать важный вывод, что для решения этой актуальной проблемы необходим комплексный подход к построению адаптивных планировщиков и математических моделей, учитывающих отсутствие точных знаний при формировании плана работ.

Для решения поставленной общей научной проблемы проведена ее декомпозиция на ряд частных задач, включающих:

- разработку концепции планирования ресурсов в нестационарных вычислительных средах;
- создание современных моделей и методов планирования и оптимизации ресурсов в условиях изменяющихся характеристик системы;
- разработку методов динамического планирования с механизмами адаптации к изменению параметров среды и непредсказуемости рабочей нагрузки;
- создание подходов к построению адаптивных систем планирования динамически-масштабируемых параллельных работ;
- поддержка механизмов адаптации к многообразию характеристик работ пользователей и требований к их выполнению с поддержкой разных уровней обслуживания и стоимости выполнения работ.

Решение этих проблем с помощью широко используемой теории расписаний и существующих алгоритмов неосуществимо в полной мере, т. к. они изначально разрабатывались для более стационарных систем.

Планирование работ на мультипроцессорах хорошо изучено в течение десятилетий. Существует множество результатов исследований различных вариаций проблемы для одной системы. Некоторые из них дают теоретические знания, в то время как другие дают рекомендации по реализации реальных систем.

Успешные теоретические решения в области планирования распределенных вычислений в гетерогенных средах с помощью аппарата теории расписаний

предложили U. Schwiegelshohn, M. Yahyapour, A. Garey, A. Graham, J. Lenstra, D. Shmoys, F. Pascual, D. Trystram, K. Jansen, P. Bouvry, A. Zomaya, F. Werner, J. Blazewicz, A. Steinberg, E. Pesch, H. Karatza, E. Bampis, M. Pinedo, P. Brucker, Н. Кузюрин. С. Жук, В. Танаев, Ю. Сотсков, В. А. Струсевич, А. Кононов, В. Топорков, А. Лазарев и другие авторы.

Теория планирования в ГРИД тесно связана с множественной упаковкой полос, которая известна как NP-трудная и имеет много реальных приложений. Машины в ГРИД рассматриваются как полосы, а приложения – как прямоугольники, высота и ширина которых равны, соответственно, времени работы и требуемому числу процессоров. При такой постановке задачи решения ищутся с ориентацией на статические характеристики программно-аппаратной инфраструктуры и пользовательских приложений. Для таких проблем, получены теоретические результаты, если время выполнения работ задано, как в оффлайн, так и в онлайн сценариях. Следовательно, они не могут в полной мере поддерживать эффективное функционирование в средах с нестационарными характеристиками.

Все вышесказанное позволяет сделать вывод о необходимости и актуальности разработки новых моделей, алгоритмов, и стратегий организации планирования в ГРИД.

Цель работы состоит в разработке новых стратегий планирования распределенных вычислений в нестационарных гетерогенных средах, обеспечивающих эффективное выполнение работ на основе использования динамического и адаптивного планирования при ограниченном знании о ресурсах и работах.

Основными задачами для достижения поставленной цели являются:

- анализ современных подходов к планированию работ в нестационарных вычислительных системах;
- разработка модели ГРИД, не требующей достоверной информации о параметрах среды и о времени выполнения работ;

- разработка алгоритмов эффективного планирования независимых параллельных работ для различных сценариев;
- получение теоретических оценок конкурентных и аппроксимационных факторов для статического, динамического и адаптивного планирования с различными критериями оптимизации.

Объектом исследования является планирование работ в распределенных гетерогенных вычислительных системах.

Предметом исследования выступают модели, алгоритмы и планировщики выполнения вычислительных работ в ГРИД.

Методы исследования. При решении поставленных задач использовались методы теории расписаний, теории алгоритмов, модели управления ресурсами в вычислительных системах, методы теоретического анализа конкурентных и аппроксимационных факторов статических и динамических расписаний в различных сценариях организации вычислений с различными критериями оптимизации.

Научную новизну диссертации представляют следующие основные научные результаты, выносящиеся на защиту и расширяющие существующий базис теории и практики планирования распределенных гетерогенных вычислений. Все результаты, представленные в данной диссертации, являются новыми.

- Сформулирована проблема планирования в гетерогенной распределенной вычислительной системе в условиях ее нестационарности и предложены подходы к ее решению путем динамической адаптации к изменению системных параметров.
- Показано, что планирование в ГРИД является более сложной задачей, чем соответствующее планирование для многопроцессорной системы, а также то, что хорошо известный списочный алгоритм составления расписаний не может гарантировать постоянного конкурентного фактора, хотя в случае с многопроцессорными системами он достигается.

- Доказано, что граница списочного алгоритма $2 - \frac{1}{m}$ Гарея и Грэхэма (одновременная подача работ), а также Naroska и Schwiegelshohn (построение онлайн расписания), к ГРИД не применима. Поэтому она не может быть гарантирована никаким полиномиальным алгоритмом, если только не $P = NP$.

- Представлены решения для планирования в ГРИД, расширяющие решения для многопроцессорного планирования. Поскольку обычное списочное расписание не подходит для ГРИД, представлен подход планирования, использующий несколько списков работ для каждой машины. В короткой нотации α | β | γ эта проблема характеризуется как $GP_m|size_j|C_{max}$ и $GP_m|r_j, size_j|C_{max}$, где α – характеристики машин, β – характеристики работ, γ – целевая функция задачи.

- Разработан алгоритм планирования работ с фиксированной степенью параллелизма и неизвестным временем обработки, использующий подход «кражи работ» (англ., job stealing), для которого доказан конкурентный фактор 5 для онлайн случая и аппроксимационный фактор 3 для оффлайн случая.

- Предложена и проанализирована адаптивная схема планирования работ с использованием концепции допустимого распределения, которая исключает определенные машины из набора машин, доступных для назначения определенной работы. Проанализирована двухуровневая модель ГРИД, которая интегрирует обе задачи планировщика: распределение параллельных работ с неопределенными требованиями ко времени выполнения по машинам и составление локальных расписаний.

- Получены 3-х и 9-ти аппроксимационные, а также 5-ти и 11-ти конкурентные факторы для алгоритмов, что улучшает и расширяет известные результаты. Решения по планированию принимаются без точной информации о производительности узлов и времени выполнения работ. Конкурентный и аппроксимационный факторы адаптивного алгоритма планирования варьируются путем изменения коэффициента допустимости.

- Представлено подробное исследование влияния коэффициента допустимости, включенного в политику планирования, на общую эффективность

работы системы. Показано, как коэффициент допустимости может быть динамически скорректирован для того, чтобы справиться с динамическим изменением рабочей нагрузки, улучшая конкурентный фактор.

- Представлено новое семейство стратегий планирования, основанных на двух фазах, которые последовательно сочетают последовательное и параллельное выполнение работ.
- Предложено планирование работ с неизвестными временами выполнения, фокусирующееся на регулировании периодов простоя машин в контексте общей политики списочных расписаний.
- Обобщены известные предельные границы производительности планирования в наихудшем случае. Введены два дополнительных параметра, помимо числа процессоров и максимального параллелизма работ, рассматриваемых в литературе: штраф за распараллеливание работ и коэффициент регулирования простоя.
- Показано как при регулировании простоя найти компромисс между более ранним началом распараллеливания (когда распараллеливается больше работ, что приводит к существенным накладным расходам на параллелизацию) и задержкой их распараллеливания (когда больше процессоров остаются без работы) до тех пор, пока не будет доступно меньшее число работ. Эта схема балансирует потребности пользователя с потребностями компьютерной системы.
- Основываясь на моделях реального времени, предложена оригинальная модель облачных вычислений для выполнения и планирования работ на основе понятия уровня обслуживания. Используя обозначение трех полей, проблема может быть описана как $1|prmp, r_{j,online},\ S_i|\sum v_j$ или $P_m|prmp, r_{j,online},\ S_i|\sum v_j$, в зависимости от выбранной модели машины. Каждый уровень обслуживания определяется слак-фактором и ценой за единицу времени обработки.
- Предложен ряд алгоритмов и проведен конкурентный анализ для различных сценариев с одним (SSL) или несколькими (MSL) уровнями

обслуживания, с одной (SM) или несколькими (MM) машинами. Для этих сценариев проанализированы алгоритмы с жадным и ограниченным принятием работ и получены границы конкурентных факторов.

- Разработан и исследован облачный VoIP планировщик, который учитывает динамическую рабочую нагрузку, зависящую от числа и типа вызовов (голосовой почты, видео/аудио конференций, передачи изображений и текста, и т. п.), вариативность времени запуска VM, свойства вызовов, и т. д. Задача планирования рассмотрена как частный случай динамической упаковки в контейнеры. Временное существование элементов (вызовов) при упаковке является принципиальной новизной данной проблемы. В отличие от стандартной формулировки, контейнеры всегда открыты, даже если они полностью заполнены, так как элементы в контейнерах могут быть удалены (завершение вызова). Экспериментальный анализ на реальных данных компании MiXvoip (телекоммуникационный и Интернет-провайдер, Люксембург) показал, что предложенные алгоритмы с методами прогнозирования нагрузки и разработанными настраиваемыми параметрами, превосходят известные стратегии, обеспечивая высокое качество обслуживания и более низкую стоимость и могут быть эффективно использованы в облачной среде VoIP.

Практическая и теоретическая значимость. Работа носит теоретический характер. Полученные в ней результаты позволяют понять природу различных аспектов планирования в распределенных гетерогенных вычислительных системах в условиях нестационарности, и получить теоретические оценки границ оптимизации. Предложены новые приближенные алгоритмы с улучшенными по сравнению с известными в литературе аппроксимационными и конкурентными факторами.

Применение вышеперечисленных результатов диссертационного исследования обеспечивает повышение эффективности планирования параллельных работ в современных вычислительных системах путем его адаптации к изменению параметров рабочей нагрузки и системы.

Также отметим, что предложенные алгоритмы имеют низкую

вычислительную сложность, и при этом позволяют получать хорошие приближенные решения.

Практическая и теоретическая значимость полученных результатов и вклад диссертанта в развитие соответствующей отрасли знаний подтверждается цитированием результатов в международных изданиях: 2445 ссылки в Google Scholar (h-index = 25), 1319 ссылки в Scopus (h-index = 18).

Основные результаты диссертационного исследования были использованы в рамках следующих научно-технических работ:

МиноБР – Министерство образования и науки Российской Федерации,

- "Исследование и разработка передовых методов защиты информации, сохранения конфиденциальности и предотвращения утечки данных при обработке данных в распределенных средах" (Проект 075-15-2020-788), 2020-2022.

РФФИ – Российский фонд фундаментальных исследований,

- "Методы и алгоритмы сбора и обработки данных Internet of Things на основе облачных и туманных вычислительных систем для поддержки интеллектуальных систем мониторинга и автоматизации "Умный город" (проект 20-47-740005), 2020-2021 гг.,
- "Разработка моделей, методов и алгоритмов календарного планирования для контейнерных вычислительных ресурсов при выполнении приложений документооборота в рамках концепции цифрового предприятия" (Проект 18-07-01224), 2018-2020 гг.

CICESE Research Center, Мексика,

- "Планирование в Федерации Облаков" (#634122), 2016-2121,
- “Оптимизация ресурсов в Гридах и системах реального времени” (№634107), 2008-2011 гг.,
- “Проблемы оптимизации ресурсов в кластерах и GRID” (#634107), 2005-2007.

CONACYT. Национальный совет по науке и технике Мексики, Мексика,

- "Стратегии управления и оптимизации энергосистемных ресурсов с качеством обслуживания", 2012-2017 гг., – CONACYT #178415,

- "Многоуровневая иерархия стратегий планирования работ для вычислительного Грида", 2006. CONACYT #48385.

ANII – Национальное агентство по изучению и инновациям, Уругвай, 2015,

- Сотрудничество с Институтом вычислений Инженерного факультета, Университет Республики. Уругвай.

FNR – (Национальный фонд научных исследований, Люксембург) сотрудничество с Люксембургским университетом, Люксембург,

- "Стратегии управления параллельными работами и их оптимизации в облаках",
- "Экологически безопасные стратегии параллельного управления работами и их оптимизация в облачных системах P-2-P".

DAAD – Германская служба академических обменов, сотрудничество с Университетом Геттингена – Институт вычислительной техники, Геттинген, Германия,

- "Управление ресурсами в облачных вычислениях",
- "Планирование рабочей нагрузки на двухуровневых архитектурах Грид".

"Допустимые стратегии выбора ресурсов при двухуровневом планировании работ для вычислительной среды"

BSC – Барселонский суперкомпьютерный центр, Испания

- "Неопределенность в облачных вычислениях".

INRIA Lille – Nord Europe, Исследовательский Центр ИНРИА Лилль,

- challenges of efficient resource provisioning under Uncertainty.

Гренобльский университет в Альпах, ИНРИЯ, Гренобль, Франция,

- эффективное распределение ресурсов.

NSF Национальный научный фонд США – CONACYT сотрудничество с Университетом Южной Калифорнии, Лос-Анджелес, США,

- "Система кэширования времени исполнения I-структуры как структура данных для параллельных кластерных вычислений", (CONACYT #32989-A), 2001-2004.

Под руководством Черных А. Н. разработан комплекс программ, позволяющий исследовать вопросы обеспечения безопасности и надежности хранимых и обрабатываемых данных в облаках в условиях неопределенности возникновения технических сбоев и различного рода хакерских атак, состоящий из:

- «Программа моделирования работы устройств концепции интернет вещей на базе системы остаточных классов» (Свидетельство о государственной регистрации программы для ЭВМ №2017660880, дата регистрации 28.09.2017);
- «Модуль оценки рисков безопасности облачных, краевых и туманных вычислений в условиях вычислительной неопределенности» (Свидетельство о государственной регистрации программы для ЭВМ №2018612694, дата регистрации 21.02.2018);
- «Среда моделирования адаптивной цифровой фильтрации в системе остаточных классов» (Свидетельство о государственной регистрации программы для ЭВМ №2019610639, дата регистрации 15.01.2019);
- «Распределенная система надежного хранения и обработки данных в мультиоблачной среде» (Свидетельство о государственной регистрации программы для ЭВМ №2019611375, дата регистрации 24.01.2019).

Разработанный комплекс программ используется при проведении совместных научных исследований, проводимых научными группами в Северо-Кавказском федеральном университете, Институте системного программирования им. В. П. Иванникова Российской академии наук, Южно-Уральском государственном университете, Московском физико-техническом институте (национальном исследовательском университете) и др.

Достоверность и обоснованность полученных в диссертации результатов подтверждена корректным применением классических методов исследования, строгими доказательствами и анализом эффективности разработанных моделей и алгоритмов. Результаты согласуются с проведенными численными экспериментами.

Соответствие диссертации паспорту специальности. Тема и основные результаты диссертации соответствуют следующим областям исследований паспорта специальности ВАК 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»:

- Модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем;
- Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования;
- Модели, методы, алгоритмы и программная инфраструктура для организации глобально распределенной обработки данных.

Апробация работы. Все результаты диссертационного исследования прошли апробацию на научных мероприятиях в России и за рубежом. Среди них выделим:

Российские конференции

- Международная конференция "Суперкомпьютерные дни в России" 2020, 2019, 2018, 2017, 2015.
- Открытая конференция ИСП РАН им. В.П. Иванникова, 2020, 2019, 2018.
- Международная конференция «Инжиниринг & Телекоммуникации», 2020, 2019, 2017, 2016, 2015.
- Национальный Суперкомпьютерный Форум (НСКФ-2020) Россия, Переславль-Залесский, 2015.

Международные симпозиумы:

- IPDPS – IEEE International Parallel and Distributed Processing, 2021, 2019, 2018, 2017, 2012, 2010, 2008, 2006 (CORE Rank A).
- CCGRID – IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing 2020, 2016 (CORE Rank A).

- Euro-Par – International European Conference on Parallel and Distributed Computing 2019, 2013: 2008 (CORE Rank A).
- CloudCom – IEEE International Conference on Cloud Computing Technology and Science. 2015 (CORE Rank A).
- ICCS – International Conference on Computational Science (2015, 2018) (CORE Rank A).
- HPCS – International Conference on High Performance Computing & Simulation 2020, 2019, 2018, 2014, 2012 (CORE Rank B).
- InterCloud-HPC – International Symposium on Cloud Computing and Services for High Performance Computing Systems 2020 (CORE Rank B).
- PDCAT – International Conference on Parallel and Distributed Computing, Applications and Technologies. 2020 (CORE Rank B).
- DEXA – International Conference on Database and Expert Systems Applications 2017 (CORE Rank B).
- GLOBECOM – IEEE Global Communications Conference (2015) (CORE Rank B).
- CLOUD – IEEE International Conference on Cloud Computing. 2013 (CORE Rank B).
- CollaborateCom – International Conference on Collaborative Computing, 2019, (CORE Rank C).
- PPAM – International Conference on Parallel Processing and Applied Mathematics 2017, 2011, 2007, 2005, 2003 (CORE Rank C).
- UCC – IEEE/ACM International Conference on Utility and Cloud Computing Companion, 2018, 2017, 2016.
- CARLA – Latin America High Performance Computing Conference, 2019, 2018, 2017, 2016.
- ISUM – International Supercomputing Conference in México 2019. 2017, 2016, 2014, 2012, 2010.

Основные результаты были апробированы и обсуждены на приглашенных

пленарных докладах и семинарах.

- Huawei 2020 – Cloud Resource Scheduling and Optimization Workshop. Moscow, 2020
- ICCS-DE 2020 – 2nd International Workshop on Information, Computation, and Control Systems for Distributed Environments, Irkutsk, Russia, 2020
- ICCIS 2020 – 1st International Conference on Systems and Information Sciences. Manta, Ecuador, 2020
- IFI – The Institute of Computer Science, Faculty of Mathematics and Computer Science, Georg-August-Universitat Gottingen, Germany, 2019
- ФГАОУ ВО «Северо-Кавказский федеральный университет», 2018
- UNIANDES – Universidad de los Andes, Bogota, Colombia, 2018
- COMTEL 2018 – X International Conference on Computer Science and Telecommunication, Lima, Perú, 2018
- ISUM 2018 – 9th International Supercomputing Conference in Mexico, Merida, México. 2018
- MPEI – Moscow Power Engineering Institute (National Research University), Moscow. 2017
- NNSU – Lobachevsky State University of Nizhni Novgorod, 2017
- CARLA 2017 – Latin American Conference on High Performance Computing. Buenos Aires, Argentina, 2017
- UCC 2017 – 1st International Workshop on Uncertainty in Cloud Computing. In conjunction with DEXA 2017, Lyon, France, 2017
- PPAM 2017 – 12th International Conference on Parallel Processing and Applied Mathematics, Lublin, Poland, 2017
- ISUM 2017 – 8th International Supercomputing Conference in Mexico, Guadalajara, México. 2017
- Tsinghua University, Department of Computer Science and Technology, Beijing, China. 2016
- SUSU – South Ural State University. Chelyabinsk, Russia. 2016

- UES – State University of Sonora. San Luis Rio Colorado, Мексика 2016
- ISUM 2016 – 7th International Supercomputing Conference in Mexico, Puebla, México. 2016
- CCERD 2015 – The 6th International Conference “Cloud Computing. Education. Research. Development". Moscow, Russia, 2015
- UdelaR – University of the Republic, Uruguay, 2015
- RuSCDays'15 – The Russian Supercomputing Days. 2015, Moscow
- UMONS – University of Mons, Mons, Belgium. 2015
- University of Dortmund, Dortmund, Germany. 2015, 2014, 2012
- BSC – Barcelona Supercomputing Center, 2015
- INRIA Lille – Nord Europe, 2015
- CIMAT – Mexico, 2014
- University of Notre Dame, South Bend, United States, 2013
- University of Luxembourg. 2014, 2012
- CGCT2009 – Collaborative and Grid Computing Technologies Workshop, Cancun, Mexico, 2009
- PPAM 2003 – Fifth International Conference on Parallel Processing and Applied Mathematics, Czestochowa, Poland, 2003
- Instytut Informatyki, Politechnika Poznanska, Poznan, 2003, 2001
- SCI 2002 – 6th World Multiconference on Systemics, Cybernetics and Informatics. Orlando, USA, 2002.
- New Trends in Scheduling in Parallel and Distributed Systems, Marseille (FRANCE) 2008, 2001
- INPG – Institut National Polytechnique de Grenoble, Grenoble, Francia, 2001
- UJF – Universite Joseph Fourier - Grenoble, Francia, 2000
- Aussois – New Trend in Scheduling in Parallel and Distributed Systems Aussois, Francia. 2014, 1998

- Lessach – Workshop on Scheduling in Computer and Manufacturing System, Lessach, Austria. 1996.

Новизна результатов диссертации подтверждена в ходе приглашенных стажировок диссертанта в известных международных научных центрах: в Университете Цинхуа (Китай), Университете Геттингена, в Дортмундском университете, в Клаустальском техническом университете (Германия), Барселонском суперкомпьютерном центре (Испания). в Центре исследований ИНРИА Лилль – Северная Европа (Франция), Гренобльском Альпийском университете (Франция), Люксембургском университете (Люксембург), Национальном политехническом институте Гренобля (Франция), Калифорнийском университете Ирвайн (США), Университете Южной Калифорнии (США), Университете Жозефа Фурье (Франция), Университете Республики (Уругвай) и т. д.

Публикации

По теме диссертации автором было опубликовано 64 статьи, в том числе 21 статья в журналах из списка ВАК или включенных в международные базы данных Scopus и Web of Science из них 12 в Q1, 1 в Q2, 4 в Q3, 2 в Q4 [1–21], 42 работы – в трудах российских и международных конференций [22–63], получено 4 свидетельства о регистрации программ для электронных вычислительных машин (ЭВМ) (Приложение 1) и один патент [64] (Приложение 2).

Личный вклад автора. Диссертационная работа представляет собой многолетнее исследование автора, объединенное тематикой и методами исследования. Все выносимые на защиту результаты получены лично автором. Отдельные элементы доказательств теорем были сделаны в соавторстве при непосредственном участии соискателя. Конфликта интересов с соавторами нет. В получение некоторых результатов были вовлечены 17 студентов магистратуры и 7 аспирантов под непосредственным руководством соискателя.

Структура работы. Диссертация состоит из введения, 7 глав, заключения, библиографии из 223 наименований, и 2 приложений. Общий объем основного

текста работы – 294 страницы, включая 49 таблиц и 72 рисунка.

Краткое содержание работы.

В первой главе рассмотрены основные концепции планирования ресурсов в нестационарных вычислительных средах, приведен краткий обзор основных современных моделей и методов, используемых при решении соответствующих задач с помощью теории расписаний. Описана природа неопределенности и приведена классификация неопределенностей.

Неопределенность возникает в результате неполного знания о процессах, контролирующих предоставление сервисов, например, когда концептуальная модель системы, используемой для предоставления сервисов, не включает все влияющие процессы или факторы. Ее моделируют с помощью теории вероятностей, теории доказательств, теории возможностей и нечетких множеств.

Надежный дизайн планировщика сводит к минимуму влияние неопределенностей на производительность и поведение системы. Традиционно он осуществлялся либо на основе вероятностного подхода, либо на основе анализа наихудшего случая. Оба подхода рассматривают неопределенность либо как отсутствие информации, как случайные переменные, либо как интервальные переменные. На самом деле, неопределенность может представлять собой сочетание разных факторов.

Неопределенность – это важный параметр, который влияет на эффективность вычислений, создавая дополнительные проблемы при планировании. Она требует разработки новых стратегий распределения ресурсов. В первой главе рассмотрены подходы к предоставлению ресурсов, выполнению приложений в условиях изменяющихся характеристик системы. Показано, что стратегии должны обеспечить возможность динамического распределения ресурсов в ответ на изменяющиеся нагрузки, а также динамически адаптироваться к ним, чтобы справиться с различными потоками задач, учитывая свойства систем.

В главе проанализирована и классифицирована неопределенность облачных вычислений и рассмотрены подходы к ее снижению. Особое внимание уделено

роли неопределенности в организации работы провайдеров, пользователей и системных администраторов. Рассмотрены динамические стратегии предоставления ресурсов и услуг, а также методы программирования при наличии неопределенности вычислительной среды. Эти проблемы являются весьма сложными и имеют ключевое значение для принятия решений по планированию работ. Другой важный вклад заключается в рассмотрении этих проблем в свете их сопоставления с другими проблемами и решениями. Кроме того, обсуждаются критерии оптимизации различных вариантов проблемы.

В главе представлено понимание того, как моделировать ГРИД с учетом неопределенности характеристик, динамики распределенных вычислений, масштабирования, отсутствия точной информации о времени вычисления работ и т. д. для предоставления надежных решений, где основной целью является не поиск абсолютно оптимальных решений, а нахождение приближенных решений нечувствительных к изменению параметров среды. Рассмотрены результаты исследований для различных вариаций проблемы планирования дающие теоретические знания или рекомендации по реализации реальных систем.

Смещение акцентов в сторону сервис-ориентированной парадигмы привело к принятию SLA (соглашение об уровне сервиса) как очень важной концепции. Использование SLA при обслуживании пользователей является принципиально новым подходом к планированию работ. При таком подходе планировщики должны учитывать ограничения в отношении QoS независимо от поведения системы. Основная идея заключается в обеспечении различных уровней обслуживания, каждый из которых адресован различным клиентам, чтобы гарантировать время выполнения работы в зависимости от предоставляемых уровней сервиса.

Рассмотрены различные стохастические, реактивные, не требующие знаний и т. п., алгоритмы и эффективные альтернативы известным технологиям детерминированной оптимизации.

Во второй главе рассматриваются вопросы, связанные с онлайн-планированием параллельных работ с неизвестными параметрами и без

прерываний. В прикладной базовой модели гетерогенная распределенная система состоит из большого числа процессоров, которые распределены по нескольким машинам разного размера. Работы являются независимыми, они имеют фиксированную степень параллелизма и поступают с течением времени. Работа может быть выполнена только на процессорах, принадлежащих одной и той же машине. Цель – минимизировать общую продолжительность выполнения всех работ.

В короткой нотации «машинная модель-ограничение-критерий оптимизации» эта проблема характеризуется как $GP_m|size_j\ |C_{max}$.

Показано, что производительность списочного алгоритма планирования Гарея и Грэхэма значительно хуже в распределенных системах, чем в мультипроцессорах.

Поскольку обычное списочное расписание не подходит для ГРИД, представлен планировщик, использующий несколько списков работ. Каждый из этих списков не требует какого-либо конкретного порядка.

Далее представлен алгоритм планирования, гарантирующий конкурентный фактор 5 и аппроксимационный фактор 3. Этот алгоритм может быть реализован с использованием подхода "кражи работ" (англ. “job stealing") и может хорошо подходить для использования в качестве отправной точки для алгоритмов планирования в реальных системах.

В главе сначала рассматривается случай одновременного поступления заявок $r_j = 0$, а затем результат распространяется на сценарий поступления заявок в разное время.

Для определения нижней границы для конкурентного фактора рассмотрена соответствующая статическая проблема с полностью доступной информацией и $r_j = 0$. Эта проблема NP сложная, т. к. $P_m||C_{max}$ – особый случай проблемы $GP_m|size_j|C_{max}$.

Доказана следующая теорема о качественной оценке рассматриваемого планирования.

Теорема 2.1. *Не существует полиномиального алгоритма, который всегда выдает план* S *с* $\frac{C_{max}(S)}{C^*_{max}} < 2$ *для* $GP_m|size_j|C_{max}$ *и всех входных данных, за исключением P=NP.*

В соответствии с теоремой 2.1, граница списочного алгоритма $2 - \frac{1}{m}$ Гарея и Грэхэма (одновременная подача работ), а также Naroska и Schwiegelshohn [177] (онлайн расписания), к ГРИД не применима. Более того, показано, что уже в случае одновременного поступления работ списочное планирование не может гарантировать постоянную границу $\frac{C_{max}}{C^*_{max}}$ для всех случаев проблемы.

Очевидно, что этот результат обусловлен дисбалансом нагрузки, т. к. машины с большим числом процессоров могут выполнять работы с небольшим параллелизмом, что приводит к вынужденному ожиданию выполнения параллельных работ. Это наблюдение предполагает сортировку списка по параллелизму работ в порядке убывания, так чтобы высоко параллельные работы планировались первыми, когда есть выбор. Однако показано, что такой подход также не гарантирует постоянного конкурентного фактора.

Общеизвестной нижней границей для оптимальной продолжительности в случае одновременной загрузки работ для решения проблемы планирования ГРИД является:

$$C^*_{max} \geq max\left\{\max_j p_j, \max_{1\leq i\leq m} \frac{\sum_{j\,|\,size_j > m_{i-1}} p_j \cdot size_j}{\sum_{v=i}^{m} m_v}\right\}$$

По сравнению с границей проблемы $P_m||C_{max}$, эта граница также учитывает недоступность машин небольшого размера для обработки высоко параллельных работ из-за отсутствия возможности их многомашинного выполнения.

Алгоритм планирования ГРИД основан на специальном распределении работ по различным машинам. Это распределение представлено с помощью понятия категорий работ, назначаемых для каждой машины.

Определение 2.1. Для каждой машины M_i существует три различные категории работ:

1. $A_i = \left\{ J_j \,\middle|\, max\left\{\frac{m_i}{2}, m_{i-1}\right\} < size_j \leq m_i \right\}$

2. $B_i = \left\{ J_j \,\middle|\, m_{i-1} < size_j \leq \frac{m_i}{2} \right\}$

3. $H_i = \left\{ J_j \,\middle|\, \frac{m_i}{2} < size_j \leq m_{i-1} \right\}$

Набор A_i содержит все работы, которые не могут быть выполнены на предыдущей (следующей меньшей) машине и требуют более 50% процессоров машины M_i. Набор B_i содержит все работы, которые не могут быть выполнены на предыдущей машине, но требуют не более 50% процессоров машины M_i. Набор H_i содержит все работы, которые требуют более 50% процессоров машины M_i, но также могут быть выполнены на предыдущей машине.

Доказана следующая теорема об аппроксимационном факторе в случае одновременной подачи работ.

Теорема 2.2. *Алгоритм одновременной подачи данных (Concurrent-Submission) гарантирует* $\frac{C_{max}}{C^*_{max}} < 3$ *для всех входных данных и всех конфигураций ГРИД.*

Доказана следующая теорема о конкурентном факторе в случае подачи работ во времени.

Теорема 2.3. *Алгоритм Over-Time-Submission гарантирует* $\frac{C_{max}}{C^*_{max}} < 5$ *для всех входных данных в случае подачи работ во времени.*

В третьей главе анализируются алгоритмы планирования работ, интегрирующие обе задачи планировщика ГРИД: распределение работ по машинам и их локальное расписание. Предлагается и анализируется адаптивная схема распределения работ с использованием концепции допустимого распределения (англ., admissible allocation). Основная идея этой схемы заключается в том, чтобы установить ограничения по распределению маленьких работ на большие машины и динамически адаптировать их к различным рабочим нагрузкам и свойствам системы. Это приводит к снижению возможного дисбаланса нагрузки, когда машины с большим числом процессоров могут выполнять работы с небольшим параллелизмом, что приводит к вынужденному

ожиданию выполнения параллельных работ.

Представлены 3-х-аппроксимационный и 5-ти конкурентный алгоритмы, названные MLBa PS и MCTa PS для случая, когда все работы могут быть выполнены на самой маленькой машине, и представлены 9-аппроксимационный и 11-конкурентный алгоритмы для общего случая.

Алгоритмы планирования для двухуровневых моделей ГРИД можно разделить на глобальную часть распределения работ и локальную часть их планирования. Таким образом, *MPS* рассматривается как двухэтапная стратегия планирования: *MPS = MPS_Alloc+PS*. На первом этапе каждая работа назначается на подходящую машину, используя заданное ограничение. На втором этапе алгоритм PS применяется к каждой машине для работ, назначенных на предыдущем этапе.

Легко заметить, что конкурентный фактор алгоритма *MPS* ограничен снизу конкурентным фактором алгоритма *PS*, учитывая вырожденную ГРИД, которая содержит только одну машину. Лучший из возможных онлайн алгоритмов *PS* имеет конкурентный фактор $2 - 1/m$, где m обозначает число процессоров в одной параллельной машине; см. результаты работы Naroska и Schwiegelshohn [177]. Следовательно, нижняя граница конкурентного фактора для любого общего двухуровневого онлайн *MPS* расписания составляет не менее $2 - 1/m$.

Проанализированы два алгоритма под названием *MLBa + PS* и *MCTa + PS*, где не все машины допустимы для данной работы, хотя и способны выполнять эту работу. Показано, что производительность стратегий распределения зависит от коэффициента допустимости, который может быть динамически адаптирован к различным рабочим нагрузкам и изменениям в конфигурации. Для этого необходимо проанализировать рабочую нагрузку за определенный промежуток времени, чтобы определить соответствующий коэффициент. Временной интервал должен устанавливаться в соответствии с характеристиками нагрузки, динамикой ее изменения и конфигурацией системы.

Коэффициент допустимости $0 < a \leq 1$ параметризует допустимость машин, используемых для распределения работ. Обратите внимание, что, по крайней

мере, одна машина допустима, т. к. a строго больше 0, в то время как $a = 1$ определяет, что все доступные машины допустимы.

Представлено подробное исследование влияния коэффициента допустимости, включенного в политику распределения работ, на общую эффективность работы системы.

Теорема 3.1. *Оффлайн-планирование параллельных работ на ГРИД с идентичными процессорами по алгоритму MCTa + PS с коэффициентом допустимости* $0 < a \leq 1$ *обладает аппроксимационным фактором*

$$\rho \leq \begin{cases} 1 + \dfrac{2}{a^2}, & if\ a \leq \dfrac{s_{f,l}}{s_{f_0,m}} \\ 1 + \dfrac{2}{a(1-a)}, & if\ a > \dfrac{s_{f,l}}{s_{f_0,m}}, \end{cases}$$

где $1 \leq f_0 \leq f \leq l \leq m$ *являются параметрами, которые зависят от конфигурации машины и рабочей нагрузки.*

Теорема 3.2. *Оффлайн-планирование параллельных работ на ГРИД с идентичными процессорами по алгоритму MLBa+PS с коэффициентом допустимости* $0 < a \leq 1$ *имеет аппроксимационный фактор*

$$\rho \leq \begin{cases} 1 + \dfrac{2}{a^2}, & if\ a \leq \dfrac{s_{f,l}}{s_{f_0,m}} \\ 1 + \dfrac{2}{a(1-a)}, & if\ a > \dfrac{s_{f,l}}{s_{f_0,m}} \end{cases}$$

с параметрами $1 \leq f_0 \leq f \leq l \leq m$*, которые зависят от конфигурации машины и нагрузки.*

Теорема 3.3. *Онлайн-планирование параллельных работ на ГРИД с идентичными процессорами с использованием алгоритмов MCTa + PS и MLBa + PS с коэффициентом допустимости* $0 < a \leq 1$ *имеет конкурентный фактор*

$$\rho \leq \begin{cases} 3 + \dfrac{2}{a^2}, & if\ a \leq \dfrac{s_{f,l}}{s_{f_0,m}} \\ 3 + \dfrac{2}{a(1-a)}, & if\ a > \dfrac{s_{f,l}}{s_{f_0,m}}, \end{cases}$$

с параметрами $1 \leq f_0 \leq f \leq l \leq m$*, которые зависят от конфигурации машины*

и нагрузки.

Теорема 3.4. *Планирование параллельных работ на распределенных системах с идентичными процессорами с использованием алгоритмов MCTa +PS, MLBa +PS и работ с размерами, которые могут быть выполнены на самой маленькой машине, имеет аппроксимационный фактор 3 и конкурентный фактор 5.*

Для оценки алгоритмов на разных инфраструктурах HPC проведен их экспериментальный анализ с использованием рабочих нагрузок, основанных на реальных производственных трассах из архивов параллельных рабочих нагрузок (PWA) (Parallel Workload Archive) и грид-рабочих нагрузок GWA (Grid Workload Archive).

Рассмотрены три сценария Грид. В Grid1 используются 7 существующих вычислительных центров с общим числом 4442 процессоров (KTH, SDSC-SP2, HPC2N, CTC, LANL, SDSC-BLUE, SDSC-DS) с 100, 128, 240, 430, 1024, 1152 и 1368 процессорами, соответственно.

В Grid2 рассматриваются 9 центров, из них 5 центров Grid DAS, а также KTH, HP2CN, CTC и LANL с общим числом 2194 процессоров. Размеры машин составляют 64, 64, 64, 64, 100, 144, 240, 430 и 1024.

В сценарии Grid3 рассматривается гораздо меньший Грид из 11 машин с общим числом 136 процессоров.

Проведена экспериментальная оценка двухэтапных стратегий планирования MPS = MPS_Alloc + PS. Разработано девять стратегий распределения MPS_Alloc и девять стратегий распределения с допустимым коэффициентом, включенным в политику планирования. Представлено подробное исследование его влияния, на общую производительность Грид.

Показано, что с точки зрения рассматриваемых критериев, стратегии распределения с допустимым коэффициентом превосходят алгоритмы, которые используют все доступные машины для распределения работ. Такие адаптивные стратегии планирования надежны и стабильны даже в сильно различающихся условиях и способны успешно справляться с различными рабочими нагрузками.

Четвертая глава посвящена задаче распараллеливания, оптимизация которой не всегда эффективна классическими методами из-за разнообразия реальных параллельных и распределенных платформ и/или сред.

Адаптивные алгоритмические схемы, способные динамически изменять распределение работ во время исполнения для оптимизации поведения глобальной системы, являются лучшей альтернативой для решения этой задачи.

В главе рассматривается планирование неклаирвоянтных (англ., non-clairvoyant) параллельных работ с известными требованиями к ресурсам, но с неизвестным временем выполнения, фокусируясь на регулирование периодов простоя (idle) при выполнении.

Рассматривается новое семейство стратегий планирования, основанных на двух фазах, которые последовательно сочетают последовательное и параллельное выполнение работ.

Обобщаются известные предельные границы производительности в наихудшем случае (аппроксимационный фактор), учитывая, помимо числа процессоров и максимальных требований к процессору, рассматриваемых в литературе, два дополнительных параметра, а именно: штраф за распараллеливание работ и коэффициент регулирования простоя. Кроме того, доказывается, что регулирование простоя может улучшить аппроксимационный фактор планирования параллельных работ в режиме разделения пространства.

Для нахождения компромисса между сложностью реальных параллельных систем и желаемой простотой их моделей для теоретического изучения рассмотрена модель распараллеливания, основанная на штрафном факторе. Такая модель используется в реальных планировщиках. Идея заключается в том, чтобы добавить к времени идеального параллельного выполнения накладные расходы, которые включают в себя время, потерянное на коммуникацию, синхронизацию, прерывания и любые дополнительные факторы, которые возникают при управлении параллельным выполнением.

Штрафной фактор неявно учитывает некоторые факторы и ограничения при распараллеливании, когда они неизвестны или их очень сложно определить

формально. Он скрывает реальные природу и характеристики приложения или параметры компьютера, хотя и известные, но слишком сложные для использования в качестве формальных ограничений планирования. Его можно рассматривать как отношение идеального ускорения исполнения к наблюдаемому.

Штраф за выполнение одной работы или всей рабочей нагрузки также можно оценить на основе эмпирического анализа, сравнительного анализа, расчетов, профилирования, оценки производительности с помощью моделирования, измерения, или на основе информации, предоставленной пользователем.

Рассмотрены несколько качественных моделей штрафа как функции от числа процессоров, выделенных на выполнение работы: константная, линейная и выпуклая. Это наиболее распространенные классы при распараллеливании реальных приложений.

Параллельная работа T_i характеризуется тройкой $T = \{p_\text{i}, k_\text{i}, \mu_k^i\}$, а именно: временем ее выполнения на одном процессоре (суммарный объем работы), числом запрошенных процессоров k_i, а также штрафным фактором за ее параллельное выполнение на k_i процессорах. Время ее параллельного выполнения $p_k^i = \frac{\mu_k^i p_i}{k_i}$.

Предполагается, что p_k^i не увеличивается при увеличении k_i, по крайней мере, для разумного числа процессоров, и $p_i = p_k^i$. Пусть $\bar{p} = \max_{1 \le i \le n}\{p_i\}$, $\overline{p_\parallel} = \max_{1 \le i \le n}\{p_k^i\}$, $\underline{k} = \max_{1 \le i \le n}\{k_i\}$, и $\bar{k} = \max_{1 \le i \le n}\{k_i\}$.

Вычисляя ускорение S_k^i при выполнении работы T_i на k процессорах как $\frac{p_1^i}{p_k^i}$, штраф μ_k^i учитывается как отношение идеального ускорения к фактическому $\mu_k^i = \frac{k}{S_k^i}$. Предполагается, что штрафной фактор работы T_i не убывает от числа выделенных процессоров k: $\mu_k^i \le \mu_{k+1}^i$ для любого $1 \le k < m$. Если работа назначена на один процессор, то $\mu_k^i = 1$.

Пусть $\bar{\mu} = \max_{1\le i\le n}\{\mu_k^i\}$ и $\underline{\mu} = \max_{1\le i\le n}\{\mu_k^i\}$. Согласно свойству монотонности работ, назначение работе большего числа процессоров уменьшает время ее выполнения, по крайней мере, до определенного порога, но увеличивает ее вычислительную площадь, следовательно

$$\frac{\underline{\mu}\bar{p}}{\bar{k}} \le \frac{\bar{p}\bar{\mu}}{\bar{k}} \le \bar{p}_{\|} \le \frac{\underline{\mu}\bar{p}}{\underline{k}}$$

Рассмотрено влияние регулирования простоев на общее планирование на основе двухэтапной стратегии. Эта стратегия состоит из двух последовательных фаз. На первой фазе, когда число работ велико, в стратегии распределяются процессоры последовательно без простоев и коммуникаций. Когда достаточное количество работ выполнено, стратегия переходит во вторую фазу с многопроцессорным исполнением. При анализе предполагается, что на первой фазе каждый процессор выполняет одну работу, каждая работа закреплена за одним процессором, таким образом, загрузка процессоров равна 1, а расписание является оптимальным. Неэффективность проявляется, когда остается менее m работ.

Доказана следующая теорема для случая, когда на второй фазе (политика распределения $1 - m$) рассматривается специальный тип расписания, называемый групповым расписанием (gang scheduling).

Теорема 4.1. *Для заданного набора из n независимых работ, каждая из которых имеет штрафной коэффициент от $\underline{\mu}$ до $\bar{\mu}$, назначенных на один и/или на m процессоров, гарантии производительности двухфазной стратегии с регулировкой простоев по параметру a могут быть оценены как:*

$$\rho_{\|}^{*} = max\{\rho_{\|}^{*1}, \rho_{\|}^{*2}\} \quad и \quad \rho^{*} = max\{\rho^{*1}, \rho^{*2}\}, \quad где \quad \rho_{\|}^{*1} \le \frac{1}{\underline{\mu}}\left(1 + \frac{a-1}{m}\right), \quad \rho_{\|}^{*2} \le \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{a}{m}\right) + \frac{a}{\underline{\mu}m}, \rho^{*1} \le 1 + \frac{a-1}{m}, и\ \rho^{*2} \le \bar{\mu}\left(1 - \frac{a}{m}\right) + \frac{a}{m}.$$

Кроме того, $\rho_{\|}^{*1}$ и ρ^{*1} достижимы в случае $m \le \frac{a\bar{\mu}-1}{\bar{\mu}-1}$ и ρ^{*2} достижимы в случае $m > \frac{a\bar{\mu}-1}{\bar{\mu}-1}$.

Показано, что гарантии производительности (аппроксимационный фактор) любого списочного алгоритма для планирования независимых параллельных жестких работ ограничены как:

$$\rho_{\parallel}^{*} \leq \frac{2m-\bar{k}}{m-\bar{k}+1} \text{ и } \rho^{*} \leq \frac{1}{m-\bar{k}+1}\left(\bar{\mu}m + \frac{\mu}{\underline{k}}(m-\bar{k})\right)$$

Последовательная гарантия производительности любого списочного алгоритма планирования независимых параллельных и последовательных работ ограничена следующим образом: $\rho^{*} \leq \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1}$.

Теорема 4.2. *(Последовательная конкурентность - Sequential competitivity). Для заданного набора из n независимых параллельных работ с изменением штрафных коэффициентов от $\underline{\mu}$ до $\bar{\mu}$, назначенных на фиксированное количество процессоров от $\underline{k}$ до $\bar{k}$, последовательный аппроксимационный фактор двухфазной стратегии может быть оценен как:*

$$\rho^{*} = max\{\rho^{*1}, \rho^{*2}\},\ c\ \rho^{*1} \leq 1 + \frac{a-1}{m}\ и\ \rho^{*2} \leq \frac{a}{m} + \frac{1}{m-\bar{k}+1}\left(\underline{\mu}(m-a) + \frac{\mu}{\underline{k}}(m-\bar{k})\right)$$

Теорема 4.3. *(Последовательная конкурентность) для смеси работ). Для заданного набора n независимых параллельных и последовательных работ с вариациями штрафных факторов от 1 до $\bar{\mu}$, назначенных на фиксированное число процессоров, от 1 до $\bar{k}$, последовательный аппроксимационный фактор двухфазной стратегии может быть оценен как:*

$$\rho^{*} \leq \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right).$$

Теорема 4.4. *Для заданного набора из n независимых параллельных работ, назначенных на фиксированное число процессоров от $\underline{k}$ до $\bar{k}$, с минимальным штрафным фактором $\underline{\mu}$, параллельный аппроксимационный фактор двухфазной стратегии может быть оценен как:*

$$\rho_{\parallel}^{*} = max\{\rho^{*1}, \rho^{*2}\},\ c\ \rho_{\parallel}^{*1} \leq 1 + \frac{a-1}{m}\ и\ \rho_{\parallel}^{*2} \leq \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right).$$

Теорема 4.5. *Конкурентность для смеси работ. Для заданного набора из n независимых параллельных и последовательных работ, назначенных на*

фиксированное число процессоров от 1 до $\bar{k}$*, параллельный аппроксимационный фактор двухфазной стратегии может быть оценен как:*

$$\rho_{\parallel}^{*} \leq \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right).$$

При таком регулировании простоя существует компромисс между более ранним началом распараллеливания, когда a мало (следовательно, распараллеливается больше работ, что приводит к большим накладным расходам на распараллеливание), и задержкой их распараллеливания, когда a большое (что приводит к тому, что больше процессоров остаются без работы), до тех пор, пока не будет доступно меньшее число работ. Эта схема балансирует потребности пользователя (работ) с потребностями компьютерной системы.

В пятой главе рассматриваются проблемы планирования виртуализированных вычислительных ресурсов, предоставляемых пользователям через Интернет в форме облачного сервиса (Infrastructure as a Service - IaaS). В типичном сценарии IaaS поставщик инфраструктуры предлагает свои ресурсы по требованию и с различными уровнями сервиса своим клиентам. Эти уровни сервиса в основном отличаются количеством вычислительной мощности, которое заказчик гарантированно получит в течение определенного периода времени.

Для формализации уровня сервиса вводится слак-фактор и цена за единицу времени обработки. Если провайдер принимает работу, она гарантированно выполняется к установленному директивному сроку, который определяется как время ее подачи плюс время обработки, умноженное на слак-фактор заданного уровня обслуживания. После того, как работа была получена, провайдер должен немедленно решить, принимает ли он эту работу или отказывается от нее.

Предлагаются различные алгоритмы и приводится конкурентный анализ для обсуждения различных сценариев для данной модели. Эти сценарии объединяют фиксированные уровни обслуживания с одной или несколькими машинами. Демонстрируется, как можно достигнуть лучшего конкурентного фактора.

Уровень обслуживания S_i связан со слак-фактором $f_i > 1$ и ценой единицы

времени обработки u_i. Поскольку работы поступают со временем ($r_{j,online}$), время обработки p_j и запрашиваемый уровень обслуживания S_i работы J_j становятся известны только в момент ее поступления $r_j \geq 0$. Работа J_j имеет крайний срок выполнения $d_j = r_j + f_i \cdot p_j$, который зависит от слак-фактора или стрейч-фактора f_i уровня обслуживания S_i.

Принятая работа с f_i будет получать в среднем не менее $1/f_i$ ресурсов машины в интервале времени от поступления до завершения работы. Чтобы сохранить ранее предоставленные гарантии уровня обслуживания, работа J_j может быть принята только в том случае, если существует такое расписание, что ни J_j, ни какая-либо ранее принятая работа не выходят за пределы своих сроков, т.е. для всех принятых работ J_j выполняется $C_i \leq d_i$, при этом C_i – это время окончания работы J_j.

Целевая функция представляет собой функцию поставщика инфраструктуры, который хочет максимизировать свой общий доход. Работа J_i с уровнем обслуживания S_i генерирует доход $v_j = x_j \cdot p_j \cdot u_i$. Двоичная переменная $x_j \in \{0, 1\}$ обозначает, принимается ли работа J_j ($x_j = 1$) или нет ($x_j = 0$).

Используя обозначение трех полей, проблема описывается либо как $1|prmp, r_{j,online},\ S_i|\sum v_j$ либо как $P_m|prmp, r_{j,online},\ S_i|\sum v_j$ в зависимости от выбранной модели машины. Проблема требует максимизации целевой функции.

Для SSL-SM, установлена верхняя граница для конкурентного фактора.

Теорема 5.1. $c_V \leq 1 - \left(1 - \frac{p_{min}}{p_{max}}\right) \cdot \frac{1}{f_I}$ *справедливо для SSL-SM с сервисным уровнем* S_I.

Для SSL-SM, используется простой алгоритм, который называется жадным принятием. Этот алгоритм принимает каждую новую работу, если эта работа и все ранее принятые работы могут быть выполнены вовремя.

Теорема 5.2. *Жадное принятие имеет конкурентный фактор* $1 - \frac{1}{f_I}$ *для SSL-SM с уровнем обслуживания* S_I.

Теорема 5.3. $c_V \leq 1 - \frac{1}{f_I}$ справедливо для SSL-PM с уровнем обслуживания S_I, если разность между самым большим и наименьшим временем обработки работы может быть произвольно большой.

Теорема 5.4. $c_V \leq \frac{f_I}{1+f_I \cdot (1-\frac{p_{min}}{p_{max}})}$ справедливо для SSL-PM с уровнем обслуживания S_I и слак-фактором $f_I < \frac{p_{max}}{p_{min}}$

Теорема 5.5. *Жадное принятие имеет конкурентный фактор* $1 - \frac{1}{f_I}$ *для SSL-PM с сервисным уровнем* S_I.

Теорема 5.6. $c_V \leq \max\left\{\frac{\frac{p_{min}}{p_{max}}}{(f_I-1)}, \frac{f_I-1+\frac{p_{min}}{p_{max}}}{f_I-1+\frac{u_I}{u_{II}}}\right\}$ справедливо для MSL-SM с двумя уровнями обслуживания S_I и S_{II} и $f_{II} < 2$.

Теорема 5.7. *Жадный алгоритм принятия имеет конкурентный фактор* $\frac{u_{II}}{u_I}\left(1 - \frac{1}{f_I}\right)$ *для MSL-SM с уровнями обслуживания* S_I *и* S_{II}.

Рассмотрен алгоритм под названием ограниченное принятие для MSL-SM. Этот алгоритм использует коэффициент принятия $h_{II} \geq 1$ и принимает работу J_j с уровнем обслуживания S_{II} только в том случае, если $d_i - C_i \geq h_{II} \cdot p_i$ для всех работ J_i с уровнем обслуживания S_{II} и $d_i \geq d_j$. Здесь C_i обозначает время завершения работы J_i в EDD-расписании с прерываниями после принятия работы J_i.

Теорема 5.8. *Ограниченное принятие имеет конкурентный фактор* $min\left\{\frac{h_{II}-1}{f_{II}}, \frac{f_I-1}{f_I} - \frac{h_{II}}{f_{II}} \cdot \left(1 - \frac{u_{II}}{u_I}\right)\right\}$ *для MSL-SM с уровнями обслуживания* S_I *и* S_{II}, *и* $h_{II} < f_{II} \cdot \left(1 - \frac{1}{f_I}\right)$.

Рассмотрен сценарий MSL-PM с различными уровнями обслуживания и параллельными машинами. MSL-PM близко соответствует реальным требованиям стандартов IaaS, которые, как правило, используют множество машин для поддержки масштабирования. Чтобы эффективно использовать большое количество машин, поставщик IaaS нуждается во многих независимых клиентах.

Для повышения гибкости предлагаются различные уровни обслуживания.

Теорема 5.9. *Жадное принятие имеет конкурентный фактор* $\frac{u_{II}}{u_I}\left(1-\frac{1}{f_I}\right)$ *для MSL-PM с уровнями обслуживания* S_I *и* S_{II}.

Некоторым машинам можно дать более высокий приоритет в обслуживании, чтобы получить лучший результат. Точнее, работы с уровнем обслуживания S_I могут быть распределены на каждую машину, в то время как работы с уровнем обслуживания S_{II} могут быть распределены только на некоторые машины, т.е. используются ограничения допустимости для машин с так называемой иерархической топологией сервера или уровнем обслуживания. Этот алгоритм использует допустимое принятие. Аналогичный подход также использовался в контексте планирования параллельных работ в ГРИД [17].

Теорема 5.10. *Допустимое принятие имеет конкурентный фактор* $\frac{1-\frac{1}{f_{II}}}{1+\frac{1-\frac{1}{f_{II}}}{1-\frac{1}{f_I}}-\frac{u_{II}}{u_I}}$ *для MSL-PM с уровнями сервисов* S_I *и* S_{II}.

Конкурентный фактор допустимого принятия больше, чем конкурентный фактор обычного жадного принятия.

Шестая глава рассматривает модификацию модели IaaS, проанализированную в главе 5. В отличие от предыдущей главы, где рассмотрены сценарии с одним или двумя уровнями обслуживания, здесь анализируются более реалистичные сценарии, где для клиентов предоставляются несколько уровней обслуживания $SL = [SL^1, SL^2, \dots, SL^l, \dots, SL^k]$, а также рассматривается двухкритериальная оптимизация: увеличение дохода провайдера и снижение энергопотребления.

Для данного SL^l работа J_j требует производительность s_j^l, которая гарантируется предоставлением соответствующей виртуальной машины, и взимается стоимость u_j^l за единицу времени выполнения в зависимости от требуемой срочности. Каждая работа J_j описывается кортежем $\left(r_j, w_j, d_j, SL_j^l\right)$, содержащим время выпуска r_j, объем работы w_j, описывающий вычислительную

нагрузку, которая должна быть выполнена до требуемого времени ответа, директивный срок выполнения d_j и уровень обслуживания $SL_j^l \in SL$.

Хотя большое количество уровней приводит к высокой гибкости для клиентов, оно также приводит к значительным накладным расходам на планирование. В главе предложен метод поиска подходящего компромисса.

Рассмотрены гетерогенные машины $M = [M_1, M_2, \ldots, M_m]$, каждая из которых описывается кортежем (s_i, eff_i), указывающим ее относительную скорость обработки s_i и ее энергоэффективность eff_i.

Используется двухуровневый подход к планированию. На верхнем уровне система проверяет, может ли работа быть принята или нет, используя политику принятия Greedy. Если работа принята, то система выбирает машину из набора допустимых машин для ее выполнения на нижнем уровне.

Предложены восемь стратегий планирования. Они характеризуются типом и объемом информации, используемой для принятия решения и делятся на три группы: i) без знаний о системе, без информации о работах и ресурсах; ii) с информацией об энергопотреблении; iii) с информацией о скорости машин.

Экспериментальный анализ проведен с использованием работ HPC, полученных из архива параллельных рабочих нагрузок (PWA) и архива рабочих нагрузок Grid (GWA). Эти трассы являются логами реальных параллельных компьютерных систем и дают хорошее представление о том, как предлагаемые схемы будут работать с реальными пользователями.

Проведен совместный анализ двух критериев сначала на основе ухудшения производительности каждой стратегии по каждой метрике; затем на основе Парето-фронта.

Предложена стратегия, которая превосходит другие алгоритмы. Она доминирует почти во всех тестах. Стратегия стабильна даже в значительно отличающихся условиях. Она обеспечивает незначительное снижение производительности и справляется с различными требованиями. Она использует минимальную информацию, что важно в нестационарной среде, и

характеризуется небольшой вычислительной сложностью; тем не менее, достигает хорошего улучшения критериев и обеспечивает гарантии качества обслуживания.

В седьмой главе формулируются и исследуются проблемы планирования, связанные с облачными системами VoIP. Они учитывают динамическую нагрузку, вариативность времени запуска VM, свойства работ, инфраструктуры, наличия других пользователей, которые совместно используют сервис, и т. д.

Введена CVoIP инфраструктура, которая состоит из m разнородных кластеров узлов связи с различными скоростями обработки вызовов. Каждый узел запускает виртуальные машины, которые описываются началом аренды, временем их развертывания, и вычислительной мощностью.

Рабочая нагрузка состоит из независимых вызовов, которые должны быть спланированы на множестве узлов. Вызов описывается кортежем, который состоит из времени его начала, продолжительности и вклада в загрузку процессора из-за используемого кодека и типа (голосовой почты, видео/аудио конференций, интерактивных телефонных меню, передачи изображений и текста, перераспределения вызовов и т. д.). Время появления вызова недоступно до его начала, а его продолжительность неизвестна до завершения вызова.

В дополнение к описанным выше параметрам VM характеризуется загруженностью (утилизацией) в момент времени t, которая зависит от используемого кодека, типа вызова, других пользователей и вычислительной мощности VM. Каждый кодек предоставляет определенное качество голоса только при достаточно низкой загрузке процессора. Теоретически, загрузка процессора на 100% обеспечивает наилучшую ожидаемую производительность. Однако при загрузке уже до 85% процессор не справляется с нагрузкой, появляются дрожание и разрывы звука.

Введена модель стоимости как функция, зависящая от числа арендуемых VM, и времени их аренды. А также функция снижения качества сервиса в зависимости от загруженности VM. Проанализирован дополнительный критерий качества, связанный с числом звонков, находящихся в режиме ожидания (on hold).

Сформулированная задача рассматривается как частный случай динамической упаковки в контейнеры. Контейнеры представляют собой виртуальные машины, а высота элементов определяет вклад вызова в загрузку виртуальной машины. Планировщик знает только вклад вызова в загрузку VM. Все решения принимаются без информации о продолжительности вызова, скорости поступления вызовов и т. д.

Принципиальной новизной данной проблемы является временное существование элементов (вызовов) в упаковке контейнера. В отличие от стандартной формулировки, контейнеры всегда открыты и динамичны, даже если они полностью заполнены. Элементы в контейнерах могут быть удалены (завершение вызова), а утилизация VM может быть уменьшена в любой момент времени, тогда VM могут использовать свободное пространство для обработки новых вызовов.

Для адаптации к разнообразным гетерогенным средам и вызовам разработаны три настраиваемых параметра: *Utilization Threshold* (UT) порог утилизации, при достижении которого, процессор не может справиться с обработкой звонков и могут появиться джиттеры и прерывистый звук. *Порог аренды* (RT) – это интервал времени, оставшийся до окончания периода аренды VM. Он позволяет избежать ситуации, когда вызов поступает незадолго до окончания времени аренды и завершается после окончания периода аренды, что приводит к продолжению аренды VM с дополнительной оплатой еще на один час, хотя другие VM свободны. *Интервал прогнозирования* (PI - Prediction interval) — это интервал времени, на который выполняется прогнозирование. Предиктор выполняет оценку утилизации на каждом временном шаге (TS – Time Step). Меньший TS обеспечивает лучшее предсказание, но увеличивает накладные расходы системы. Наоборот, больший TS уменьшает накладные расходы системы, но снижает точность предсказания.

Параметры могут быть настроены и динамически адаптированы к различным предпочтениям, рабочим нагрузкам и свойствам облака.

Результаты экспериментов на реальных данных компании MiXvoip (Люксембург) показывают, что предложенные алгоритмы с методами прогнозирования нагрузки превосходят известные стратегии, обеспечивая высокое качество обслуживания и более низкую стоимость и могут быть эффективно использованы в облачной среде VoIP.

Глава 1. ОСНОВНЫЕ КОНЦЕПЦИИ ПЛАНИРОВАНИЯ РЕСУРСОВ В НЕСТАЦИОНАРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДАХ

Несмотря на обширные исследования вопросов неопределенности в различных областях, от вычислительной биологии до принятия экономических решений, исследование неопределенности для вычислительных систем ограничено. В большинстве работ рассматриваются явления неопределенности в восприятии пользователями качеств, намерений и действий провайдеров облачных вычислений, конфиденциальности, безопасности и доступности. Однако роль неопределенности в предоставлении ресурсов и услуг, моделей программирования и т. д. еще не была должным образом рассмотрена в научной литературе.

Существует множество типов неопределенностей, связанных с распределенными вычислительными средами, в частности с облачными вычислениями, которые необходимо учитывать при оценке эффективности планирования работ и услуг. Роль неопределенности в предоставлении услуг и ресурсов облачных систем рассмотрена в [18, 60]. Проанализированы основные источники неопределенности, фундаментальные подходы к планированию в условиях неопределенности, такие как реактивные, стохастические, нечеткие и т. д. Рассмотрен потенциал этих подходов к оптимизации облачных вычислений в условиях не стационарности и рассмотрены методы снижения времени выполнения работ (приложений, заданий) при планировании ресурсов.

1.1 Введение

Облачные вычисления широко признаны как практикующими специалистами, так и исследователями в качестве надежного решения для хранения и обработки данных как в коммерческих, так и в научных целях. Хотя

облачные вычисления обладают многими преимуществами, не все проблемы с ними решены в полной мере, особенно в области безопасности, надежности, производительности, и т. п.

Ситуацию усугубляет варьируемость характеристик, которая сопровождает все эти проблемы. Подавляющее большинство исследований в области планирования предполагает наличие полной и надежной информации о проблеме, работах, т. е. статической детерминированной среде выполнения.

Однако в облачных вычислениях услуги и ресурсы подвержены значительной нестационарности во время обеспечения доступа к ресурсам и их использованию. Нестационарность является значимой проблемой облачных вычислений, которая создает дополнительные сложности для конечных пользователей, поставщиков ресурсов и администраторов. Требуется отказ от привычных парадигм, адаптация существующих моделей к эволюции вычислительных средств и разработка новых стратегий планирования и управления ресурсами для эффективного преодоления нестационарности.

Активно ведутся исследования неопределенности в восприятии пользователями качеств, намерений и действий провайдеров, конфиденциальности, безопасности, доступности и т. д. среди прочих аспектов облачных вычислений [210]. Тем не менее, их роль в планировании ресурсов и услуг еще не была должным образом рассмотрена.

Существует множество источников неопределенности. Таблица 1.1 описывает некоторые из них и кратко поясняет их влияние на планирование. В их числе: эластичность работ, динамическое изменение характеристик, виртуализация со слабой связью приложений с инфраструктурой на которой они выполняются, вариативность времени предоставления ресурсов, неточность оценки времени выполнения работ, изменение времени обработки и передачи данных, временные ограничения обработки (директивные сроки), изменение реальной пропускной способности и другие явления. Рабочая нагрузка может резко меняться. Трудно точно оценить время выполнения работ, построить модели ее прогнозирования на основе истории вычислений, динамически

скорректировать прогноз, исправить ошибки в прогнозах и т. д. [9].

Таблица 1.1 – Параметры облачных вычислений и основные источники их неопределенности

Параметры \ Источники неопределенности	Данные (разнообразие, значение)	Виртуализация	Поступление работ	Миграция	Потребление энергии	Отказоустойчивость	Масштабируемость	Себестоимость (динамическое ценообразование)	Наличие ресурсов	Эластичность	Консолидация	Коммуникация	Репликация	Облачная инфраструктура	Эластичное обеспечение ресурсов	Время предоставления ресурсов
Реальная производительность		●	●	●	●			●	●	●	●	●	●	●	●	●
Реальная пропускная способность	●		●	●	●		●	●	●	●		●	●	●	●	●
Время обработки		●	●	●	●			●	●	●	●	●	●	●	●	●
Доступная память	●		●	●	●	●	●	●	●	●	●			●	●	●
Число процессоров		●	●		●		●	●	●	●				●	●	●
Доступное пространство для хранения	●			●	●	●	●	●	●	●				●	●	●
Время передачи данных	●			●	●			●						●	●	●
Ресурсный потенциал		●	●			●	●	●		●				●	●	●
Производительность сети	●			●		●	●	●						●	●	●

Реальная производительность может изменяться за счет совместного использования общих ресурсов разными виртуальными машинами. Точные знания о системе получить невозможно. Такие параметры, как фактическая скорость процессора, на которой обрабатывается виртуальная машина, число доступных процессоров, или фактическая пропускная способность изменяются со временем. Принцип эластичности облачных ресурсов, когда пользователь может менять запросы на ресурсы оказывает положительное влияние на качество обслуживания, но добавляет новый фактор неопределенности.

Неопределенность может присутствовать в различных компонентах вычислительного и коммуникационного процессов. Важны следующие знания: как конкретные динамические вычислительные и коммуникационные характеристики влияют на планирование; как эти характеристики могут быть использованы планировщиком для снижения влияния неполноты информации и

достижения желаемого QoS; как можно эффективно решить соответствующую проблему оптимизации; как обеспечить масштабируемое и надежное поведение ГРИД при ограничениях, таких как бюджет, QoS, SLA, стоимость энергозатрат и т.д. Изучаются различные стохастические, адаптивные, реактивные и не требующие знаний алгоритмы планирования, рассматриваемые как эффективные альтернативы известным технологиям детерминированной оптимизации теории расписаний.

Неопределенные факторы, которые могут снизить надежность и безопасность данных такие как неожиданные и несанкционированные модификации данных, аппаратные и программные сбои, ошибки дисков, нарушение целостности или потеря данных, вредоносные вторжения, фальсификации, отказ в доступе на длительное время, утечка информации, сговор и др., рассматриваются в [53]. Авторы предлагают настраиваемую, надежную и безопасную схему распределенного хранения данных с усовершенствованным методом обнаружения и исправления данных, а также скорости их кодирования/декодирования.

Случаи технических сбоев, нарушений безопасности данных и сговора трудно предсказать. Этот тип неопределенности является одной из основных проблем при проектировании надежной ИТ-инфраструктуры. Для снижения избыточности данных при шифровании и вреда, вызванного облачным сговором, предлагаются модифицированные пороговые Asmuth-Bloom и взвешенные схемы обмена секретами Mignotte [209].

В большинстве существующих решений оптимизации предполагается, что поведение виртуальных машин и сервисов предсказуемо и стабильно в работе. В реальных облачных инфраструктурах эти предположения не оправданы. Хотя большинство провайдеров гарантируют определенную скорость процессора, емкость памяти и объем локального хранилища для каждой выделенной виртуальной машины, фактическая производительность зависит от используемого физического оборудования, а также от использования общих ресурсов другими виртуальными машинами, назначенными на одну и ту же вычислительную

машину. Это справедливо и для коммуникационной инфраструктуры, где реальная пропускная способность очень динамична и ее трудно гарантировать.

Оптимизация и планирование облачных VoIP-услуг для предоставления пользователям соответствующих уровней качества обслуживания, а провайдерам услуг и минимизации стоимости используемых виртуальных машин в облаке рассмотрены в [29]. Эта проблема осложняется непредсказуемым влиянием временных задержек при запуске виртуальных машин, изменением скорости обработки звонков и неопределенным временем разговоров (Глава 7).

Объединение виртуализированных динамически масштабируемых вычислительных ресурсов, хранилищ, программного обеспечения и услуг облачных сред добавляет новое измерение к проблеме планирования. Способ предоставления сервисов зависит не только от свойств и требуемых ресурсов, но и от других пользователей, которые совместно используют ресурсы. Управление и оптимизация облачной инфраструктуры является сложной задачей. Существующие модели планирования недостаточно точно учитывают неопределенность и динамические изменения производительности, присущие неоднородным и разделяемым инфраструктурам.

Чтобы лучше понять последствия такой нестационарности, в этой главе изучаются проблемы планирования ресурсов и услуг, связанные с существующими облачными инфраструктурами: приватными, публичными и гибридными.

1.2 Анализ основных источников неопределенности параметров вычислительной среды

Несмотря на обширные исследования проблем неопределенности в последние десятилетия в различных областях, начиная с физики, вычислительной биологии и заканчивая принятием решений в экономике и социальных науках [65], изучение неопределенности для вычислительных систем лимитировано.

Снижение влияния неопределенности на производительность, эффективность планирования, безопасность и надежность облачных систем является быстро растущей темой исследований. Ее анализ становится неотъемлемой частью разработки стратегий планирования ресурсов и услуг.

В данной главе представлен анализ того, как планировать облачные вычисления с учетом неопределенности предоставления ресурсов, динамики распределенных вычислений, масштабирования, отсутствия точной информации о времени вычисления работ для предоставления надежных решений управления ресурсами. Основной целью является не поиск абсолютно оптимальных решений, а нахождение хорошего поведения не сильно чувствительного к воздействию изменения параметров среды.

Существующие неопределенности можно классифицировать несколькими разными способами в соответствии и с их природой.

1) Долгосрочная неопределенность облаков связана с тем, что их концепция плохо изучена и непреднамеренные финансовые, политические и др. факторы могут повлиять на поведение облачных ресурсов.

2) Ретроспективная неопределенность связана с отсутствием информации о результатах облачных решений в прошлом.

3) Техническая неопределенность является следствием невозможности прогнозирования точных результатов инвестиционных, архитектурных, программных и сервисных решений.

4) Стохастическая неопределенность является результатом вероятностной (стохастической) природы процессов и явлений в сложных неоднородных средах, где не существует достоверной статистической информация, либо поведение является стохастическим, но необходимая статистическая информация для оценки вероятностных характеристик отсутствует, и гипотеза о стохастическом характере требует проверки.

5) Ограничительная неопределенность обусловлена частичным или полным незнанием условий и ограничений, при которых принимаются решения об использовании и развитии состояния среды.

6) Неопределенность поведения участников возникает из-за потенциального конфликта основных заинтересованных сторон: провайдеров ресурсов, пользователей и администраторов, где каждая сторона имеет свои предпочтения, цели и задачи, обладает неполной, неточной информацией о мотивах и поведении сторон.

7) Неопределенность целей связана с их конфликтностью и невозможностью выбора одной цели для всех участников или необходимостью построения многоцелевой модели оптимизации, где рассматривается проблема многокритериального выбора оптимальных решений и где существует возможно бесконечное число решений.

Эти неопределенности можно сгруппировать в две группы, определяющие параметрические и системные неопределенности.

Параметрические неопределенности возникают в результате неполного знания и изменения параметров, например, когда данные являются неточными или не в полной мере репрезентативными для планирования ресурсов. Они, как правило, оцениваются с использованием статистических методов и выражаются вероятностно. Их анализ количественно определяет влияние случайных входных величин на результаты оптимизации. Эффективность и точность вероятностного анализа неопределенности является вопросом компромисса. Этот тип неопределенности не уменьшается, т. к. он является свойством самой системы.

Системная неопределенность возникает в результате неполного понимания процессов, контролирующих планирования сервисов, например, когда концептуальная модель системы, используемой планировщиком, не включает все влияющие процессы или факторы. Неопределенность можно уменьшить, если получать больше информации. Ее моделируют с помощью теории вероятностей, теории доказательств, теории возможностей и нечетких множеств.

Надежный дизайн сводит к минимуму влияние неопределенностей на производительность и поведение системы. Традиционно он осуществлялся либо на основе вероятностного подхода, либо на основе анализа наихудшего случая. Оба подхода рассматривают неопределенность либо как случайные переменные,

либо как интервальные переменные. На самом деле, неопределенность может представлять собой сочетание обоих факторов.

Моделирование методом Монте-Карло может быть использовано для проведения оценки устойчивости в рамках оптимизации. Вероятностный подход рассматривается как наиболее строгий подход к анализу неопределенности и смягчению ее последствий в связи с его согласованностью с теорией анализа решений.

Приведем примеры целевых функций оптимизации в стохастической среде (таблица. 1.2) и оценки качества решений.

Пусть работа j должна обрабатываться P_j единиц времени, где P_j является случайной переменной.

Пусть $\mathbb{E}[P_j]$ будет ожидаемым значением времени обработки работы j, а p_j - конкретной реализацией P_j.

Можно предположить, что все случайные переменные времени обработки стохастически независимы и определяются дискретными распределениями вероятностей, и без потери общности считать, что P_j, а также все даты поступления r_j и окончания d_j работ являются интегральными значениями.

Следуя определениям Megow и др. [168], стохастическая политика $\prod$ является ρ-аппроксимируемой (ρ-конкурентной) для $\rho \geq 1$, если для всех случаев I, $\mathbb{E}[\prod(\mathrm{I})] \leq \rho\mathbb{E}[OPT(\mathrm{I})]$, где $\mathbb{E}[\prod(\mathrm{I})]$ и $\mathbb{E}[OPT(\mathrm{I})]$ обозначают соответственно оптимальные значения, полученные с помощью $\prod$.

Решением стохастической проблемы планирования является не расписание, а так называемая политика планирования, которая принимает решения о планировании в моменты времени t без информации о будущем, например, о фактическом времени выполнения работ p_j, которые еще не завершены ко времени t. Пусть C_j и w_j будут временем завершения и весом (важностью или приоритетом) работы j соответственно.

Цель состоит в том, чтобы минимизировать ожидаемые целевые функции. Эти функции можно сгруппировать по следующим категориям: обычные целевые

функции, которые не ухудшают время завершения работ, такие как общее взвешенное время завершения, общее взвешенное число запоздалых работ, максимальная задержка и так далее, и нерегулярные целевые функции, такие как ожидаемая задержка, дисперсия времени завершения, общие расходы [87].

Таблица 1.2. Примеры целевых функций в стохастической среде

Параметры	Определение
Ожидаемое общее взвешенное время завершения (Expected total weighted completion time)	$\mathbb{E}\left[\sum_{J\in j}(w_j \cdot C_j)\right] = \sum_{j\in J}(w_j \cdot \mathbb{E}[C_j])$
Ожидаемое среднее время оборачиваемости (Expected mean turnaround time)	$\frac{1}{n}\sum_{j=1}^{n}\mathbb{E}\left[C_j - r_j\right]$
Ожидаемое среднее время ожидания (Expected mean waiting time)	$\frac{1}{n}\sum_{j=1}^{n}\mathbb{E}\left[C_j - P_j - r_j\right]$
Ожидаемое среднее ограниченное замедление (Expected mean bounded slowdown)	$\frac{1}{n}\sum_{j=1}^{n}\frac{\mathbb{E}[C_j - r_j]}{\max\left[10, \mathbb{E}[P_j]\right]}$
Ожидаемая общая взвешенная задержка (Expected total weighted tardiness)	$\sum_{j=1}^{n}\mathbb{E}[w_j \cdot \max(P_j - d_j, 0)]$
Ожидаемый срок выполнения (Expected makespan)	$\mathbb{E}[C_{max}] = max(\mathbb{E}[C_j])$

Эти показатели обычно используются для выражения целей различных заинтересованных сторон (конечных пользователей, поставщиков ресурсов и администраторов). Показатель надежности гарантирует непревышение определенного уровня деградации (в пределах допустимого отклонения) характеристик системы, несмотря на колебания нестационарных параметров вычислительной среды [69]. Он также может быть измерен среднеквадратическим отклонением, дифференциальной энтропией и т. д. [91].

1.3 Проектирование приложений для работы в условиях неопределенности

Понимание и учет неопределенности должны привести к повышению эффективности планирования ресурсов. Большинство облачных приложений требуют наличия коммуникационных ресурсов для обмена информацией между

сервисами, с базами данных или с конечными пользователями. Однако провайдеры могут не знать количество данных, которыми придется управлять, или количество вычислений, необходимых для выполнения работ. Например, каждый раз, когда пользователю требуется статус банковского счета, может потребоваться разное время на его проверку и разный объем данных на доставку.

Лишь немногие подходы к моделям программирования учитывают коммуникационные требования и зачастую весьма абстрактно. Более того, если приложения должны использовать облако, среда их выполнения неизвестна во время разработки - число доступных машин, их местоположение, их возможности. Топология и реальная пропускная способность каналов связи также могут быть неизвестны заранее. В целом среда выполнения отличается для каждого вызова программы/сервиса. Чтобы справиться с такой неопределенностью, программисты либо должны явно писать адаптивные программы, либо облачное программное обеспечение должно справляться с такой нестационарностью [13, 66] (Глава 2, 3, 4, 6 и 7).

Пользовательские адаптивные решения основаны на огромных усилиях по программированию. Для эффективного использования облака программы должны быть отделены от среды исполнения. Программы должны разрабатываться для единообразных и предсказуемых виртуальных сервисов, тем самым упрощая их разработку. Модель облачных приложений должна обеспечивать высокий уровень представления вычислений и коммуникаций, основанный на природе программы и не зависящий от среды исполнения. Отображение вычислений на серверах, балансировка нагрузок между различными серверами, удаление недоступных серверов из вычислений, отображение коммуникационных задач и балансировка коммуникационных нагрузок между различными линиями связи должны обеспечиваться системой исполнения.

Новая модель приложений CA-DAG (Коммуникационно-ориентированная модель направленного ациклического графа - Communications-Aware DAG model) для облачных вычислений, которая преодолевает недостатки существующих подходов и позволяет более эффективно сглаживать неопределенности

предложена в [39, 41].

Она основана на использовании направленного ациклического графа – Directed Acyclic Graph, который наряду с вычислительными вершинами имеет отдельные вершины для представления коммуникационных задач. Такое представление позволяет принимать самостоятельные решения о планировании ресурсов, назначать процессоры для выполнения вычислительных работ и сетевые ресурсы для передачи информации. Предлагаемая коммуникационная модель создает пространство для оптимизации многих существующих решений по планированию ресурсов, а также для разработки совершенно новых схем планирования повышенной эффективности.

Программа представлена в виде ориентированного ациклического графа $G = (V,E,\omega,\varphi)$. Набор вершин $V = \{V_c, V_{comm}\}$ состоит из двух непересекающихся подмножеств V_c и V_{comm}. Набор $V_c \subseteq V$ представляет вычислительные задачи, а набор $V_{comm} \subset V$ представляет коммуникационные задачи программы.

Вычислительная задача $v_i^c \in V_c$, представляется парой (I, D_c) с числом I инструкций (объемом работ), которые должны быть выполнена в определенный срок D_c. Задача коммуникации $v_i^c \in V_{comm}$ описывается параметрами $(S,\ D_{comm})$ и определяется как количество информации S в битах, которое должно быть успешно передано в течение установленного срока D_{comm}. Положительные веса $\omega(v_i^c)$ и $\varphi(v_i^{comm})$ отражают соответственно стоимость вычислений в узле $v_i^c \in V_c$ и стоимость коммуникации в узле $v_i^{comm} \in V_{comm}$. Набор ребер E состоит из направленных ребер e_{ij}, которые представляют собой зависимости между вершинами $v_i \in V$ и $v_j \in V$. Это помогает определить порядок выполнения задач, которые не обмениваются данными.

Основное различие между вершинами связи V_{comm} и ребрами E заключается в том, что V_{comm} представляет собой коммуникационные задачи, возникшие в сети, что делает их предметом коммуникационных конфликтов, значительной задержки и ошибок связи. Набор E соответствует зависимостям

между вычислительными и коммуникационными задачами, определяющим порядок их выполнения.

1.4 Планирование ресурсов в условиях неопределенности

Ключевое положение теории расписаний направлено на то, как сопоставить набор работ с набором ресурсов, рассматривая один или несколько критериев оптимизации. Как правило, существует два способа: статическое планирование и динамическое планирование. При статическом подходе подробная информация о работах и характеристиках процессора, а также топологии системы известна заранее. К сожалению, производительность облачных ресурсов предсказать сложно, т. к. эти ресурсы не предназначены для одного конкретного пользователя, и, кроме того, нет информации о топологии. Более того, в целом, из-за техники виртуализации невозможно получить точные знания о состоянии системы. Реальные характеристики со временем меняются. Кроме того, провайдеры постоянно ищут пути обновления и совершенствования ресурсов для обеспечения QoS. При этом параметры меняются.

Смещение акцентов в сторону сервис-ориентированной парадигмы привело к принятию SLA как очень важной концепции. Использование SLA при обслуживании пользователей является принципиально новым подходом к планированию. При таком подходе планировщики должны учитывать ограничения в отношении QoS независимо от нестационарности поведения системы. Основная идея заключается в обеспечении различных уровней обслуживания (SL), каждый из которых адресован различным клиентам, чтобы гарантировать время выполнения работы в зависимости от SL.

Основываясь на моделях режимов реального времени, модель планирования работ, где каждый SL описывается слак-фактором (slack-factor) и ценой за единицу времени обработки введена в [42]. Если провайдер принимает работу, она гарантированно выполняется в срок. Теоретически проанализированы модели

одиночных (SM) и параллельных машин (PM), с одним (SSL) и несколькими уровнями обслуживания (MSL). Анализ основан на конкурентном факторе, который измеряется как соотношение между доходом поставщика инфраструктуры, полученным алгоритмом планирования, и оптимальным доходом. Алгоритмы основаны на адаптации алгоритма EDD (работы с ближайшим сроком выполнения запускаются в первую очередь) для планирования рабочих мест со сроками выполнения (Глава 5 и 6).

Чтобы показать целесообразность и конкурентоспособность алгоритмов, проведено исследование их производительности с использованием моделирования в [57, 156]. Параметры, которые являются критически важными для практического внедрения алгоритмов планирования, проанализированы с использованием рабочих нагрузки реальных производственных гетерогенных HPC-систем.

1.5 Балансировка нагрузки вычислительных систем

Одной из возможных методик решения проблем вычислительного и коммуникационного дисбаланса, связанных с неопределенностью, является автоматическая балансировка нагрузки, позволяющая улучшить распределение ресурсов. Для эффективной балансировки важно определить: понятия перегрузки системы и ее недостаточной загрузки; кто и когда инициализирует балансировку нагрузки; число работ, подлежащих миграции; временной интервал, используемый для миграции; число VM, выбранных для миграции, и т. д. Это помогает достичь высокого уровня использования ресурсов и качества обслуживания за счет эффективного распределения вычислительных ресурсов.

Эластичный алгоритм балансировки нагрузки распределяющий входящий трафик (VM запросы, работы) по нескольким системам для достижения более высокого качества обслуживания разработан в [123]. Он обнаруживает перегруженные ресурсы и автоматически перенаправляет трафик на недостаточно

загруженные ресурсы. Если все узлы облака перегружены, то он может автоматически увеличить мощность обработки запросов в ответ на входящий трафик. Когда облако недогружено, оно может уменьшить масштаб. Емкость может быть увеличена или уменьшена в реальном времени в зависимости от потребляемых вычислительных и сетевых ресурсов.

Эластичность позволяет обрабатывать непредсказуемую рабочую нагрузку и избегать перегрузки. Введено понятие допустимости ресурсов, когда для выполнения конкретной работы выбирается только ограниченный набор ресурсов, учитываемый в стратегиях планирования, чтобы избежать перегрузки или простаивания [17]. Миграция работ может привести к огромным накладным расходам на коммуникацию. Допустимый фактор ограничивает такие накладные расходы, избегая отправки работ на более дальние узлы. Допустимый коэффициент учитывает статические факторы (такие как расстояние) и динамические факторы (такие как фактическая пропускная способность и трафик в сети).

Эти характеристики не учитываются в большинстве систем, т. к. их трудно определить количественно, и они варьируются в зависимости от приложений.

1.6 Адаптивное планирование

Расписание работ на мультипроцессорах хорошо изучено в течение десятилетий. Существует множество результатов исследований для различных вариаций проблемы планирования для одной системы. Некоторые из них дают теоретические знания, в то время как другие дают рекомендации по реализации реальных систем. Однако до сих пор проблема адаптивного планирования решалась редко, что может привести к неэффективному распределению ресурсов и плохому энергопотреблению [20].

Одной из структурных причин неэффективности онлайн-планирования работ является использование больших машин для выполнения работ с

небольшими требованиями к процессору, что может привести к тому, что высокопараллельные работы вынуждены ждать их выполнения. С этой целью введен допустимый коэффициент, характеризующий доступность машин для размещения работ [17]. Основная идея заключается в том, чтобы установить ограничения на распределение работ и динамически адаптировать их к различным нагрузкам и свойствам системы (Глава 3).

Конкурентный фактор адаптивного алгоритма онлайн планирования MLBa+PS с допустимым распределением заданий, который варьируется от 17 до бесконечности путем изменения допустимого коэффициента, был выведен для конкретных характеристик рабочей нагрузки. Этот результат был расширен для более общей модели рабочей нагрузки с конкурентным фактором 17 в [21].

3-аппроксимационные и 5-конкурентные алгоритмы, названные MLBa+PS и MCTa+PS для случая, когда все работы подходят к самой маленькой машине, при этом для общего случая получен аппроксимационный фактор 9 и конкурентный фактор 11 представлены в [21]. Рассмотрена модель планирования с двумя этапами. На первом этапе работы распределяются на подходящую машину, а на втором этапе локальное планирование применяется к каждой машине независимо друг от друга.

В реальном сценарии допустимый коэффициент может быть динамически скорректирован в зависимости от изменения конфигурации и/или нагрузки. С этой целью можно проанализировать результаты прошлой рабочей нагрузки и распределения в течение заданного промежутка времени для определения соответствующего допустимого коэффициента. Этот временной интервал должен устанавливаться в соответствии с динамикой изменения характеристик рабочей нагрузки и конфигурации. Также можно итеративно аппроксимировать допустимый коэффициент.

1.7 Планирование в условиях неопределенности

В последние годы теория вероятностей и статистические методы включаются в расписания для обработки неопределенностей. Обширное исследование в этой области, основные результаты и тенденции можно найти в книге Соцкова и Вернера [206]. В ней обсуждаются подходы, использующие стохастические и нечеткие методы, а также важные вопросы надежности и стабильности планирования. Неопределенность рассматривается в двух основных рамках: стохастическое планирование и онлайн-планирование. Стохастическое планирование решает проблемы, при которых свойства работ, например, время обработки, крайние сроки выполнения и время их поступления, моделируются как случайные переменные, точные значения которых неизвестны до их поступления и завершения, соответственно. Онлайн-планирование характеризуется отсутствием знаний о будущих работах. Решения могут приниматься каждый раз, когда появляется новые работы. Известны только те работы, которые поступили раньше.

Модель планирования в условиях неопределенности, которая сочетает онлайн-планирование и стохастическое планирование рассматрена в [168], [169] и [213]. Работы приходят с течением времени, и нет никаких знаний о будущих работах. Предполагается, что время обработки работ является стохастическим. Как только работа становится известна, планировщик узнает только вероятностное распределение времени обработки. Авторы рассматривают стохастическую политику онлайн-планирования на идентичных параллельных машинах, чтобы свести к минимуму ожидаемое значение времени выполнения заданий. Доказано постоянное соотношение производительности 2 для вытесняющего онлайн-стохастического планирования, чтобы минимизировать сумму взвешенных сроков выполнения на идентичных параллельных машинах.

Та же самая граница 2 для стохастического онлайн-планирования с прерываниями на однородно связанных машинах с ограниченными скоростями доказана в [92]. Основные результаты по решению проблем со случайными

временами обработки, сроками выполнения, поломками машин с учетом различных объективных функций, как регулярных, которые являются не уменьшающимися функциями времени выполнения работы, так и нерегулярных, таких как ожидаемая взвешенная трудоемкость/запоздалые сроки выполнения работы исследуются в [87]. Обсуждаются измерения производительности и рисков, ожиданий, дисперсий и стохастических заказов, которые влияют на алгоритмы планирования.

Существующие подходы к решению проблем, связанных с неопределенностью передачи данных, рассматриваются в [164]. Анализируется алгоритм планирования, который смягчает последствия возмущений во входных данных во время выполнения. Он основан на разложении графа приложения на выпуклые наборы вершин (англ., convex sets of vertices).

Другой класс проблем планирования с неопределенными параметрами рассматривается в [141]. Параметры представлены в виде векторов со всеми возможными значениями, которые могут не иметь вероятностного распределения. Производительность измеряется по критериям Minmax и Minmax regret. Бикритериальный анализ надежности и стабильности планирования в условиях неопределенности представлен в [124]. В качестве показателей надежности и стабильности авторы рассматривают суммарное ожидаемое время протекания потока и суммарную дисперсию времени завершения работ, соответственно.

Как уже обсуждалось, алгоритмы планирования облачных вычислений, как правило, делятся на два этапа: распределение нагрузки и их локальное исполнение. На первом этапе подходящая машина назначается для каждой работы по заданному критерию. В такой схеме прогнозирование времени выполнения работы и времени ожидания очереди важно для повышения эффективности ресурсов.

Точное прогнозирование времени выполнения работы является сложной задачей. Иногда сложно улучшить прогнозирование по историческим данным, исправить прогноз, отступить от прогноза и т. д. Историческая информация и применение различных методов машинного обучения, включая линейную и

квадратичную регрессию, деревья принятия решений, метод опорных векторов и ближайших соседей используются в [203], [101], [143].

Предсказание времени выполнения на основе сходства приложений, выполненных в прошлом, представлено в работах [203], [9]. Метод непараметрической регрессии, в котором оценка времени выполнения работы вычисляется по результатам прошлых наблюдений, используется в [135]. Самоподобие и характеристики "тяжелого хвоста" для создания моделей масштабируемости для высокопроизводительных кластеров применяют в [10]. Авторы формулируют задачу распределения ресурсов при наличии неопределенности времени выполнения работ и предлагают новую адаптивную стратегию распределения, названную Pareto Fractal Flow Predictor (PFFP). Рассматриваются два шага для прогнозирования времени выполнения. На первом этапе моделируется процесс организации очереди в виде агрегирования ряда самоподобных переменных для прогнозирования времени выполнения заданий в очереди. На втором этапе прогнозируется оставшееся время выполнения текущей работы на машине с использованием условной вероятности и тяжелого хвоста.

Проблемы упреждающего планирования на гетерогенных P2P системах, где число и мощность ресурсов меняются с течением времени, а алгоритмы планирования свободны от информации о характеристиках приложений анализируются в работе [16]. Авторы рассматривают планирование с репликацией работ для преодоления возможного плохого распределения ресурсов в условиях неточной информации и обеспечения хорошей производительности.

Анализируется энергопотребление стратегий распределения работ, исследуя пороговые значения репликации, а также деактивацию динамических компонентов. Основная идея подхода заключается в том, чтобы установить пороговые значения репликации и динамически адаптировать их к различным критериям оптимизации, рабочим нагрузкам и свойствам ресурсов.

Сравнены три группы стратегий планирования: не обладающие информацией, а также учитывающие скорости ресурсов и их мощности. Проведен совместный анализ двух показателей производительности, двухобъективный

оптимизационный анализ на основе оптимального набора Парето и сравнены двадцать алгоритмов с точки зрения доминирования Парето.

1.8 Выводы по первой главе

Неопределенность – это важный параметр, который влияет на эффективность вычислений, создавая дополнительные проблемы при планировании. Она требует разработки новых стратегий управления ресурсами. В этой главе рассмотрены подходы к планированию ресурсов, выполнению работ и предоставлению каналов связи в условиях неопределенности характеристик системы. Показано, что планировщик должен обеспечить возможность динамического распределения и управления ресурсами в ответ на изменяющиеся нагрузки в режиме реального времени, а также динамически адаптироваться к ним, чтобы справиться с различными потоками задач и свойствами облаков для обеспечения QoS.

Проанализированы и классифицированы неопределенности, возникающие при организации и проведении вычислений, и рассмотрены подходы к их снижению. Особое внимание уделено новым тенденциям, будущим направлениям в этой области, роли неопределенности со стороны провайдеров, пользователей и брокеров.

Рассмотрены динамические стратегии планирования ресурсов и услуг, а также программирование при наличии неопределенности. Эти проблемы являются весьма сложными и имеют ключевое значение для принятия решений по управлению ресурсами, с которыми сталкиваются пользователи/поставщики ресурсов.

Другое важное заключение состоит в выводе о необходимости рассмотрения этих проблем в свете их сопоставления с другими проблемами: стохастическим планированием, адаптивным и не требующим знаний подходами, балансировкой нагрузки и т. д.

Глава 2. ОНЛАЙН-ПЛАНИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ РАБОТ С НЕИЗВЕСТНЫМИ ПАРАМЕТРАМИ

В этой главе рассматриваются вопросы, связанные с онлайн-планированием параллельных работ с неизвестным временем выполнения. В прикладной базовой модели ГРИД состоит из большого числа процессоров, которые разделены на несколько машин разного размера. Работы являются независимыми с фиксированной степенью параллелизма и поступают одна за другой в режиме онлайн, т. е. работы поступают одна за другой без какого-либо знания о работах, которые будут поступать позже.

Работа может быть выполнена только на процессорах, принадлежащих одной и той же машине. Цель планирования – минимизировать общую продолжительность выполнения всех работ.

Покажем, что производительность списочного алгоритма планирования Гарея-Грэхэма значительно хуже в распределенных системах, чем в мультипроцессорных [43]. Далее представим алгоритм планирования, гарантирующий конкурентный фактор 5. Этот алгоритм может быть реализован с использованием подхода «кражи работ» и может хорошо подходить для использования в качестве отправной точки для построения алгоритмов планирования в реальных системах.

2.1 Введение

Изначально система ГРИД, представленная Фостером и Кессельманомв 1998 году [116], описывает использование географически распределенных ресурсов для согласованного решения проблем в виртуальных организациях. Хотя ГРИДы не ограничены вычислительными ресурсами и могут включать в себя произвольные сервисы и устройства, многие ГРИД -установки представляют

собой HPC-системы, которые обычно включают в себя параллельные компьютеры с различной производительностью.

В области параллельных и распределенных вычислений ГРИД стала распространенной технологией со многими существующими производственными инсталляциями. Они позволяют исследователям со всего мира получить прозрачный доступ к вычислительным ресурсам, предоставляемым различными провайдерами. В связи с размерами и динамической природой ГРИД, процесс выбора и распределения вычислительных работ на доступные ресурсы должен осуществляться автоматически и эффективно. Это является задачей системы планирования. Различные методы планирования были предложены и реализованы в различных производственных ГРИД. Эти системы, как правило, основаны на методах планирования для параллельных процессоров с использованием дополнительного уровня планирования [195].

В целом проблема планирования работ на мультипроцессорах хорошо изучена и является предметом исследований на протяжении десятилетий. Многочисленные результаты исследований существуют по различным вариантам этой проблемы. Некоторые из них дают теоретические знания, в то время как другие дают подсказки для реализации реальных систем. Тем не менее, планирование в распределенных системах чаще рассматривается практиками, которые ищут подходящие реализации.

Теоретических результатов по планированию в ГРИД мало. Большинство из них посвящено распределению нагрузки, см., например, [191]. В [118] обсуждается планирование последовательных независимых работ на системах с различными скоростями процессора, которые варьируются во времени и между разными машинами. Авторы утверждают, что время выполнения всех работ (makepan) как цель планирования не применима, и предлагают другой критерий, основанный на общем потреблении цикла процессора.

Производительность различных 2-ступенчатых алгоритмов в отношении критерия makepan рассматривается в [58]. Представлен алгоритмы планирования с аппроксимационным фактором 10. Модель схожа с моделью, описанной в

разделе 2.2.

В большинстве реальных задач планирования большое число и тип ограничений практически всегда мешают теоретическим исследованиям получать значимые результаты. Это особенно верно для ГРИД, которые подвержены неоднородности, динамическому поведению, различным типам работ и другим ограничениям. Поэтому модель любого теоретического исследования по планированию должна абстрагироваться от реальности. С другой стороны, должны соблюдаться ключевые свойства существующих ГРИД, чтобы обеспечить преимущества для реальных внедрений. Поскольку вычислительные ГРИД часто рассматриваются как преемники одиночных параллельных компьютеров, начнем с простой модели параллельных вычислений и расширим ее до вычислений на ГРИД.

Одной из самых базовых моделей является модель Гарея-Грэхэма [120], которая предполагает наличие мультипроцессорной системы с идентичными процессорами, а также независимых, жестких, параллельных работ с неизвестным временем обработки. Любой произвольный и достаточно большой набор одновременно доступных процессоров на одной машине может быть использован исключительно для выполнения такой работы. Как уже говорилось, эта модель не соответствует ни всем реальным системам, ни всем реальным приложениям. Но, тем не менее, эти предположения разумны. Например, параллельные компьютеры с их дорогостоящей инфраструктурой стоят инвестиций только в том случае, если они обрабатывают параллельные работы. Более того, почти все современные технологии поддерживают произвольные распределения процессоров. Несмотря на то, что между процессорами параллельного компьютера могут быть различия, касающиеся оперативной памяти или некоторых других свойств, эти процессоры в целом очень похожи.

В то время как одни приложения способны работать с разной степенью параллелизма, другие специально построены для эффективной работы на фиксированном числе процессоров. Кроме того, почти всегда существует предел эксплуатируемого параллелизма приложения. Поскольку эффективность

реализации параллельного приложения с межпроцессорным взаимодействием может сильно зависеть от других работ, выполняемых с помощью того же самого процессора, такие процессоры часто предоставляются исключительно одному приложению. Такой подход также учитывает вопросы безопасности.

Обычно на многопроцессорном компьютере работает много пользователей. Поэтому работы являются независимыми или, по крайней мере, система планирования не знает о зависимостях между этими работами. Хотя некоторые пользователи могут знать о времени обработки их заданий, однако некоторые исследования показывают, что такие пользовательские оценки ненадежны [150]. Эти наблюдения показывают, что модель Гарея-Грэхэма все еще является действительной базовой абстракцией параллельного компьютера и его приложений.

Наша модель просто расширяет эту модель, предполагая, что процессоры распределены по нескольким машинам и что параллельные работы не могут выполняться на нескольких машинах. На практике есть некоторые работы, которые используют многомашинное исполнение, но такие работы практически никогда не встречаются в производственных системах. Часто мультипроцессоры в ГРИД инсталлируются в разное время, что приводит к различным типам аппаратного обеспечения. Поэтому вычислительная система часто состоит из разнородных параллельных машин.

Однако, можно утверждать, что идентичная модель процессора все же разумна, т. к. современные процессоры в основном отличаются числом ядер, а не скоростью работы процессора. Более того, в области высокопроизводительных вычислений параллельные машины, как правило, являются относительно новыми и современными технологиями. Но даже когда игнорируются эти аргументы, все равно остаются два основных свойства ГРИД: раздельные параллельные машины и неоднородность процессора.

Обращаться к обоим свойствам в одной модели следует только после того, как достаточно хорошо рассмотрены модели с одним свойством. Неоднородность процессора уже является предметом классических машинных моделей Q_m и R_m

[183], в то время как только в [58] дают некоторые результаты для отдельной параллельной идентичной машинной модели. Поэтому в данной главе основное внимание уделено этому свойству ГРИД.

Что касается модели работы, то будем придерживаться версии Гарея-Грэхэма [177]: работы независимы и поступают одна за другой с течением времени.

Для работы характерно время ее подачи, фиксированная степень параллелизма – жесткие работы (англ., rigid jobs), а также время обработки, неизвестное до тех пор, пока задание не будет выполнено – подразумевается неклаирвоянтное расписание (англ., non-clairvoyant scheduling).

Работа может быть выполнена только на процессорах, принадлежащих одной и той же машине в режиме разделения пространства. Однако следует обратить внимание, что не требуется, чтобы задание было назначено процессорам сразу же после его подачи, как в случае с некоторыми онлайн-работами [68]. Такое требование не имеет особого смысла для планирования с неизвестным временем выполнения работ, т. к. в худшем случае это приведет к очень плохому распределению нагрузки. Более того, во многих реальных системах работы могут мигрировать между очередями, если другие машины простаивают.

С точки зрения системы, цель планировщика, как правило, заключается в достижении баланса нагрузки. В теории расписаний целью минимизации обычно является время выполнения работ. Хотя эта цель имеет некоторые недостатки, особенно в онлайн-сценариях с независимыми работами, она проста в обращении и поэтому часто используется даже в этих сценариях [68]. Таким образом, эта цель рассматривается в данной главе. Как обычно в контексте онлайн, оценивается алгоритмы планирования, определяя верхние границы конкурентных факторов, т.е. рассматриваем соотношение между продолжительностью нашего расписания и оптимальной продолжительностью.

Больше теоретических результатов известно для проблем с заданным временем выполнения работ, как в оффлайн, так и в онлайн режимах. Рассмотрим основные результаты, без подробностей, потому что они не являются целью

данной главы.

Упаковка полос (англ., strip packing) – одна из классических задач комбинаторной оптимизации. При двухмерной полосе шириной 1 и бесконечной высоте предлагается упаковать набор прямоугольных элементов в полосу таким образом, чтобы ни один элемент не перекрывался, а боковые стороны упакованных прямоугольников были параллельны боковым сторонам полосы. Цель состоит в том, чтобы минимизировать высоту полосы, используемой при упаковке элементов. Упаковка полос, как известно, является NP-сложной задачей и имеет много реальных применений.

Упаковка нескольких полос (англ., multiple strip packing) сводится к минимизации максимальной высоты, используемой между полосами для упаковки заданных прямоугольников. Следовательно, машины в ГРИД можно рассматривать как полосы, а приложения - как прямоугольники, высота и ширина которых равны, соответственно, времени работы и требуемому числу процессоров.

Однако, упаковка полосок имеет дополнительное ограничение, что прямоугольник должн быть назначена на последовательно нумерованные процессоры. Таким образом, любой алгоритм упаковки полос может быть применен для параллельного планирования работ.

Многополосная упаковка была впервые рассмотрена С. Н. Жуком [63]. Он доказал, что проблема не допускает алгоритма аппроксимации с коэффициентом производительности строго меньше 2, если только не $P = NP$, даже если есть только две полосы. Был предложен полу-онлайн алгоритм, который назначает задания на полоску в режиме онлайн, но упаковывает элементы внутри полосы после появления всех элементов (и, таким образом, на данном этапе может быть использован алгоритм упаковки полос в режиме оффлайн). Применен алгоритм назначения работ в нижний левый угол (англ., bottom-left decreasing) и показано, что конкурентный коэффициент составляет 10.

В работе [181] авторы моделируют оффлайн систему, состоящую из кластеров одинакового размера с одинаковыми процессорами, и предлагают

алгоритм с гарантированным наихудшим соотношением производительности на глобальном makepan равным 4.

В работах [78, 79] авторы рассматривают вариант той же проблемы с работами, у которых известны время выполнения и которые не требуют больших ресурсов, чем те, которые имеются на самой маленькой машине. Предлагаемый алгоритм планирования достигает соотношения 5/2. Он имеет низкую вычислительную сложность, которая может быть использована на системах реального размера.

В [80, 137], рассмотрена проблема планирования параллельных работ на гетерогенных платформах. Авторы предлагают алгоритм аппроксимации с аппроксимационным фактором 2, который является наилучшим, если только не $P = NP$. Однако 2-аппроксимация требует использования алгоритма с большим временем выполнения. Он улучшает предыдущий лучший результат с коэффициентом 2 + ε для упаковки нескольких полос [81, 102].

В [82], авторы обсуждают несколько существующих результатов и показывают, как получить 5/2- и 7/3-аппроксимации. Затем они ограничивают задачу заданиями ограниченного размера, получив алгоритм с аппроксимирующим фактором 2.

В [103], предложен глобальный централизованный механизм совместного решения, который улучшает глобальную производительность системы, когда каждая организация минимизирует время выполнения своих собственных работ. Получен фактор аппроксимации равный 3, по отношению к глобальному оптимальному времени. Показано, что эта граница является асимптотически достижимой.

В работе [218], авторы рассматривают два варианта гетерогенных кластеров: с разной шириной (числом процессоров), но одинаковыми скоростями процессора и с разной скоростью, но одинаковой шириной. Онлайн-алгоритм имеет максимальный конкурентный фактор 14.2915, для первого случая и 18.2788 для второго случая.

После этого введения сначала формально опишем модель в разделе 2.2.

Затем в разделе 2.3 покажем, что граница $2 - 1/m$ Гарея-Грэхэма, не может быть гарантирована в ГРИД никаким полиномиальным алгоритмом, если только не $P = NP$. В разделе 2.4 приведем примеры, демонстрирующие, что планировщик не может гарантировать постоянную конкурентную границу, даже если список работ отсортирован по параллелизму в порядке убывания. В разделе 2.5 предложим новый алгоритм планирования и докажем аппроксимационный фактор 3 для оффлайн случая. Наконец, этот алгоритм распространим на случай постоянного поступления работ, и докажем конкурентный фактор 5.

2.2 Онлайн планирование распределенных гетерогенных вычислительных систем

ГРИД состоит из m машин. Считаем, что машина M_iимеет размер m_i, если она содержит m_i процессоров. Все процессоры в ГРИД идентичны. Для упрощения описания предполагается, что машина индексируется так, что $m_{i-1} \leq m_i$, и $m_0 = 0$. Для описания модели используем GP_m.

Работы независимы и поступают с течением времени. Работа J_j характеризуется своим временем обработки $p_j > 0$, временем готовности, т. е. временем (моментом) поступления работы в систему $r_j \geq 0$ и фиксированной степенью параллелизма $size_j \leq s_m$, т. е. числом процессоров, которые должны быть выделены работе в процессе ее выполнения. Рассмотрим неклаирвоянтное планирование (англ., non-clairvoyant), т.е. время обработки работы становится известным только после того, как работа завершена. Далее, не допускается ни межмашинное выполнение, ни прерывания, т. е. работа J_j должна выполняться на процессорах $size_j$ одной машины без прерывания.

Введем обозначение $i = a(j)$, чтобы указать, что работа J_j будет выполнена на машине M_i. Время окончания работы J_j в расписании S обозначается $C_j(S)$. Однако, можем просто использовать C_j, если это не приводит к неоднозначности.

Расписание возможно, если $r_j + p_j \leq C_j$ для всех работ J_j и если в любое время t и для каждой машины M_i, по крайней мере используются m_i процессоров, т. е. имеем

$$m_i \geq \sum_{J_j \,|\, C_j - p_j \leq t < C_j \wedge i = a(j)} size_j$$

для каждой машины M_i.

Цель - найти расписание, которое минимизирует продолжительность makepan $C_{\max}(S) = max_j\{C_j(S)\}$. В короткой нотации трех полей α | β | γ, где α — характеристики машин; β — характеристики работ; γ — целевая функция задачи предложенной Грэмом и др. [126], эта проблема характеризуется как $GP_m|size_j\,|C_{max}$.

Оптимальная продолжительность выполнения работ обозначается как

$$C^*_{max} = \max_{legal\ schedule\ S} C_{max}(S)$$

Оценим производительность онлайн-алгоритма, определяя его конкурентный фактор или верхнюю границу для него. Здесь конкурентным фактором алгоритма *A* является максимум $\frac{C_{max}(S)}{C^*_{max}}$ для всех задач, если расписание *S* производится с помощью алгоритма *A*.

2.3 Аппроксимируемость алгоритмов планирования

В этой главе сначала рассмотрим случай одновременного поступления заявок $r_j = 0$, а затем распространим результаты на сценарий поступления заявок в разное время.

Во-первых, определим нижнюю границу для конкурентного фактора. Для этого рассмотрим соответствующую статическую проблему с полностью доступной информацией и $r_j = 0$. Эта проблема NP сложная, т. к. $P_m||C_{max}$ - особый случай проблемы $GP_m|size_j|C_{max}$. Найти хорошие алгоритмы

аппроксимации, как показано в следующей теореме, также непросто.

Теорема 2.1 *Не существует полиномиального алгоритма который всегда выдает расписания* S *с* $\frac{C_{max}(S)}{C^*_{max}} < 2$ *для* $GP_m|size_j|C_{max}$ *и всех входных данных, за исключением* $P = NP$.

Доказательство. Допустим, $m = 2$ и $m_1 = m_2$, n работ с $\sum_{j=1}^{n} size_j = 2m_1$ и $p_j = 1$ для всех работ. т. к. расписание без задержек можно легко преобразовать в расписание без задержек и увеличения времени работы [183], достаточно рассмотреть только расписания без задержек. Для данных случаев каждое расписание будет либо получать $C_{max} = 1$, либо $C_{max} = 2$. Поэтому каждый алгоритм, гарантирующий $\frac{C_{max}(S)}{C^*_{max}} < 2$, должен производить оптимальное расписание для описанных входных данных. Однако для этого требуется решение проблемы разбиения, которая в обычном смысле является NP сложной.

2.4 Списочные алгоритмы планирования

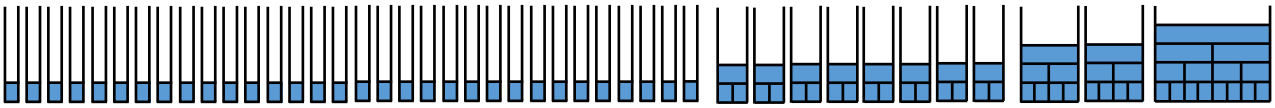
В соответствии с Теоремой 2.1, граница списочного алгоритма $2 - \frac{1}{m}$, см. Гарея-Грэхэма [121] (одновременная подача работ), а также Naroska и Schwiegelshohn [177] (онлайн планирование), к ГРИД не применима. Более того, следующий пример показывает, что уже в случае одновременного поступления работ, списочное расписание не может гарантировать постоянную границу для $\frac{C_{max}}{C^*_{max}}$ во всех случаях решения проблемы. Следует отметить, что аналогичный пример уже был приведен в [58].

Пример 2.1 Пусть $k > 1$ будет целым числом. В нашей системе предполагаем одну машину M_m с $m_m = 2^k$ процессорами, и $2 \cdot 4^{k-1}$ одинаковых машин с процессорами $2^{k-\kappa}$ для каждого κ с $1 \leq \kappa \leq k$. В результате получим в общей сложности 2^{2k} процессоров и $m = 1 + 2\frac{4^k - 1}{3}$ машин в системе.

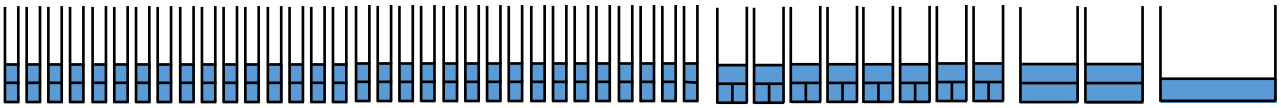
Кроме того, у нас есть $2^{2(k-\kappa)}$ работ с $size_j = 2^k$ для всех $0 \leq \kappa \leq k$, в результате чего общее число работ составит $\frac{4^{k+1}-1}{3}$. У всех этих работ $p_j = 1$.

Предположим, что работы отсортированы по параллелизму в порядке возрастания. Тогда расписание будет начинаться со всех работ с $size_j = 1$ одновременно на всех машинах. В момент 1 все работы с $size_j = 2$ начинают обработку на всех машинах с $m_i \geq 2$, в то время как все машины с $m_i = 1$ должны оставаться в режиме ожидания. Процесс повторяется до тех пор, пока, наконец, последняя работа с $size_j = 2^k$ не начнется в момент времени k на машине M_m с получением $C_{max} = k + 1$ (рис. 2.1а).

Однако, если список отсортирован по параллельности в порядке убывания, то каждое задание J_j присваивается машине M_i таким образом, что размер и параллельность совпадают $size_j = m_{a(j)}$. В результате получается оптимальное время $C^*_{max} = 2$ (рис. 2.1б).



(a) Списочное расписание



(b) Оптимальное расписание

Рисунок 2.1 – Расписания примера 4.1: (a) наихудшее списочное расписание (b) оптимальное расписание

Очевидно, что этот результат обусловлен дисбалансом нагрузки, т. к. машины с большим числом процессоров выполняют работы с небольшим параллелизмом, что приводит к вынужденному ожиданию выполнения параллельных работ. Это наблюдение предполагает сортировку списка по параллелизму работ в порядке убывания, так чтобы высокопараллельные работы

планировались первыми, когда есть выбор.

К сожалению, такой подход также не гарантирует постоянного конкурентного фактора. Для пояснения достаточно сложного примера отдельно рассмотрим следующие два случая. Пример 2.2 показывает, что планирование списка все еще может препятствовать одновременному выполнению нескольких параллельных работ на одной машине, даже если список отсортирован по параллелизму работ в порядке убывания.

Пример 2.2 Рассмотрим машину M_i таким образом, что μ это число процессоров на самой большой машине с меньшим числом процессоров, чем m_i.

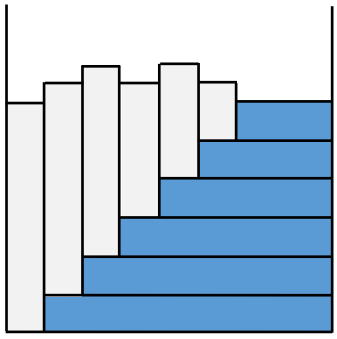
Для каждого k с $\mu + 1 \leq k \leq m_i - 1$, у нас одна работа J_j с очень малым временем обработки $(p_j \to 0)$ и $size_j = k$. Небольшое время обработки гарантирует, что выполнение таких параллельных работ не займет много времени, хотя они должны выполняться одно за другим. Кроме того, существует большое число последовательных работ $size_j = 1$ с различным временем обработки. Т. к. рассматривается неклаирвоянтное планирование, последовательные работы не могут быть распознаны во время планирования. Для худшего случая предполагаем, что время обработки последовательных работ достаточно велико, чтобы каждая последовательная работа завершалась после завершения последнего из вышеупомянутых параллельных работ.

Списочное расписание с сортировкой по параллельности в порядке убывания начнется с $size_j = m_i - 1$ в момент времени 0. В то же время запускается последовательная работа, т. к. на эту машину больше не помещается ни одна параллельная работа. После завершения параллельной работы алгоритм одновременно запускает работы с $size_j = m_i - 2$ и другую последовательную работу. Этот процесс продолжается и создается расписание, в котором работа с $size_j = \mu + 1$ выполняется одновременно с последовательными работами $m_i - \mu - 1$ (рисунок 2.2).

Эта ситуация не изменится позже, если не будет завершено, по крайней мере, две последовательных работы одновременно. Из-за малого времени

обработки параллельных работ выполнение этого расписания занимает очень мало времени. Если имеется несколько машин с одинаковым числом процессоров, то просто умножаем работы соответственно.

Основываясь на примере 2.2, теперь можно рассмотреть пример 3.3, где планирование списков основано на сортировке по параллелизму в порядке убывания, а на каждой машине M_i с $m_i > 1$, почти одновременно, запускается одно параллельное задание с несколькими последовательными работами.



$S_i = 9$

Рисунок 2.2 – Расписания примера 4.2

Пример 2.3 Пусть $k > 2$ будет любым целым числом. В нашей системе содержится k различных типов машин, например, b_κ машин размером $m_\kappa = 2^{\frac{(\kappa+2)(\kappa-1)}{2}}$ на каждую $1 \le \kappa \le k$. Говорим, что эти машины имеют тип κ. Далее, для каждой $1 \le \kappa \le k$ существует κ рабочих мест типа κ, т.е. эти работы имеют параллельность $size_\kappa = 2^{\frac{\kappa(\kappa-1)}{2}}$.

Заметим, что

$$\frac{m_\kappa}{size_\kappa} = \frac{2^{\frac{(\kappa+2)(\kappa-1)}{2}}}{2^{\frac{\kappa(\kappa-1)}{2}}} = 2^{\kappa-1}$$

Работы типа κ могут выполняться одновременно на машине типа κ. Время обработки всех работ составляет около 1. Однако, время обработки выбирается таким образом, чтобы не было двух работ, выполненных в одно и то же время на одной и той же машине в списочном расписании S.

В оптимальном расписании на машинах типа κ выполняются только работы

типа κ. Это дает $C^*_{max} \approx 2$ если $a_\kappa \leq 2b_\kappa \cdot \frac{m_\kappa}{size_\kappa} = b_\kappa \cdot 2^\kappa$ справедливо для всех $1 \leq \kappa \leq k$ и если есть хотя бы одна κ с $a_\kappa > b_\kappa \cdot 2^{k-1}$.

В расписании S на каждой машине типа κ запускается только одна работа типа κ примерно в момент времени t, при этом $t < \kappa - 1$ является неотрицательным целым числом. Примерно в это же время на всех машинах типа κ и выше запускаются все оставшиеся работы типа κ таким образом, что на любой такой машине не $size_\kappa$ процессоров простаивают для $\kappa \neq k$ (рисунок 2.3). В этом расписании $C_{max}(S) \approx k$.

Наконец, нам необходимо определить соответствующие значения для a_κ и b_κ. Для этого используем обратную рекурсию: $b_k = 1$ и $a_k = 2^{k-1} + k - 1$. Для $1 \leq \kappa < k$, выбираем

$$b_k = \left\lceil \frac{\sum_{h=k+1}^{k} b_h \cdot (m_h - size_h)}{m_k - size_k(k-1)} \right\rceil$$

$$a_k = \frac{\sum_{h=k+1}^{k} b_h \cdot (m_h - size_h)}{size_k} + b_k \left(k - 1 + \frac{m_k}{size_k} \right)$$

Обратите внимание, что такой выбор всегда возможен, т. к. у нас есть $2^i - i \geq 1$ для всех не отрицательных целых i. Далее, $b_\kappa \frac{m_\kappa}{size_\kappa} < a_\kappa \leq 2b_\kappa \frac{m_\kappa}{size_\kappa}$ справедливо для всех $1 \leq \kappa \leq k$.

В таблице 2.1 и таблице 2.2 приведены числа для $k = 3$ и $k = 4$ соответственно. Т. к. этот пример уже требует очень большого числа машин и работ даже для небольшого k, он в основном представляет теоретический интерес. Также следует обратить внимание, что этот алгоритм не является распределенным.

Таблица 2.1 – Параметры примера 2.2 с $k = 3$

K	1	2	3
$size_k$	1	2	8
m_k	1	4	32
a_k	112	56	6
b_k	56	14	1

Таблица 2.2 – Параметры примера 2.3 с $k = 4$

K	1	2	3	4
$size_k$	1	2	8	64
m_k	1	4	32	512
a_k	4,480	2,240	224	11
b_k	2,240	560	28	1

Пример 2.3 показывает, что может быть трудно определить фиксированный порядок в списке работ, чтобы гарантировать постоянный конкурентный фактор для проблемы планирования ГРИД. Поэтому планирование распределенных систем сложнее, чем многопроцессорное планирование.

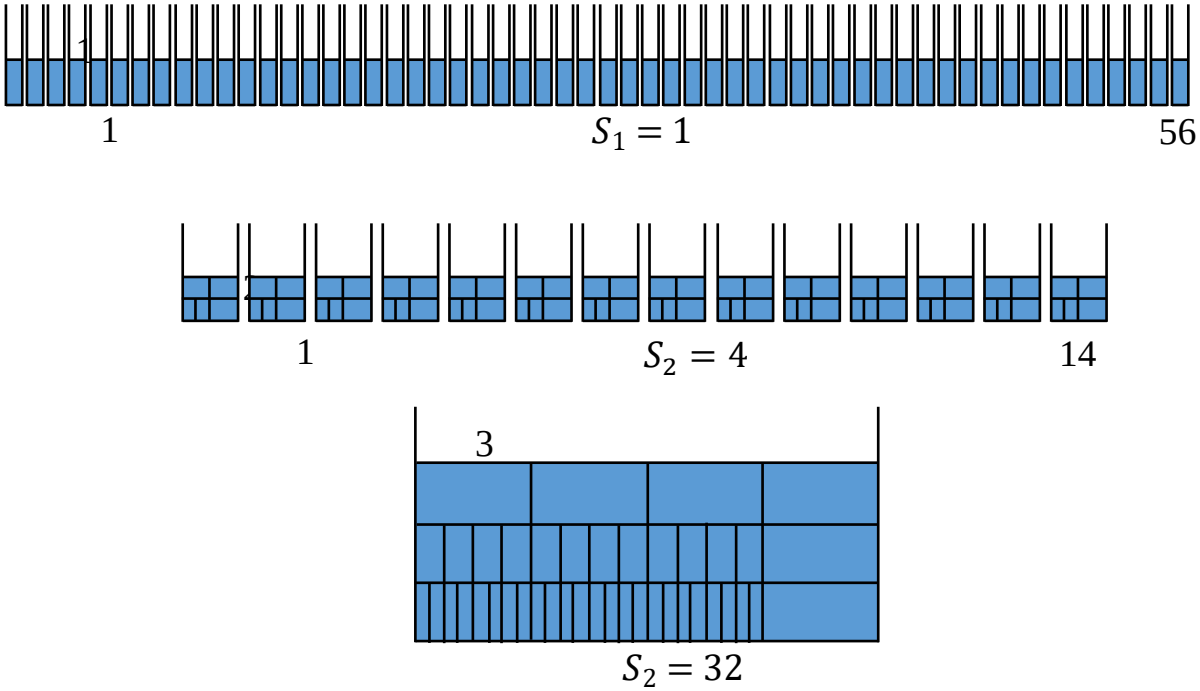
2.5 Алгоритм распределения ресурсов в распределенных системах

Поскольку обычное списочное расписание не подходит для ГРИД, представим планировщик, использующий несколько списков. Каждый из этих списков не требует какого-либо конкретного порядка. Начнем с использования общеизвестной нижней границы для оптимальной продолжительности в случае одновременной загрузки работ для решения проблемы планирования ГРИД:

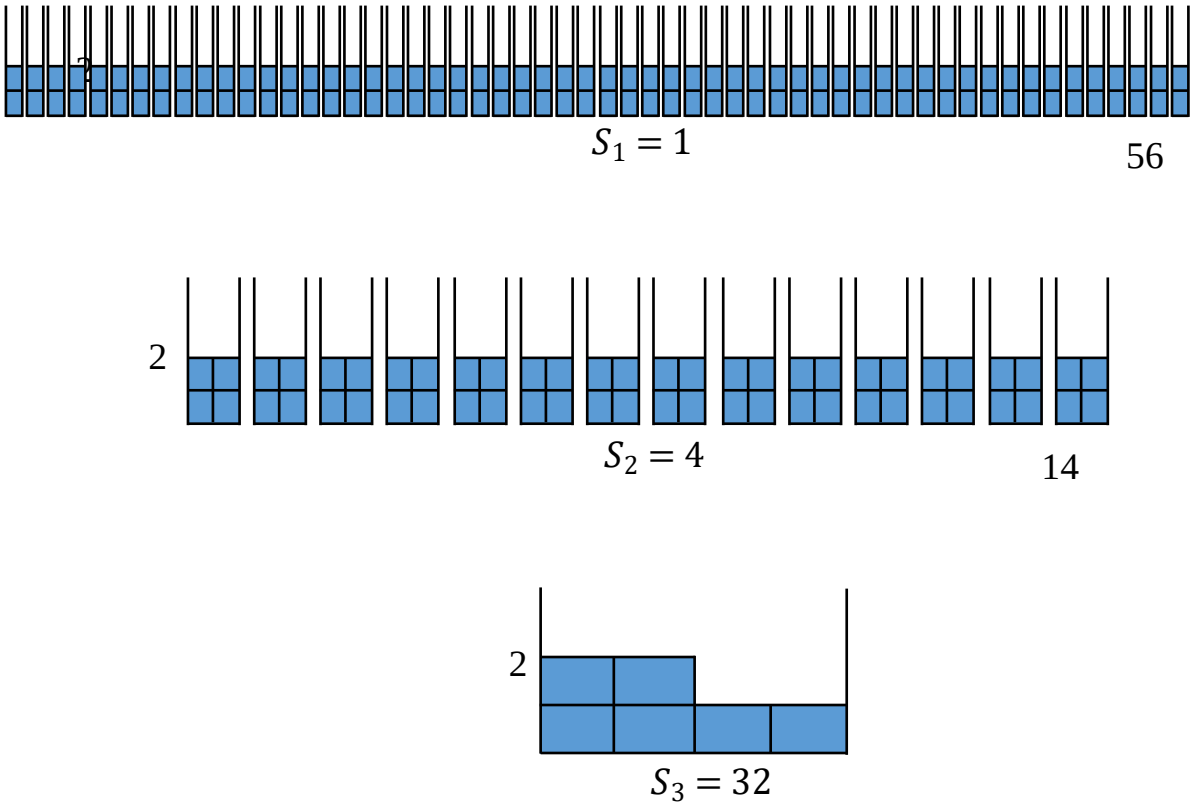
$$C^*_{max} \geq max\left\{\max_j p_j, \max_{1\leq i\leq m} \frac{\sum_{j\,|\,size_j > m_{i-1}} p_j \cdot size_j}{\sum_{v=i}^{m} m_v}\right\}$$

По сравнению с границей проблемы $P_m||C_{max}$, эта граница также учитывает недоступность машин небольшого размера для обработки высокопараллельных работ из-за отсутствия их многосайтового выполнения.

Алгоритм планирования ГРИД основан на различных первоначальных распределениях работ на различных машинах. Эти распределения представлены с помощью категорий работ, назначаемых для каждой машины.



(а) Списочное расписание



(б) Оптимальное расписание

Рисунок 2.3 – Расписания примера 2.3 для k=3: a) расписание худшего случая b) оптимальное расписание

Определение 2.1 Для каждой машины M_i существует три различные категории работ:

1) $A_i = \left\{ J_j \,\middle|\, max\left\{\frac{m_i}{2}, m_{i-1}\right\} < size_j \leq m_i \right\}$

2) $B_i = \left\{ J_j \,\middle|\, m_{i-1} < size_j \leq \frac{m_i}{2} \right\}$

3) $H_i = \left\{ J_j \,\middle|\, \frac{m_i}{2} < size_j \leq m_{i-1} \right\}$

Набор A_i содержит все работы, которые не могут быть выполнены на предыдущей (следующей меньшей) машине и требуют более 50% процессоров машины M_i. Набор B_i содержит все работы, которые не могут быть выполнены на предыдущей машине, но требуют не более 50% процессоров машины M_i. Набор H_i содержит все работы, которые требуют более 50% процессоров машины M_i, но также могут быть выполнены на предыдущей машине.

Заметим, что для каждой работы J_j существует ровно один индекс i с $m_{i-1} < size_j \leq m_i$, т. к. m_i упорядочены. Если $m_{i-1} \geq \frac{m_i}{2}$, то $J_j \in A_i$, в противном случае job J_jнаходится в A_i или в B_i. Поэтому каждая работа J_jпринадлежит ровно к одной категории A или B.

Очевидно, что-либо $B_i = \emptyset$ либо $H_i = \emptyset$ выполняется для каждой машины M_i. Работа в категории B_i не может принадлежать какой-либо другой категории, в то время как работа в категории H_i должна также принадлежать либо категории A_{i-1}, либо категории H_{i-1}. Поэтому $H_i \cap H_{i-1} \neq \emptyset$ требует $A_{i-1} \subseteq H_i$.

Далее на рисунке 2.4 представлен алгоритм планирования ГРИД для случая одновременной подачи работ. Этот алгоритм использует основной список L_i и список поддержки S_i для каждой машины M_i. Список L_iсодержит все работы, которые готовы к планированию на машине M_i, в то время как список S_i просто отслеживает те работы в H_i, которые еще не были перенесены в L_i. Обратите внимание, что работа может находиться в списках нескольких машин одновременно.

Algorithm 2.1. Grid Concurrent-Submission

```
1   for i ← 1 to m do
2       L_i ← A_i
3       S_i ← H_i
4   endfor
5   Update
6   repeat
7       for i ← 1 to m do
8         while enough processors are idle on machine i do
9             schedule a job from L_i on machine i
10            remove the scheduled job from all lists L_k
11            if any list L_k = ∅
12                Update
13            endif
14        endwhile
15      endfor
16  until all jobs are scheduled
```

Рисунок 2.4 – Алгоритм планирования ГРИД для случая одновременной подачи работ ($r_j = 0$)

Процедура $Update$ на рисунке 2.5 является ключевым компонентом алгоритма. Он поддерживает списки различных машин. Если на машине M_i ($L_i = \emptyset$) нет работ, готовых к выполнению, то на машине M_i включены работы из H_i, если они уже доступны для расписания на машине M_{i-1}. Если работ такого типа не существует ($H_i = \emptyset$) и ни одна работа не готова для выполнения на машине M_i, то все работы в B_i включены для планирования на машине M_i. Обратите внимание, что при использовании последовательной нотации программы важно обрабатывать эти наборы в порядке возрастания индексов машин.

Предположим, что время обработки этой процедуры не вносит в план дополнительных простоев. Также следует помнить, что целью данной главы является не презентация эффективной реализации процедуры, а демонстрация алгоритмической концепции.

Рассмотрим работу j, которая находится в H_i и A_{i-1}. Алгоритм помещает

эту работу в L_{i-1} при старте. Процедура Update введет ее в L_i, как только все работы из A_i будут запланированы, если только j уже не запустилась. Работа $j \in H_{i+1} \cap A_{i-1}$ станет элементом L_{i+1} после того, как будут запланированы все работы из A_{i+1} и $A_i \cap H_{i+1}$ и т. д. Процедура Обновление гарантирует, что список L_i не будет пустым, если работы есть в любом списке $L_{i'}$ c $i' < i$.

Алгоритм Concurrent-Submission на рисунке 2.4 сначала инициализирует списки L_i и S_i для всех машин. Т. к. для некоторых машин список L_i может быть пустым, обновление процедур также вызывается один раз сразу после этого. Позднее, процедура Update вызывается снова, если на какой-нибудь машине была запущена последняя работа списка L_i.

Далее покажем, что алгоритм Concurrent-Submission предотвращает промежуточные интервалы расписания, когда большинство процессоров машины простаивают.

Лемма 2.2 *Алгоритм Concurrent-Submission гарантирует, что на каждой машине более 50% процессоров всегда заняты выполнением работ до начала выполнения последней работы на этой машине.*

Доказательство. Предположим, что у нас есть

– как минимум $\frac{m_i}{2}$ простаивающих процессоров на какой-то машине M_i в некоторое время t и

– работа J_j c $C_j - p_j > t$ u $a(j) = i$.

Помните, что сначала на машине M_i запланированы работы с A_i и H_i. Поэтому более $\frac{m_i}{2}$ процессоров всегда заняты на машине M_i до тех пор, пока все работы из A_i и H_i не будут завершены. Кроме того, ни один из заданий, требующих в большинстве случаев $\frac{m_i}{2}$ процессоров, не может быть в списке L_i в то же время t, т. к. достаточно процессоров простаивают, чтобы запустить это задание немедленно. Следовательно, список L_i должен быть пустым в момент времени t. Т. к. работа J_jвсе еще не запланирована в момент времени t она должна принадлежать $A_{i'} \backslash H_i$ или $B_{i'}$ для какой-то машины $M_{i'}$ с $i' < i$, в

результате чего$L_{i'} \neq \emptyset$ в момент времени t. Однако процедура Update гарантирует, что список L_iне будет пустым, если в каком-либо списке $L_{i'}$ с $i'' < i$. Это противоречие.

Algorithm 2.2. Update

```
1   for i ← 1 to m do
2     if L_i = ∅
3       if i ≠ 1 and S_i ≠ ∅
4         L_i ← L_{i−1} ∩ S_i
5         S_i ← S_i \ L_i
6       Elseif B_i ≠ ∅
7         L_i ← B_i
8       endif
9       if i ≠ 1 and L_{i−1} ≠ ∅ and S_i = ∅
10        L_i ← L_{i−1}
11      endif
12    endif
13  endfor
```

Рисунок 2.5 – Обновление списков для случая одновременной подачи работ

К сожалению, примеры 2.1 и 2.3 показывают, что Лемма 5.2 недостаточна для доказательства постоянного конкурентного фактора. Кроме того, необходимо сбалансированное "использование" машин в системе. Лемма 5.3 показывает, что в случае несбалансированного использования ресурсы высокопараллельных машин не тратятся впустую для выполнения работ с небольшим параллелизмом.

Лемма 2.3 Пусть $J_{j'}$ будет любой работой с $C_{j'}(S) = C_{max}$ в расписании S, производимом алгоритмом Concurrent-Submission. Если есть машина M_i с $i < a(j')$ и как минимум $\frac{m_i}{2}$ процессоры находятся в режиме ожидания $t < C_{j'} - p_{j'}$, то при одновременной отправке по алгоритму Concurrent-Submission будет получено расписание S' с одним и тем же makespan ($\left(C_{max}(S') = C_{max}(S)\right)$, если все задания $A_{i'}$ и $B_{i'}$ с $i' \leq i$ будут удалены.

Доказательство. Утверждение верно, если ни одна машина M_k с $k > i$ не выполняет работу из множеств $A_{i'}$ or $B_{i'}$ с $i' \leq i$.

Предположим, что в множестве $A_{i'}$ или $B_{i'}$ с $i' \leq i$ есть работа J_j и что эта работа выполняется на машине M_k с $k > i$ Благодаря процедуре Update, работа J_j может быть включена в L_k только для некоторых $i < \kappa \leq k$, если все работы с A_κ и B_κ уже запланированы. Поэтому удаление всех работ в категориях $A_{i'}$ and $B_{i'}$ с $i' \leq i$ не повлияет на время выполнения любой работы в наборах A_κ и B_κ с i < к ≤ k.

Наконец, для некоторых машин M_v с $v > i$ есть $J_{j'} \in A_v$ или $J_{j'} \in B_v$, т. к. работа $J_{j'}$ не стартовала на машине M_i в момент времени t. Поэтому удаление всех работ в $A_{i'}$ и $B_{i'}$ с $i' \leq i$ не может уменьшать $C_{max}(S)$.

Лема доказана

Теперь докажем конкурентный фактор алгоритма одновременной подачи данных.

Теорема 2.2 *Алгоритм одновременной подачи данных гарантирует* $\frac{C_{max}}{C^*_{max}} <$ 3*, для всех входных данных и всех конфигураций ГРИД в случае одновременной подачи.*

Доказательство. Пусть $J_{j'}$ будет любой работой с $C_{j'}(S) = C_{max}$ в расписании S, выданном алгоритмом Concurrent-Submission. Если есть машина M_i с $i < a(j')$ и как минимум простаивающими $\frac{m_i}{2}$ процессорами до $C_{j'} - p_{j'}$, то можно удалить все работы в множествах $A_{i'}$ и $B_{i'}$ с $i' \leq i$. Это не уменьшает соотношение $\frac{C_{max}}{C^*_{max}}$, т. к. C_{max} остается неизменным из-за Лемма 5.3 и C^*_{max} не может увеличиться. При необходимости можно выполнить этот процесс повторно.

На основании Леммы 2.3 предположим, что есть некоторое время $t < C_{j'} - p_{j'}$, когда не более 50% процессоров некоторой машины $M_{i'}$ с $i' > a(j')$ выполняют задания в момент времени t в S, и что все машины $M_{i'}$ с $i' < k$ могут быть проигнорированы для анализа, т. к. они не могут выполнять работы из A_k и B_k для $\kappa > k$.

Тогда у нас есть $L_{i'} \neq \emptyset$ и $L_{a(j)} \neq \emptyset$ в момент t. Опять же, это невозможно в связи с выполнением процедуры *Update*.

Это приводит к тому, что

$$C_{max}(S) = p_{j'} + C_{j'} - p_{j'} < p_{j'} + 2 \cdot \frac{\sum_{J_{j \in A_v \cup B_v} | v \geq k} p_j \cdot size_j}{\sum_{v \geq k} m_v} \leq C^*_{max} + 2C^*_{max}$$

$$= 3C^*_{max}$$

Теорема доказана

Однако результат Теоремы 2.2 не является достижимым (tight). Интуитивно понятно, что достижимость требует наличия планирования, в котором используется только 50% процессоров, в то время как оптимальное расписание выполняет те же работы без простоя процессоров. Кроме того, в конце расписания должна быть еще одна длительная работа, которая не может начаться раньше. Мы не можем найти такой пример, но можем представить пример, показывающий, что этот алгоритм производит конкурентный фактор, произвольно приближающийся к 2.5 для некоторых входных данных и конфигураций ГРИД.

Пример 2.4 Рассмотрим простую систему с $m = 2$, $m_1 = 1$, и $m_2 = 21$. В A_1 7 работ: 6 одинаковых с $size_j = 1$ и $p_j = 1$, и последняя работа в этом списке с $size_j = 1$ и $p_j = 4$.

В A_2 есть 2 работы: первая работа с $size_j = 11$ и $p_j = 3$, и последняя работа с $size_j = 11$ и $p_j = \epsilon \to 0$. Наконец, B_2 содержит работу с $size_j = 8$ и $p_j = 3$, затем три идентичных работы с $size_j = 7$ и $p_j = 1$, а в конце - одну работу с $size_j = 8$ и $p_j = \epsilon \to 0$. Обе работы с $p_j = \epsilon$ нужны только для того, чтобы предотвратить преждевременное опустошение A_2 и B_2.

В расписании S все работы в A_1 назначены на машину M_1. Самая длительная работа начинается последней в момент времени 6 и определяет продолжительность $C_{max}(S) = 10$. Это возможно только в том случае, если L_2 не становится пустым до времени 6. Продолжительная работа в A_2начинается в момент времени 0, а за ней сразу следует другая. Таким образом, B_2 становится L_2 в момент времени 3, а первая работа B_2 начинается в момент времени 3 и

завершается в момент времени 6. Следующие три работы B_2 выполняются одновременно с первой работой и запускаются в моменты времени 3, 4 и 5 соответственно, а последняя запускается в момент времени 6, см. рисунок 2.6. Таким образом, L_2 становится пустым в момент времени 6.

Оптимальное расписание легко собрать с помощью $C^*_{max} = 4 + \epsilon$ (рисунок 2.6). При этом получается отношение

$$\lim_{\epsilon \to 0} \frac{C_{max}(S)}{C^*_{max}} = \lim_{\epsilon \to 0} \frac{10}{4 + \epsilon} = 2.5$$

Более сложный пример дает более низкую границу для конкурентного фактора алгоритма Concurrent-Submission, которая произвольно приближается к 21/8 = 2,625.

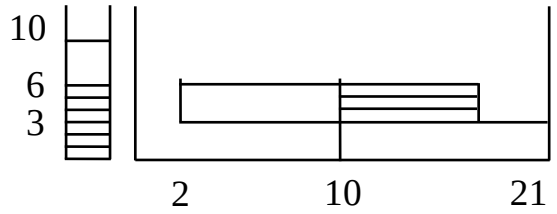
2.6 Общая проблема онлайн планирования

Наконец, рассмотрим общую проблему. Для решения этой задачи можно модифицировать алгоритм Concurrent-Submission, используя результаты работы [197]. Это увеличит конкурентный фактор в 2 раза и приведет к $\frac{C_{max}(S)}{C^*_{max}} < 6$.

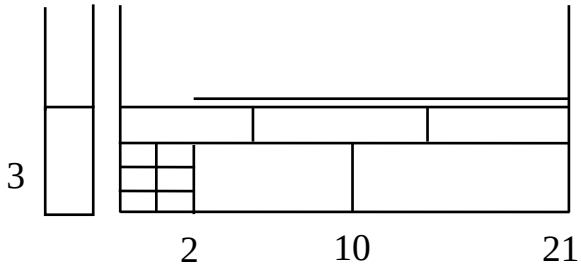
Однако на практике такой алгоритм вряд ли приемлем, т. к. требует от вновь подаваемых работ значительного времени ожидания, даже если работа не очень параллельна и имеется достаточное число процессоров. Поэтому рассмотрим простую модификацию алгоритма Concurrent-Submission, который генерирует лучший конкурентный фактор и может быть более значимым на практике.

Сначала меняем множества A, B и H со статических на динамические. Как только работа запланирована, она удаляется из всех подмножеств и списков. С другой стороны, любая новая поданная работа вводится в соответствующие множество. Обратите внимание, что разница между H_i and S_i становится меньше, но она все еще существует, т. к. работа удаляется из H_i после планирования и из

S_i после внесения в L_i, соответственно.



(а) ГРИД расписание



(б) Оптимальное расписание

Рисунок 2.6 – Расписания для примера 5.5: a) расписание алгоритма для параллельной подачи работ, b) оптимальное расписание

Во-вторых, модифицируем алгоритм Concurrent-Submission, удаляя все списки L_i и S_i всякий раз, когда подается новая работа, и перезапускаем процедуру Update в целях инициализации. Результирующий метод называется Over-Time-Submission. Конечно, подходящая реализация может избежать некоторых из этих модификаций списка и выполнить другие модификации эффективно. Но в этой главе не будем рассматривать эффективность реализации.

Анализ Over-Time-Submission в значительной степени основан на анализе *Concurrent-Submission* приведнnном в разделе 2.5.

Теорема 2.3 *Алгоритм Over-Time-Submission гарантирует* $\frac{C_{max}}{C^*_{max}} < 5$ *для всех входных данных в случае подачи работ во времени.*

Доказательство. Предположим, что r это время подачи последней работы в расписании S. По истечении времени r применяются механизмы Over-Time-Submission. Однако, если раньше подача работ по алгоритму Concurrent-Submission начиналась с того, что все процессоры простаивали, то теперь

процессоры могут быть заняты выполнением некоторых работ, которые были поданы ранее. Работам из множеств A или H, возможно, придется подождать, пока будет доступно достаточное число процессоров. В течение этого времени $\frac{m_i}{2}$ и более процессоров могут находиться в режиме ожидания на машине M_i. Но т. к. все списки L_i были опустошены, этот промежуток времени ограничен максимальным временем обработки любой работы. Поэтому условия Леммы 2.2 и 2.3 применяются не позднее времени $r + max_j p_j$. В качестве границ уравнения (1) и неравенства $r < C^*_{max}$ выполняются в случае over-time-submission и у нас есть

$$C_{max}(S) < r + \max_j p_j + 3C^*_{max} < 5C^*_{max}$$

Как и в случае с алгоритмом Concurrent-Submission, граница теоремы 2.1 не является жесткой. Но можно показать, что нижняя граница конкурентного фактора алгоритма Over-Time-Submission близка к 4.5.

2.7 Выводы по второй главе

В этой главе представлена модель распределенной системы, которая охватывает основные, на наш взгляд, свойства ГРИД систем. На основе этой модели проанализирована фундаментальная проблема, которая была получена из одной из самых ранних и основных проблем многопроцессорного планирования. На наш взгляд, это первый всесторонний теоретический анализ планирования в ГРИД. Показано, что планирование ГРИД является более сложной задачей, чем соответствующее многопроцессорное планирование, и что хорошо известный планировщик не может гарантировать постоянного конкурентного фактора, хотя в случае с многопроцессорными системами он работает очень хорошо. Этот результат сохраняется даже в том случае, если список отсортирован по параллельности работ в порядке убывания.

Приведенные примеры показывают, что никакой статический порядок сортировки списков, основанный на параллелизме работ, не может гарантировать

хорошего планирования ГРИД, не зависящего от конфигурации ГРИД.

Представлен новый алгоритм, который основывается на нескольких списках и гарантирует конкурентное отношение 3 в сценарии с одновременной подачей всех работ. Т. к. рассматриваются неклаирвоянтные работы, эта проблема также имеет некоторые онлайн-свойства.

Затем этот алгоритм расширен на общий сценарий подачи работ во времени, в результате чего конкурентный фактор составляет 5. Насколько известно, впервые для данного типа задач доказано наличие постоянного конкурентного фактора.

Хотя в этой главе не рассматриваются детали реализации алгоритмов, отметим, что значительная часть предложенного алгоритма может быть реализована распределенным образом: Каждая машина имеет свои собственные списки работ и планирует только их. Первоначально каждая работа распределяется ровно по одному списку A_i или B_i. Предполагая глобальную информационную систему ГРИД, это может быть достигнуто с помощью простого подхода «Мастер-Ведомый». Если в списках нет работ, то эта машина начинает использовать список соседней машины, т.е. машина может "украсть" работу у соседа. Но динамическая информация о наличии работ все равно должна быть разделена между несколькими машинами. Алгоритм кражи работ у соседней машины кажется подходящим для реализации в реальных системах. Но в таких реальных системах необходимо учитывать и другие свойства вычислительных систем таких, например, как топология и пропускная способность. Тем не менее, предлагаемый алгоритм может служить отправной точкой для построения эвристических алгоритмов планирования, которые реализуются в реальных вычислительных системах.

Глава 3. АДАПТИВНОЕ ПЛАНИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ КОНЦЕПЦИИ ДОПУСТИМОГО РАСПРЕДЕЛЕНИЯ РАБОТ

В этой главе анализируются алгоритмы планирования работ, интегрирующие обе задачи планировщика ГРИД: распределение работ по машинам и их локальное расписание. Предложена и проанализирована адаптивная схема распределения работ с использованием концепции допустимого распределения (англ., admissible allocation) [17, 21]. Основная идея этой схемы заключается в том, чтобы установить ограничения по распределению работ и динамически адаптировать эти ограничения к различным рабочим нагрузкам и свойствам системы. Этот подход приводит к снижению возможного дисбаланса нагрузки, когда машины с большим числом процессоров могут выполнять работы с небольшим параллелизмом, приводя к вынужденному ожиданию выполнения параллельных работ.

Разработаны 3-х аппроксимационный и 5-ти конкурентный алгоритмы, названные MLBa PS и MCTa PS для случая, когда все работы могут быть выполнены на самой маленькой машине, и 9-ти аппроксимационный и 11-ти конкурентный алгоритм для общего случая.

Чтобы показать практическую применимость методов, проведено комплексное исследование практической эффективности предложенных стратегий и их производных с использованием моделирования (глава 3.5) [21]. Для этого использовались трассы реальной рабочей нагрузки и соответствующие конфигурации ГРИД. Проанализированы девять стратегий планирования, которые требуют разного объема информации по трем сценариям ГРИД. Продемогстрировано, что предложенные стратегии хорошо работают по десяти показателям, которые отражают как пользовательские, так и системные цели.

3.1 Введение

Планирование является важным аспектом достижения высокой производительности распределенных систем. Различные алгоритмы планирования уже предложены и внедрены в разные типы систем. Однако в этой области все еще остается много нерешенных вопросов. Одной из основных проблем является разработка скоординированных механизмов обеспечения ресурсами, которые позволяют более эффективно использовать возможности и повышать качество обслуживания.

В целом проблема планирования работ на мультипроцессорах хорошо изучена и является предметом исследований на протяжении десятилетий рассматривая теоретические аспекты или подсказки для реализации реальных систем.

Планирование же в ГРИД почти исключительно рассматривается специалистами-практиками, которые ищут подходящие реализации. Существуют лишь очень немногие теоретические результаты по планированию в ГРИД, которые представлены в главе 2 и которые рассмотрим далее в разделе 3.3.

В этой главе проанализируем базовую двухуровневую модель ГРИД. Самый высокий уровень состоит из планировщика ГРИД (ресурсного брокера или мета-планировщика), который получает запросы на работы и распределяет работы на подходящий ресурс ГРИД. Управление конкретным ресурсом это задача локальной системы управления, которая знает текущее состояние своей машины и назначенных ей работ. Локальное планирование применяется независимо к каждой машине. На каждом уровне учитываются различные ограничения и спецификации. Здесь рассмотрены параллельные работы, которые имеют заданную степень параллелизма и во время выполнения которых к ним должны быть закреплены исключительно заданное число процессоров или ядер.

Политика допустимого распределения, которая исключает определенные машины из набора машин, доступных для распределения определенной работы

была предложена в [17].

В данной главе проводится теоретический анализ двух алгоритмов планирования, названных Min-Lower-Bound with Admissible Job Allocation (MLBa) и Min-Completion-Time with Admissible Job Allocation (MCTa), с учетом заданных ограничений и критерия оптимизации времени выполнения, т. е. рассматривается минимизация наибольшего времени выполнения любой работы в системе [21].

В частности, показано, что makespan полученный предложенными алгоритмами не превышает оптимальную продолжительность работы более чем на определенный фактор. Этот фактор называется конкурентным в случае онлайн планирования и аппроксимационным фактором в случае оффлайн, соответственно.

В [51] показано что конкурентный фактор алгоритма MLBa+PS колеблется от 17 до бесконечности при изменении коэффициента допустимости.

В [21] улучшаются и расширяются известные результаты [51] достигая аппроксимационного фактора 9 для случая оффлайн и конкурентного фактора 11 для случая онлайн планирования. Также получены аппроксимационный фактор 3 и конкурентный фактор 5 для случаев, когда все рабы подходят к самой маленькой машине, как в проблеме, обсуждаемой в [78].

Чтобы показать практичность и конкурентоспособность алгоритмов, проведено комплексное исследование их производительности с использованием моделирования в [21]. Использованы рабочие нагрузки на основе реальных производственных систем при рассмотрении трех сценариев ГРИД на основе гетерогенных HPC-систем и исследовании проблемы планирования работ с неопределенными требованиями по времени выполнения, т. к. в реальной среде выполнения доступны только оценки времени выполнения работ пользователями.

Алгоритмы ориентированы на параллельные работы с интенсивными вычислениями и принимают решения по планированию без точной информации о производительности, в частности, о времени выполнения работ. Они просты, работают по принципу одна работа обрабатывается за другой (англ. "job-by-job") и позволяют эффективно реализовывать их в реальных системах.

3.2 Политика допустимого распределения работ

Проблема онлайн планирования определяется следующим образом: n параллельных работ $J_1, J_2, \dots, J_n$ должны быть назначены на m параллельных машин $N_1, N_2, \dots, N_m$. Пусть m_i будет число одинаковых процессоров машины N_i, также называемых размером машины N_i. Пусть $s_{f,l} = \sum_{i=f}^{l} m_i$ будет общее число процессоров, принадлежащих машинам от N_f, до N_1. Предположим, без потери общности, что параллельные машины располагаются в неубывающем порядке их размеров $m_1 \leq \cdots \leq m_m$.

Каждая работа J_j описывается кортежом $(r_j, size_j, p_j, p'_j)$, определяющим следующие характеристики работы: время ее появления $r_j \geq 0$ – время относительно начала расписания, ее размер $1 \leq size_j \leq m_m$ – требования к процессору, время выполнения p_j и оценку пользовательского времени выполнения p'_j.

В онлайн сценарии, никакие параметры работ не доступны до момента их подачи. Планирование затруднено, если время обработки работы становится известным только по окончании ее выполнения. Это называется неклаирвоянтным сценарием. В некоторых реальных системах пользователь должен предоставить оценку времени обработки своей работы, чтобы предотвратить растрату вычислительных ресурсов при работе с программами, в которых есть ошибки, например, неограниченные циклы. Эта оценка также может быть использована планировщиком, хотя оценки могут быть довольно неточными. В этой главе рассмотрены оба сценария.

Пусть $w_j = p_j \cdot size_j$ это обьем работы J_j, также называемая ее областью в расписании или ее потребляемым ресурсом. Работы подаются с течением времени и должны быть немедленно и окончательно распределены на одну машину. Однако, выделение процессоров для работы может быть отложено до тех пор,

пока требуемое число процессоров не будет реально доступно. Работа выполняется в режиме разделения пространства путем исключительно точного выделения $size_j$ процессоров на непрерываемый период времени p_j. Т.к. не допускается ни прерывания, ни мульти-машинного исполнения, ни совместного распределения процессоров с разных машин, работа J_jможет выполняться на машине N_i только в том случае, если $size_j \leq m_i$. Напоминаем, что точное время обработки и точный объем работы не доступны планировщику в наших сценариях.

Применим $g_j = i$ для обозначения того, что работа J_j назначена на машину N_i, n_i для обозначения чила работ, назначенных на машину N_i, и $W_i = \sum_{g_k=i} w_k$ для обозначения общего объема работ, назначенных на машину N_i.

Время завершения работы J_j экземпляра I в расписании S обозначается $c_j = (S, I)$. Формально, продолжительность выполнения работ в расписании S и экземпляре I равна $C_{max}(S, I)$. Оптимальное время окончания работы экземпляра I обозначается $C^*_{max}(I)$. В Дальнейшем, везде, где это возможно, не вызывая двусмысленности, опустим экземпляр I и расписание S.

Обозначим модель машины ГРИД как GP_m. В нотации трех полей α | β | γ, наша задача планирования характеризуется как $GP_m|r_j, size_j|C_{max}$ для критерия оптимизации C_{max}. Целевые функции основаны на метриках, рассмотренных в разделе 3.2. Как и в [17], для обозначения этой проблемы используется нотация MPS (Multiple machine Parallel Scheduling - параллельное машинное планирование), а нотация PS (Parallel Scheduling - параллельное планирование) описывает параллельное планирование работ на одной параллельной машине $GP_m|r_j, size_j|C_{max}$.

Локальная система управления ресурсами (англ., Local Resource Management System – LRMS) может использовать различные алгоритмы планирования. В экспериментальных частях данной главы предполагается, что LRMS использует алгоритм параллельного планирования в режиме онлайн: политика First-Come-First-Serve с алгоритмом EASY backfilling [66], где

планировщик может использовать более поздние задания для заполнения дыр в планировании, даже если это задерживает ожидаемое время начала других заданий, до тех пор, пока ожидаемое время начала первого задания не задерживается. Чтобы применить EASY backfilling, используется пользовательское расчетное время выполнения.

Формально, конкурентный фактор алгоритма A определяется как $\rho_A = \max_I \frac{C_{max}(S_A,I)}{C^*_{max}(I)}$ для всех возможных экземпляров проблемы. Опять же, опустим алгоритм A, если это не вызывает двусмысленности. Аппроксимационный фактор определяется аналогично для детерминистических (оффлайн) задач планирования.

Следует обратить внимание, что в нашем детерминистическом планировании все задачи доступны в начале и планировщик знает параметры всех задач. Предполагается, что задействованные ресурсы стабильны и предназначены для работы в ГРИД.

Также рассматриваются хорошо известные показатели производительности алгоритмов (метрики), обычно используемые для обозначения целей различных участников планирования ГРИД (конечных пользователей, местных поставщиков ресурсов и администраторов ГРИД), такие как: среднее замедление (англ., mean slowdown): $SD_b = \frac{1}{n}\sum_{j=1}^{n} \frac{c_j - r_j}{\max\{10, p_j\}}$ и среднее время ожидания (англ., mean waiting time) $t_w = \frac{1}{n}\sum_{j=1}^{n}(c_j - p_j - r_j)$.

Чтобы исключить влияние очень коротких работ (т.е. с близким к нулю временем выполнения), замедление ограничено часто используемым порогом в 10 с.

В данной работе используется время ожидания t_w вместо времени получения (ответа) $TA = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j)$, и суммы времен ожидания работ) $SWT = \sum_{j=1}^{n}(c_j - p_j - r_j)$.

Различия между этими метриками являются константами независимо от используемого планировщика: $\frac{1}{n}\sum_{j=1}^{n} p_j$.

Не используются ресурсо-ориентированные метрики, такие как утилизация $U = \sum_{j=1}^{n} \frac{p_j \cdot size_j}{C_{max} \cdot s_1, m}$ и пропускная способность $Th = \frac{n}{C_{max}}$.

Между ними существует тесная взаимосвязь: поскольку $C^*_{max}, \frac{p_j \cdot size_j}{s_1, m}$, и n являются константами для данного эксперимента; x - процентное уменьшение C_{max} соответствует $\frac{x}{100\% - x}$ процентному увеличению утилизации и пропускной способности.

Заметим, что в экспериментальном анализе нижняя граница оптимального времени завершения $\hat{C}^*_{max} = \max\left\{\max_j(r_j + p_j), \frac{\sum_{j=1}^{n} w_j}{s_1, m}\right\}$ используется вместо оптимального времени C^*_{max} для расчета конкурентного фактора.

3.3 Иерархическая система планирования

Планирование работ является важнейшей частью эффективного функционирования вычислительных систем. Различные аспекты проблемы обсуждаются в литературе для решения новых задач, стоящих перед распределенными системами: централизованные, иерархические и распределенные модели [17, 130, 222]; статические [83, 134] и динамические политики планирования [147]; многокритериальная оптимизация [146]; адаптивные политики, связанные с динамическим поведением ресурсов [67, 151]; автономное управление [186]; ограничения QoS [117]; экономические модели [86]; выбор ресурсов [192]; планирование данных и ресурсоемких вычислений [63, 152]; планирование потоков работ [219, 208]; локализация данных, привязки ресурсов для исполнения и хранения данных [98]; репликация [96]; оценка производительности [157].

Теоретические оценка планирования в распределенных системах изучаются для того, чтобы дать подсказки для реализации реальных систем, в которых могут быть доступны только пользовательские оценки времени выполнения работ.

Следовательно, важно, чтобы планировщик мог интегрировать пользовательские оценки времени и доступную информацию с динамическими и статическими стратегиями распределения ресурсов. Это является одной из целей данной главы.

Распределенные системы существенно отличаются друг от друга по размеру, их рабочая нагрузка очень динамична. Важной характеристикой производительности для алгоритмов планирования является качество обслуживания. Поэтому теоретический анализ наихудшего случая является актуальным подходом, т. к. он обеспечивает такие гарантии в системах с не стационарными параметрами.

Относительно иерархической задачи планирования, задача с работами, которые могут потребовать больше ресурсов, чем имеется на самой маленькой машине рассмотрена в [58]. Представлены алгоритмы MLBa+LSF и MCTa+LSF с аппроксимационным фактором 10. На первом уровне планировщиком распределенной системы для распределения работ по машинам используются стратегии "Минимальный нижний предел расписания" и "Минимальное время выполнения работы". На втором уровне применяется алгоритм локального планирования (работы большего размера обрабатываются первыми).

В данной главе эти результаты улучшены, путем введения коэффициента допустимости, который ограничивает доступность машин ГРИД для распределения работ. Конкурентный фактор адаптивного онлайн алгоритма планирования MLBa + PS, варьирующимся от 5 до бесконечности путем изменения коэффициента допустимости, был получен для специфических характеристик нагрузки.

Результат для более общей модели рабочей нагрузки получен в [17]. Конкурентный фактор алгоритма MLBa+PS колеблется от 17 до бесконечности при изменении коэффициента допустимости. В [21] аппроксимационный фактор улучшен до 9 для оффлайн и конкурентный фактор до 11 для онлайн сценариев. Также обсуждаются различные вариации задачи и доказываются их аппроксимационный и конкурентный факторы.

3.4 Алгоритмы

Рассмотрим алгоритмы, которые не знают о работах ничего, кроме числа незавершенных работ в системе, и их требований к числу процессоров.

3.4.1 Двухуровневое планирование

Алгоритмы планирования для двухуровневых моделей ГРИД можно разделить на глобальную часть распределения работ и локальную часть их планирования. Таким образом, *MPS* рассматривается как двухэтапная стратегия планирования: *MPS = MPS_Alloc+PS*. На первом этапе каждая работа назначается на подходящую машину, используя заданный критерий выбора. На втором этапе алгоритм PS применяется к каждой машине для работ, назначенных на предыдущем этапе.

Легко заметить, что конкурентный фактор алгоритма *MPS* ограничен снизу конкурентным фактором алгоритма *PS*, учитывая вырожденную ГРИД, которая содержит только одну машину. Лучший из возможных онлайн неклаирвоянтных алгоритмов (англ., non-clairvoyant, с неизвестным временем выполнения работ) *PS* имеет конкурентный фактор $2 - 1/m$, где m обозначает число процессоров в одной параллельной машине; см. результаты работы [177] и [11, 5]. Следовательно, нижняя граница конкурентного фактора для любого общего двухслойного онлайн *MPS* планирования составляет не менее $2 - 1/m$.

3.4.1.1 Стратегии распределения работ

Выделяется три уровня информации, доступной для распределения работ. Каждый уровень отличается по типу и объему информации, необходимой для создания расписания (см таблицу 3.1).

Уровень 1: после того, как работа поступила в систему, становятся известны ее требования к процессору. Информация о требуемом времени обработки работ отсутствует, т.е. на этом уровне рассматриваются неклаирвоянтное планирование. Алгоритм может использовать информацию о ранее выполненных работах. Алгоритмы Random, MLp, MPL, LBal_S распределяют работу на машину случайным образом, на машину с наименьшей загрузкой процессора ($\min_{i=1,\dots,m}\left(\frac{n_i}{m_i}\right)$), на машину с наименьшими требованиями к процессору ($\min_{i=1,\dots,m}\left\{\sum_{g_k=i}\left(\frac{size_k}{m_i}\right)\right\}$), а также на машину с наименьшим стандартным отклонением требований к процессору ($\min_{i=1,\dots,m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(PL_i^q-\overline{PL}\right)^2}$), где $PL_{i=1,..,m}^q=\frac{1}{m_i}\sum_{g_{k=1}}\left(size_k+size_j^q\right)$ и $size_j^q$ - размер работы J_j, добавленной на машину q.

Уровень 2: Алгоритм имеет доступ ко всей информации уровня 1 и к оценке времени выполнения работы p'_j. MLB распределяет работу на машину с наименьшим объемом работ на процессор $\min_{i=1,\dots,m}\left\{\sum_{g_k=i}\frac{size_k\cdot p'_k}{m_i}\right\}$ на момент распределения работы J_j.

Уровень 3: Алгоритм имеет доступ ко всей информации уровня 2, а также ко всем локальным расписаниям. MCT, MWT, MWWT_S, MST назначают работу J_j на сайт с самым ранним временем завершения $\min_{i=1,\dots,m}\{C_{max}^i\}$ до назначения работы J_j, на машину с минимальным средним временем ожидания работы $\min_{i=1,\dots,m}\left\{\sum_{g_k=i}\frac{t_w^k}{n_i}\right\}$, на машину с минимальным взвешенным средним временем ожидания $\min_{i=1,\dots,m}\left\{\sum_{g_k=i}\left(t_w^k\cdot\frac{weight_k}{n_i}\right)\right\}$, и на машину с самым ранним временем начала работы $\min_{i=1,\dots,m}\{s_j^i\}$ соответственно. Более подробную информацию о стратегиях распределения можно найти в [9].

Таблица 3.1 – Стратегии планирования

Стратегия	Уровень	Описание
Random	1	Распределяет работу на машину случайным образом
ML_p	1	Распределяет работу j на машину с наименьшей загрузкой процессора в момент r_j: $\min_{i=1..m}\left(\frac{n_i}{m_i}\right)$. Мотивация ML_p заключается в том, чтобы сбалансировать нагрузку между процессорами машин.
MPL	1	Распределяет работу j на машину с наименьшими требованиями к процессору в момент r_j: $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{size_k}{m_i}\right\}$. Интуиция, лежащая в основе MPL заключается в том, чтобы все процессоры были максимально загружены. Одним из преимуществ MPL является его простота. В нем не учитывается ни время выполнения задания, ни оценка времени выполнения.
LB_{al}_S	1	Распределяет работу j на машину с наименьшим стандартным отклонением требований к процессору (учитывая все машины) когда работа назначена на него $\min_{q=1..m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(PL_i^q-\overline{PL}\right)^2}$, где $PL_{i=1..m}^q=\frac{1}{m_i}\sum_{g_k=i}\left(size_k+size_j^q\right)$ и $size_j^q$ - размер работы J_j, добавленной на машину q
MLB	2	Распределяет работу j на машину с наименьшей загрузкой на процессор в момент r_j r_j: $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{size_k\cdot p\prime_k}{m_i}\right\}$
MCT	3	Распределяет работу j на машину с самым ранним временем завершения $\{C_{max}^i\}$, где $C_{max}^i=\max_{g_k=i}(C_k^i)$, и C_k^i время окончания работы J_k на машине i. Это приводит к тому, что некоторые работы назначаются на машины, которые не имеют для них минимального времени завершения. MCT пытается минимизировать общее время завершения работы
MWT	3	Распределяет работу j на машину с минимальным средним временем ожидания $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k}{n_i}\right\}$
$MWWT\_S$	3	Распределяет работу j на машину с с минимальным взвешенным средним временем ожидания $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{t_w^k\cdot weight_k}{n_i}\right\}$, где $weight_k=\{size_k,p'_k,w_k\}$
MST	3	Распределяет работу j на машину с самым ранним временем начала работы для этой работы $\min_{i=1..m}\{s_j^i\}$. Для однородных платформ MST назначает каждую работу на машину с минимальным ожидаемым временем завершения этой работы (Earliest-Finish-Time).

3.4.1.2 Допустимое распределение работ

Анализ проблемы назначения работ на распределенные машины в режиме онлайн до сих пор решался редко. К сожалению, в худшем случае это может привести к неэффективному использованию всей системы. Одной из структурных причин такой неэффективности при онлайн расписаниях является потенциальное использование больших машин работами с небольшими требованиями к числу процессоров, что заставляет высоко параллельные работы ждать окончания их выполнения если они посланы позже.

Допустимым набором машин для работы J_j являются машины с индексами $\{f_j, \ldots, l_j\}$ (или же просто $\{f, \ldots, l\}$, если это не вызывает двусмысленности), где f_j - наименьший индекс i такой, что $m_i \geq size_j$, а l_j наименьший машинный индекс такой, что $s_{f_j,l_j} \geq a \cdot s_{f_j,m}$ (см. рисунок 3.1), где s_{f_j,l_j} общее число процессоров, принадлежащих машинам от N_{f_j} до N_{l_j}. Следует обратить внимание, что всегда $l_j \leq m$. Коэффициент допустимостми (admissible) $0 < a \leq 1$ параметризует допустимость машин, используемых для распределения работ. Заметим, что, по крайней мере, одна машина допустима, т. к. a строго больше 0, в то время как $a = 1$ определяет, что все доступные машины допустимы. Если f - наименьший индекс, например, $m_f \geq size_j$, то работу можно распределить по машинам от индекса f до m. Коэффициент допустимостми уменьшает доступные машины до машин с индексами от f до l (см. рисунок 3.1).

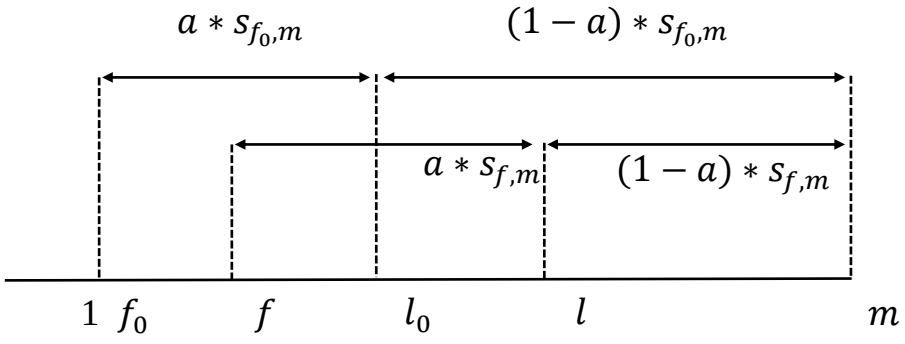


Рисунок 3.1 – Пример допустимых машин для назначения работ с фактором a

Предположим, что работа J_j выделена машине из множества $f,\ldots,l$ содержащего процессоры $a \cdot s_{f,m}$ (см. рисунок 3.1). Следовательно, $(1-a) \cdot s_{f,m}$ процессоров исключаются из $s_{f,m}$ процессоров, доступных для J_j. Отметим, что машины индексируются в порядке не убывания их размеров. Очевидно, что общий набор машин представлен целым набором $1,\ldots,m$.

3.5 Анализ алгоритмов

В этом разделе проводится анализ двух алгоритмов *MLBa + PS* и *MCTa + PS*, в которых не все машины допустимы для некоторой работы, хотя и способны ее выполнять. Показывается, что производительность стратегий распределения зависит от допустимого фактора, и может быть динамически адаптирована к различным рабочим нагрузкам и изменениям в конфигурации.

Теорема 3.1. *Оффлайн расписание параллельных работ на ГРИД с идентичными процессорами по алгоритму MCTa + PS с коэффициентом допустимости* $0 < a \leq 1$ *обладает аппроксимационным фактором*

$$\rho \leq \begin{cases} 1+\dfrac{2}{a^2}, & if\ a \leq \dfrac{s_{f,l}}{s_{f_0,m}} \\ 1+\dfrac{2}{a(1-a)}, & if\ a > \dfrac{s_{f,l}}{s_{f_0,m}} \end{cases},$$

где $1 \leq f_0 \leq f \leq l \leq m$ *являются параметрами, которые зависят от конфигурации машины и рабочей нагрузки.*

Доказательство. Предположим, что время окончания работ машины k также является C_{max} ГРИД, так что удовлетворяется $C_{max} = C_k$. Далее, пусть работа J_d будет последней работой, которая была добавлена на эту машину.

Т. к. J_d была добавлена на машину k, MCT_a гарантирует $C_k \leq C_i$ перед добавлением J_d. Следовательно, исходя из 2-конкурентости алгоритма PS, имеем

$C_k \le \min C_i = \min\left\{2 \cdot \max\left\{p_{max}, \frac{W_i}{m_i}\right\}\right\}$, и $C_k \le max\left\{min\{2p_{max}\}, min\left\{2\frac{W_i}{m_i}\right\}\right\}$.

Пусть машина e будет машиной с наименьшим соотношением $\frac{W_i}{m_i} : \frac{W_e}{m_e} = \min_{f \le i}\left\{\frac{W_i}{m_i}\right\}$. Следовательно, $C_k \le max\left\{2p_{max}, 2\frac{W_e}{m_e}\right\}$. Добавляя J_d получаем $C_{max} \le C_k + p_d \le C_k + p_{max}$ и

$$C_{max} \le max\left(2\frac{W_i}{m_i} + p_{max}, 3p_{max}\right) \quad (3.1)$$

Теперь рассмотрим свойства оптимального расписания.

$$W_{f,l} = \sum_{i=f}^{l} W_i = \sum_{i=f}^{l} \frac{W_i}{m_i} \cdot m_i \ge \sum_{i=f}^{l} \frac{W_e}{m_e} \cdot m_i = \frac{W_e}{m_e} \sum_{i=f}^{l} m_i = \frac{W_e}{m_e} \cdot s_{f,l} \quad (3.2)$$

Пусть J_b это работа, имеющая наименьший размер среди всех работ, которые могут быть выполнены на машине N_f при нашем планировании, т.е. $l_b \ge f$. Следовательно, работы, упакованные на машине $N_f, \dots, N_l$ не могут быть отнесены к машине с меньшим индексом, чем f_b. Т. к. J_b может выполняться на машине N_f, у нас есть $C^*_{max} \ge \frac{W_{f,l}}{s_{f_0,m}}$. Подставляя (3.2) в эту формулу, получаем $C^*_{max} \ge \frac{W_e}{m_e} \cdot \frac{s_{f,l}}{s_{f_0,m}}$.

Учитывая, что $s_{f,l} \ge a \cdot s_{f,m}$ получаем $s_{f,l} \ge a^2 \cdot s_{f_0,m}$ если выполняется $a \le \frac{s_{f,m}}{s_{f_0,m}}$.

Это дает $C^*_{max} \ge \frac{W_e}{m_e} \cdot \frac{s_{f,l}}{s_{f_0,m}} \ge \frac{W_e}{m_e} \cdot a^2$.

Благодаря $C^*_{max} \ge p_{max}$, формула (3.1) преобразуется в $\rho \le \max\left\{\frac{2 \cdot W_e}{m_e \cdot C^*_{max}} + \frac{p_{max}}{C^*_{max}}, \frac{3p_{max}}{C^*_{max}}\right\} \le max\left\{1 + \frac{2}{a^2}, 3\right\}$ и $\rho \le 1 + \frac{2}{a^2}$, поскольку $a < 1$.

Учитывая условие $s_{f_0,m} \le s_{f,m} + a \cdot s_{f_0,m}$ (см. рисунок 3.1) и если $a > \frac{s_{f,m}}{s_{f_0,m}}$, то $C^*_{max} \ge \frac{W_e}{m_e} \cdot \frac{s_{f,l}}{s_{f_0,m}} \ge \frac{W_e}{m_e} \cdot \frac{a \cdot s_{f,m}}{\frac{s_{f,m}}{1-a}} = \frac{W_e}{m_e} \cdot a \cdot (1-a)$ и $\rho \le max\left\{\frac{2 \cdot W_e}{m_e \cdot C^*_{max}} + \frac{p_{max}}{C^*_{max}}, \frac{3p_{max}}{C^*_{max}}\right\} \le max\left\{1 + \frac{2}{a \cdot (1-a)}, 3\right\}$. Следовательно, $\rho \le 1 + \frac{2}{a \cdot (1-a)}$, поскольку $a < 1$.

Теорема 3.2. Оффлайн расписание параллельных работ в ГРИД с

идентичными процессорами по алгоритму $MLBa{+}PS$ с коэффициентом допустимости $0 < a \leq 1$ имеет аппроксимационный фактор

$$\rho \leq \begin{cases} 1 + \dfrac{2}{a^2}, & if\ a \leq \dfrac{s_{f,l}}{s_{f_0,m}} \\ 1 + \dfrac{2}{a(1-a)}, & if\ a > \dfrac{s_{f,l}}{s_{f_0,m}} \end{cases}$$

с параметрами $1 \leq f_0 \leq f \leq l \leq m$, которые зависят от конфигурации машины и нагрузки.

Доказательство. Предположим, что время окончания работ машины k также является C_{max} ГРИД, так что удовлетворяется $C_{max} = C_k$.

Далее, пусть работа J_d будет последней работой, которая была добавлена на эту машину. Машины $f = f_d, \dots, l = l_d$ составляют набор $M_{admissible}(d)$. Т. к. J_d была добавлена на машину k, MLB_a гарантирует $\frac{W_k}{m_k} \leq \frac{W_i}{m_i}$ для всех $i = f, \dots, l$. Следует обратить внимание, что W_k не включает в себя работу J_d.

Таким образом, основываясь на свойстве 2-конкурентности алгоритма PS, получаем

$$C_k \leq max\left\{2 \cdot \frac{W_k}{m_k}, 2 \cdot p_{max}\right\} \tag{3.3}$$

$$W_{f,l} = \sum_{i=f}^{l} W_i = \sum_{i=f}^{l} \frac{W_i}{m_i} \cdot m_i \geq \sum_{i=f}^{l} \frac{W_k}{m_k} \cdot m_i = \frac{W_k}{m_k} \sum_{i=f}^{l} m_i = \frac{W_k}{m_k} \cdot s_{f,l} \tag{3.4}$$

Пусть J_b будет работой с наименьшим размером среди всех работ, выполняемых на машинах $N_f, \dots, N_l$, используя обозначение $f_0 = f_b$. Следовательно, работы, упакованные в $N_f, \dots, N_l$, не могут быть назначены на машину с меньшим индексом, чем f_0. Т. к. J_b выполняется на одной из машин $N_f, \dots, N_l$, то получаем $l_b \geq f$ и $C^*_{max} \geq \frac{W_{f,l}}{s_{f_0,m}}$.

Заменяя (3.4), получаем $C^*_{max} \geq \frac{W_k}{m_k} \cdot \frac{s_{f,l}}{s_{f_0,m}}$

Т. к. схема MLB_a использует W_k без включения работы J_b, необходимо дополнительно учитывать работу J_b. В худшем случае C_k увеличивается на $p_b \leq C^*_{max}$, в результате чего $c_{max} \leq C_k + C^*_{max}$ и $\rho \leq 1 + \frac{C_k}{C^*_{max}}$.

Используя (3.3), получаем $\rho \le 1 + \frac{max\left\{2\frac{W_k}{m_k}\right\},2p_{max}}{C^*_{max}}$.

Если $s_{f,l} \ge a \cdot s_{f,m}$ получаем $s_{f,l} \ge a^2 \cdot s_{f_0,m}$ если $a \le \frac{s_{f,m}}{s_{f_0,m}}$. Это дает $C^*_{max} \ge \frac{W_k}{m_k} \cdot \frac{s_{f,l}}{s_{f_0,m}} \ge \frac{W_k}{m_k} \cdot a^2$ и $\rho \le 1 + \frac{max\left\{2\frac{W_k}{m_k}\right\},2p_{max}}{C^*_{max}}$. С учетом $\frac{2}{a^2} \ge 2$, получается $\rho \le 1 + \frac{2}{a^2}$.

Если $s_{f_0,m} \le s_{f,m} + a \cdot s_{f_0,m}$ (see Fig. 3.1), и если $a > \frac{s_{f,m}}{s_{f_0,m}}$, получаем $C^*_{max} \ge \frac{W_k}{m_k} \cdot \frac{s_{f,l}}{s_{f_0,m}} \ge \frac{W_k}{m_k} \cdot \frac{a \cdot s_{f,m}}{\frac{s_{f,m}}{1-a}} = \frac{W_k}{m_k} \cdot a \cdot (1-a)$ и $\rho \le 1 + \frac{\max\left\{2\frac{W_k}{m_k}\right\},2p_{max}}{C^*_{max}} \le 1 + \frac{2}{a \cdot (1-a)}$.

Заметим, что обе границы аппроксимации MLBa+PS и MCTa+PS дают один и тот же результат $\rho \le 9$ при $a = 0.5$.

Теорема 3.3. *Онлайн расписание параллельных работ на ГРИД с идентичными процессорами с использованием алгоритмов MCTa + PS и MLBa + PS с коэффициентом допустимости* $0 < a \le 1$ *имеет конкурентный фактор*

$$\rho \le \begin{cases} 3 + \frac{2}{a^2}, & if\ a \le \frac{s_{f,l}}{s_{f_0,m}} \\ 3 + \frac{2}{a(1-a)}, & if\ a > \frac{s_{f,l}}{s_{f_0,m}} \end{cases}$$

с параметрами $1 \le f_0 \le f \le l \le m$, *которые зависят от конфигурации машины и нагрузки.*

Доказательство. Предположим, что r это время поступления последней работы в расписании. После момента r можно применить независимый алгоритм. Однако он должен начинаться с того момента, когда все процессоры будут простаивать. Время необходимое до его достижения ограничено максимальным временем обработки любого задания. Поэтому расписание заканчивается не позднее времени $r + p_{max} + C_{max}$. Теорема действительна т. к. выполняются неравенства $C^*_{max} \ge r$ и $C^*_{max} \ge p_{max}$, см. теоремы 1 и 2.

Примечание. На рисунках 3.2 и 3.3 показаны границы конкурентного фактора стратегий *MCTa + PS* и *MLBa + PS*, если коэффициент допустимости

$a \leq \frac{s_{f,l}}{s_{f_0,m}}$ и $a > \frac{s_{f,l}}{s_{f_0,m}}$, соответственно. Видно, что если $a \leq \frac{s_{f,l}}{s_{f_0,m}}$, то границы наихудшего случая меняются с ∞ на 5 (рисунок 3.2) в зависимости от a. Если $a > \frac{s_{f,l}}{s_{f_0,m}}$ то границы наихудшего случая меняются с ∞ (если а близко к 0) на ∞ (если а близко к 1) (см. рисунок 3.3) с минимальным значением 11, если $a = 0.5$.

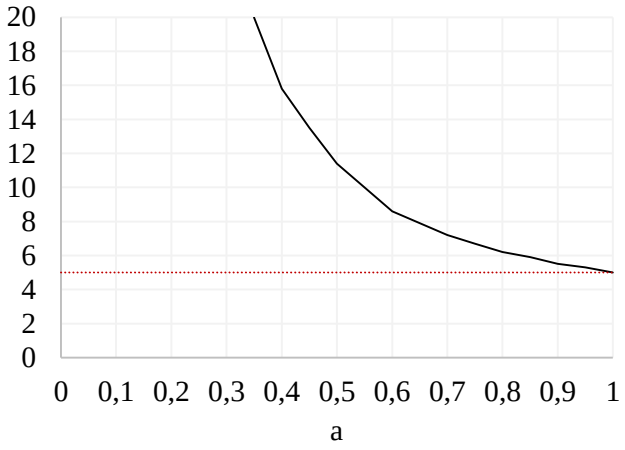


Рисунок 3.2 – Конкурентный фактор ρ стратегий *MCTa + PS* и *MLBa + PS* если $a \leq \frac{s_{f,l}}{s_{f_0,m}}$

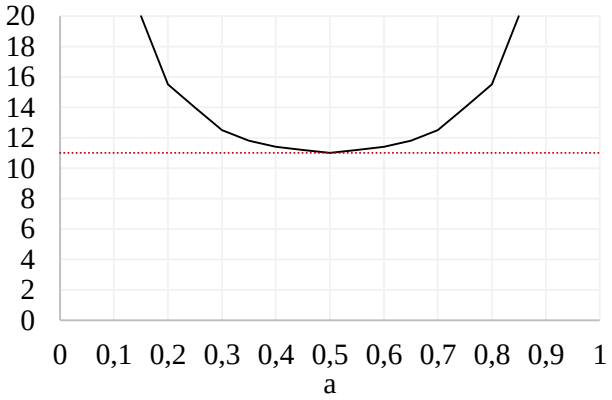


Рисунок 3.3 – Конкурентный фактор ρ стратегий *MCTa + PS* и *MLBa + PS* если $a > \frac{s_{f,l}}{s_{f_0,m}}$

На рисунке 3.4 показана результирующая граница, являющаяся максимумом для наихудших конкурентных факторов, показанных на рис. 3.2 и 3.3. Граница

дает $\rho \leq 11$ при $a = 0.5$.

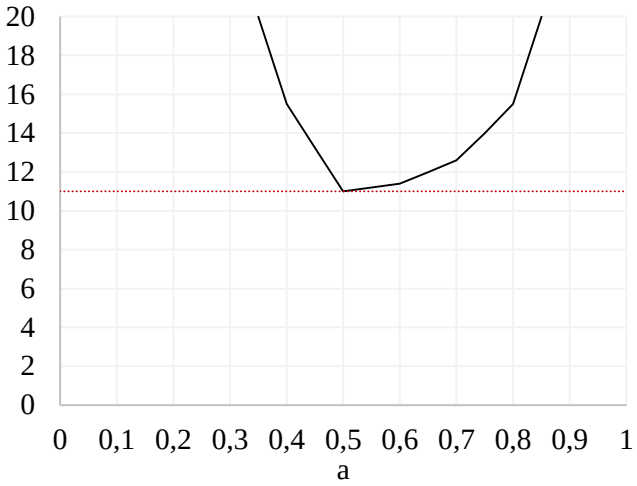


Рисунок 3.4 – Конкурентный фактор ρ стратегий *MCTa + PS* and *MLBa + PS*

Теорема 3.4. *Планирование параллельных работ на распределенных системах с идентичными процессорами с использованием алгоритмов MCTa +PS, MLBa +PS и работ с размерами, которые могут быть выполнены на самой маленькой машине, имеет аппроксимационный фактор 3 и конкурентный фактор 5.*

Доказательство. Доказательство следует сразу же после рассмотрения ограничений, связанных с размерами работ. Если все работы имеют размеры не более, чем размер самой маленькой машины, $size_j \leq m_1$, то все работы могут быть распределены по всем машинам от 1 до m. Следовательно, получаем $f_0 = f = 1$. Учитывая $l = m$, получаем $s_{f,l} = s_{f_0,m}$, $a = 1$. Учитывая теоремы 3.1, 3.2, 3.3 получаем $\rho \leq 3$ для случая оффлайн и $\rho \leq 5$ для случая онлайн.

3.6 Параметры экспериментального анализа

Для оценки производительности необходимо решить два фундаментальных вопроса. С одной стороны, для получения надежных результатов необходимы репрезентативные данные о рабочие нагрузки. С другой стороны, необходимо создать среду тестирования для получения воспроизводимых и сравнимых результатов. В этом разделе опишем параметры моделирования на основе логов Грид с акцентом на упомянутые два вопроса.

3.6.1 Рабочая нагрузка

Точность оценки в значительной степени зависит от применяемых рабочих нагрузок. Для тестирования производительности выполнения работ в среде Грид используем рабочие нагрузки, основанные на реальных производственных трассах. Тщательно реконструированные трассы реальных суперкомпьютеров обеспечивают реалистичный поток работ для оценки производительности алгоритмов планирования работ на основе моделирования.

В данной главе используются трассы из архивов PWA и GWA.

Объединение этих машин в Грид потребует слияния пользователей и их работ. Это не гарантирует представления реальной Грид. Тем не менее, в отсутствие общих трасс рабочих нагрузок Грид, это разумная отправная точка для оценки стратегий планирования на основе реальных трасс. Были рассмотрены нормализация временных зон, нормализация профилированных временных интервалов и фильтрация недействительных работ. Обратите внимание, что они также предоставляют пользовательские оценки времени выполнения для всех заданий. Фоновая нагрузка (локально сгенерированные задания) является важным вопросом в неспециализированных Грид-средах, но в данной работе не рассматривается.

3.6.2 Конфигурация Грид

Одной из целей данной работы является оценка алгоритмов в больших инфраструктурах HPC. Рассматриваются три сценария.

В Grid1 используются 7 существующих вычислительных центров (KTH, SDSC-SP2, HPC2N, CTC, LANL, SDSC-BLUE, SDSC-DS) с общим числом процессоров 4442 (таблица 3.2). Используются соответствующие трассы (логи) реальных рабочих нагрузок. Машины, которым принадлежат эти трассы, имеют 100, 128, 240, 430, 1024, 1152 и 1368 процессоров соответственно.

Таблица 3.2. – Характеристики Grid1

	Система	Procs	Log
1	KTH—Swedish Royal Institute of Technology	100	KTH-SP2–1996–2.swf, 28489
2	SDSC-SP2—San Diego Supercenter SP2	128	работ, 204 пользователей
3	HPC2N—High Performance Computing Center North, Sweden	240	SDSC-SP2–1998–3.1-cln.swf, 73496 работ, 437 пользователей
4	CTC—Cornell Theory Center	430	HPC2N-2002–1.1-cln.swf, 527371
5	LANL—Los Alamos National Lab	1024	работ, 256 пользователей
6	SDSC-BLUE—San Diego Supercenter Blue Gene	1152	CTC-SP2–1996–2.1-cln.swf, 79302 работ, 679 пользователей
7	SDSC-DS—San Diego Supercenter Data Star	1368	LANL-CM5–1994–3.1-cln.swf, 201387 работ, 211 пользователей
			SDSC-BLUE-2000–3.1-cln.swf, 250,440 работ, 468 пользователей
			SDSC-DS-2004–1-cln.swf, 96089 работ, 460 пользователей

Таблица 3.3 – Характеристики Grid2

	Система	Procs	Log
1	DAS2—University of Amsterdam	64	Gwa-t-1-anon_jobs-reduced.swf, 1124772
2	DAS2—Delft University of Technology	64	работ, 333 пользователей
3	DAS2—Utrecht University	64	
4	DAS2—Leiden University	64	
5	KTH—Swedish Royal Institute of	100	KTH-SP2–1996–2.swf, 28489 работ, 204
6	Technology	144	пользователей
7	DAS2—Vrije University Amsterdam	240	Gwa-t-1-anon_jobs-reduced.swf (cont.)
	HPC2N—High Performance Computing		HPC2N-2002–1.1-cln.swf, 527371 работ,
8	Center North, Sweden	430	256 пользователей
9	CTC—Cornell Theory Center	1024	
	LANL—Los Alamos National Lab		CTC-SP2–1996–2.1-cln.swf, 79302 работ, 679 пользователей

В Grid2 рассматриваются 9 центров, из них 5 центров Grid DAS, а также KTH, HP2CN, CTC и LANL с общим числом процессоров 2194 (таблица 3.3). Размеры машин составляют 64, 64, 64, 64, 100, 144, 240, 430 и 1024 процессора. Используются соответствующие трассы.

В сценарии Grid3 рассматривается гораздо меньший Grid из 11 сайтов с общим числом процессоров 136. Машины имеют следующие размеры: 4, 4, 4, 4, 4, 8, 8, 8, 16, 16, 32 и 32 процессора. Такой сценарий используется для имитации высокой нагрузки работ с высоким параллелизмом и низким параллелизмом. Он создает тестовые случаи, рассмотренные в теоремах 1-3, где не все работы можно разместить на наименьшей машине.

В Grid3 используем нагрузку Grid2 с заданиями, требующими не более 32 процессоров. В этой рабочей нагрузке 37% работ требуют 1 процессор; 23%, 10%, 6% и 5% работ требуют, соответственно, 2, 4, 8 и 32 процессора.

В сценариях Grid1 и Grid2 большинство работ может быть распределено на наименьшую машину, как это рассматривается в Теореме 4. Например, в сценарии Grid2 большинство работ требуют не более 32 процессоров (97%) и могут быть распределены на самую маленькую машину размером 64.

3.7 Результаты моделирования

В этом разделе представлены и проанализированы результаты моделирования стратегий распределения заданий с использованием показателей производительности, описанных в разделе 3.2.1. Детали конфигураций Грид и трасс, используемых в экспериментах, описаны в разделе 3.6.

Экспериментальная оценка двухэтапных стратегий планирования MPS = MPS_Alloc + PS проводится в два этапа. На первом этапе сравниваются девять стратегий распределения, представленных в разделе 3.4.1.1. Пользовательские оценки времени выполнения работ используются как в MPS_Alloc, так и в PS. Во

второй части подробно исследуется влияние допустимого коэффициента, включенного в политику распределения заданий, на общую производительность Грид. Показывается, что представленные алгоритмы выгодны при определенных условиях и допускают эффективную реализацию в реальных системах.

Для получения достоверных статистических значений были смоделированы эксперименты с интервалом в 6 месяцев для Grid1 и Grid2, и 30 экспериментов по 7 дней для Grid3.

3.7.1 Метод оценки качества алгоритмов

Поскольку в общей формулировке проблема является многокритериальной, рассматривается несколько критериев эффективности. Часто поставщики и пользователи ресурсов имеют различные, иногда противоречивые цели: от минимизации времени отклика до оптимизации использования ресурсов. Управление ресурсами Грид может использовать многокритериальную поддержку принятия решений. Для этой цели может быть применена общая методология многокритериальных решений, основанная на оптимальности Парето.

Однако очень трудно добиться быстрого решения с помощью доминирования по Парето, как это необходимо для управления ресурсами. Проблема очень часто упрощается до одноцелевой задачи или до различных методов объединения целей. Существуют различные способы моделирования предпочтений, например, они могут быть заданы в явном виде заинтересованными сторонами для указания важности (веса) каждого критерия или относительной важности между критериями. Это может быть сделано путем определения весов критериев или ранжирования критериев по их важности.

Чтобы обеспечить эффективное сопровождение в выборе лучшей стратегии, проведен комбинированный анализ нескольких метрик в соответствии с методологией, предложенной в [211] и примененной для проблемы планирования в [9, 56, 21]. Цель - найти надежную и хорошо работающую стратегию во всех

тестовых случаях, ожидая, что она будет хорошо работать и в других условиях, например, при различных конфигурациях и рабочих нагрузках.

Анализ проводится следующим образом. Во-первых, оценивается ухудшение (деградация, относительная ошибка) в производительности каждой стратегии по каждой метрике. Это делается по отношению к наилучшим показателям по данной метрике:

$$100 \cdot \left(\frac{metric}{best\_metric} - 1\right).$$

Таким образом, для трех основных метрик каждая стратегия характеризуется 3-мя числами (9 для 3х Грид), отражающими ее относительное ухудшение производительности в тестовых случаях. Затем эти 3 значения усредняются (предполагая равную важность каждой метрики) и ранжируются стратегии для каждой ГРИД. Лучшая стратегия, с наименьшим средним снижением производительности, имеет ранг 1; худшая стратегия имеет ранг 9.

Затем вычисляется среднее снижение производительности для Grid1, Grid2 и Grid3. Основная цель - определить стратегии, которые надежно работают в разных сценариях; т. е. попытаться найти компромисс, который учитывает все наши тестовые случаи.

3.7.2 Производительность стратегий

Данные о производительности различных комбинаций стратегий распределения с алгоритмом локального планирования *EASY* представлены ниже.

Таблица 3.4 показывает среднюю деградацию девяти стратегий распределения в Grid1 и Grid2, и Grid3 учитывая 10 часто используемых метрик производительности.

Таблица 3.4 – Процент деградации производительности Grid1, Grid2 и Grid3

Метрика		Стратегия				
		MCT	*MLB*	*LBal_S*	*MLp*	*MPL*
ρ	Grid1	0	0	0	2	0
	Grid2	0	0	0	0	0
	Grid3	2	22	19	87	17
SD	Grid1	1501	469	20	93	0
	Grid2	5734	1031	10	62	0
	Grid3	1	33	13	63	19
SD_b	Grid1	1284	408	18	71	0
	Grid2	4001	754	17	53	0
	Grid3	1	33	10	65	18
Th	Grid1	0	0	0	2	0
	Grid2	0	0	0	0	0
	Grid3	1	18	17	47	14
TA	Grid1	47	17	0	4	0
	Grid2	93	34	1	3	0
	Grid3	1	32	9	65	15
U	Grid1	0	0	0	2	0
	Grid2	0	0	0	0	0
	Grid3	2	18	16	46	15
t_w	Grid1	1256	463	11	119	0
	Grid2	4833	1799	70	153	0
	Grid3	1	32	9	65	16
WTA	Grid1	100	49	0	35	2
	Grid2	135	74	2	19	0
	Grid3	2	28	8	112	16
WTA_w	Grid1	23	5	0	6	0
	Grid2	20	7	1	4	0
	Grid3	2	24	10	102	16
WWT_s	Grid1	378	187	0	130	6
	Grid2	700	384	10	101	0
	Grid3	2	28	8	112	16

Метрика		Стратегия			
		Random	*MST*	$MWWT\_S_a$	*MWT*
ρ	Grid1	21	0	7	0
	Grid2	2	0	0	0
	Grid3	83	0	45	26
SD	Grid1	53741	100	17658	236
	Grid2	17688	110	17609	999
	Grid3	41	0	54	43
SD_b	Grid1	41924	91	12576	204
	Grid2	12196	111	7550	628
	Grid3	46	0	50	42
Th	Grid1	18	0	7	0
	Grid2	2	0	0	0
	Grid3	44	0	27	19
TA	Grid1	1923	3	641	9
	Grid2	345	5	175	18
	Grid3	49	0	46	40
U	Grid1	18	0	7	0
	Grid2	2	0	0	0
	Grid3	46	0	30	20
t_w	Grid1	51929	80	17303	252
	Grid2	18027	252	9132	933
	Grid3	49	0	47	40
WTA	Grid1	2513	0	909	35
	Grid2	457	7	271	55
	Grid3	111	0	13	34
WTA_w	Grid1	815	1	292	8
	Grid2	91	2	65	11
	Grid3	110	0	11	30
WWT_s	Grid1	9483	1	3429	132
	Grid2	2378	39	1411	287
	Grid3	112	0	13	34

В таблице 3.5 показано деградация производительности стратегий по трем основным метрикам.

Таблица 3.5 – Процент деградации производительности стратегий для 3-х Грид и 3-х метрик

Метрика		Стратегия				
		MCT	*MLB*	*LBal_S*	*MLp*	*MPL*
ρ	Grid1	0	0	0	2	0
	Grid2	0	0	0	0	0
	Grid3	2	22	19	87	17
SD_b	Grid1	1284	408	18	71	0
	Grid2	4001	754	17	53	0
	Grid3	1	33	10	65	18
t_w	Grid1	1256	463	11	119	0
	Grid2	4833	1799	70	153	0
	Grid3	1	32	9	65	16

Метрика		Стратегия			
		Random	*MST*	*MWWT_S$_a$*	*MWT*
ρ	Grid1	21	0	7	0
	Grid2	2	0	0	0
	Grid3	83	0	45	26
SD_b	Grid1	41924	91	12579	204
	Grid2	12196	111	7550	628
	Grid3	46	0	50	42
t_w	Grid1	51929	80	17303	252
	Grid2	18027	252	9132	933
	Grid3	49	0	47	40

В таблице 3.6 показано средняя деградация производительности и рейтинг для сценариев Grid1, Grid2 и Grid3.

Таблица 3.6 – Процент деградации производительности стратегий и их ранг для Grid1, Grid2 и Grid3

		MCT	*MLB*	*LBal_S*	*MLp*	*MPL*
Среднее	Grid1	846.7	290.3	9.7	64.0	0.0
	Grid2	2,944.7	851.0	29.0	68.7	0.0
	Grid3	1.3	29.0	12.7	72.3	17.0
Среднее по всем тестам		1,264.23	390.10	17.13	68.33	5.67
Ранг	Grid1	7	6	2	4	1
	Grid2	7	6	2	3	1
	Grid3	2	5	3	9	4
Ранг по всем тестам		7	6	2	4	1

		Random	*MST*	*MWWT_S*	*MWT*
Среднее	Grid1	31291.3	57.0	9962.0	152.0
	Grid2	10075.0	121.0	5560.7	520.3
	Grid3	59.3	0.0	47.3	36.0
Среднее по всем тестам		13808.53	59.33	5190.00	236.10
Ранг	Grid1	9	3	8	5
	Grid2	9	4	8	5
	Grid3	8	1	7	6
Ранг по всем тестам		9	3	8	5

Видно, что в больших Грид стратегии распределения MPL и LBal_S, учитывающие только размеры работ, работают лучше других алгоритмов по метрикам времени ожидания, ограниченного замедления и коэффициента конкуренции. Дополнительная информация, используемая на этапе распределения, такая как доступная на уровне B и C, не помогает построить

лучшие планирования.

В небольших Грид с высокой нагрузкой стратегии распределения MST и MCT, использующие локальную информацию о планировании, также называемую информацией о состоянии ресурсов, немного превосходят другие подходы по тем же метрикам, если используются пользовательские оценки времени выполнения. Однако во всех тестовых случаях MPL и LBal_S демонстрируют преимущества.

Оказывается, что MPL и LBal_S являются самыми эффективными алгоритмами. Помимо производительности, использование MPL и LBal_S не требует дополнительных накладных расходов на управление, таких как запрос информации о локальных распределениях или построение предварительных расписаний брокером Грид. Вывод получен на основе широкого диапазона параметров моделирования и метрик.

Показано, что в больших Грид небольшое ухудшение конкурентного фактора, вероятно, связано с тем, что в рассматриваемых онлайн сценариях и при данных рабочих нагрузках последние поступившие задания определяли сроки выполнения.

3.7.3 Допустимые стратегии распределения работ

Допустимые факторы, используемые во всех экспериментах, находятся в диапазоне от 0,1 до 1 с шагом 0,1. Таблицы 3.7-3.12 показывают результаты допустимого распределения.

Таблица 3.7 – Процент улучшения метрик у стратегий с допустимым распределением, Grid3

Метрика	Стратегия				
	MCT_a	MLB_a	$LBal\_S_a$	MLp_a	MPL_a
ρ	1.25	2.51	0.37	24.79	1.54
	0.5	0.5	0.6	0.5	0.5
SD	1.06	0.37	4.08	65.75	5.68

	0.5	0.6	0.5	0.5	0.5
SD_b	0.96	0.80	3.12	62.69	5.20
	0.5	0.2	0.5	0.5	0.5
Th	0.51	3.14	0.00	34.68	0.0
	0.5	0.5	0.6	0.5	-
TA	1.06	1.11	2.75	57.83	4.79
	0.5	0.5	0.5	0.5	0.5
U	1.12	2.55	0.32	31.25	1.64
	0.5	0.5	0.6	0.5	0.5
t_w	1.06	1.11	2.75	57.88	4.80
	0.5	0.5	0.5	0.5	0.5
WTA	1.41	4.13	2.46	36.06	4.78
	0.5	0.5	0.5	0.5	0.5
WTA_w	1.83	4.22	1.91	29.87	4.31
	0.5	0.5	0.5	0.5	0.5
WWT_s	1.41	4.13	2.46	36.09	4.79
	0.5	0.5	0.5	0.5	0.5

Метрика	Стратегия			
	$Random_a$	MST_a	$MWWT\_S_a$	MWT_a
ρ	22.68	0.78	2.54	3.06
	0.5	0.6	0.6	0.5
SD	5.44	1.78	6.18	12.80
	0.5	0.5	0.2	0.2
SD_b	7.76	1.78	4.49	11.64
	0.5	0.5	0.2	0.2
Th	25.62	1.57	3.35	1.19
	0.5	0.6	0.5	0.5
TA	10.22	1.80	0.67	8.28
	0.5	0.5	0.2	0.2
U	29.43	0.67	2.44	3.01

	0.5	0.6	0.6	0.5
t_w	10.24	1.80	0.67	8.29
	0.5	0.5	0.2	0.2
WTA	29.81	0.67	0.81	7.09
	0.5	0.5	0.5	0.5
WTA_w	30.58	0.76	0.53	7.13
	0.5	0.5	0.6	0.5
WWT_s	29.84	0.67	0.81	7.10
	0.5	0.5	0.5	0.5

Таблица 3.8 – Процент деградации производительности стратегий с $a = 1$ и $a = 0.5$, Grid3

Метрика	Стратегия					
	MCT_a	MCT	MLB_a	MLB	$LBal\_S_a$	$LBal\_S$
ρ	1	2	20	23	19	19
SD_b	79	81	139	139	102	102
SD	62	63	114	115	79	79
Th	1	2	16	19	17	17
TA	43	45	87	89	56	56
U	1	2	16	18	16	16
t_w	44	45	87	90	57	57
WTA	1	3	24	29	9	9
WTA_w	1	3	19	25	11	11
WWT_s	1	3	24	29	9	9
Среднее	23.4	24.9	54.6	57.9	37.5	37.5

Метрика	Стратегия					
	MLp_a	MLp	MPL_a	MLP	$Random_a$	$Random$
ρ	41	87	15	17	41	83
SD_b	0	192	102	114	140	154
SD	0	168	81	91	118	136
Th	29	47	17	14	30	45
TA	0	137	58	66	93	115
U	29	46	13	15	30	46
t_w	0	137	58	66	93	115
WTA	36	113	11	17	49	113
WTA_w	43	104	12	17	47	112
WWT_s	37	114	11	17	49	113
Среднее	21.5	114.5	37.8	43.4	69	103.2

Метрика	Стратегия					
	MST_a	MST	$MWWT\_S_a$	MWWT_S	MWT_a	MWT
ρ	0	0	43	45	22	26
SD_b	76	80	174	176	137	156
SD	59	62	142	143	112	130
Th	0	1	25	28	19	19
TA	41	44	110	111	85	101
U	0	0	29	30	18	20
t_w	41	44	110	111	85	102
WTA	0	1	12	13	25	35
WTA_w	0	1	11	12	22	31
WWT_s	0	1	12	13	25	35
Среднее	21.7	23.4	66.8	68.2	55	65.5

Таблица 3.9 – Процент деградации производительности стратегий с $a = 0.5$, Grid3

Метрика	Стратегия				
	MCT_a	MLB_a	$LBal\_S_a$	MLp_a	MPL_a
ρ	1	20	20	41	15
SD_b	79	139	94	0	102
SD	62	114	73	0	81
Th	1	16	19	29	17
TA	43	87	52	0	58
U	1	16	16	29	13
t_w	44	87	52	0	58
WTA	1	24	6	36	11
WTA_w	1	19	8	43	12
WWT_s	1	24	6	37	11
Среднее	23.4	54.6	34.6	21.5	37.8

Метрика	Стратегия			
	$Random_a$	MST_a	$MWWT\_S_a$	MWT_a
ρ	41	0	43	22
SD_b	140	76	174	137
SD	118	59	142	112
Th	30	0	25	19
TA	93	41	110	85
U	30	0	29	18
t_w	93	41	110	85
WTA	49	0	12	25
WTA_w	47	0	11	22
WWT_s	49	0	12	25
Среднее	69	21.7	43	55

В таблице 3.10 показаны проценты улучшения производительности стратегий с учетом наших трех основных метрик.

Таблица 3.10 – Процент улучшения производительности стратегий с допустимым распределением если $a = 0.5$ (3 метрики)

Метрика	Стратегия				
	MCT_a	MLB_a	$LBal\_S_a$	MLp_a	MPL_a
ρ	1.25	2.51	0.37	24.79	1.54
SD_b	0.96	0.80	3.12	62.69	5.20
t_w	1.06	1.11	2.75	57.88	4.80
Среднее	1.09	1.47	2.08	48.45	3.85
Ранг	9	7	6	1	4

Метрика	Стратегия			
	$Random_a$	MST_a	$MWWT\_S_a$	MWT_a
ρ	22.68	0.78	2.54	3.06
SD_b	7.76	1.78	4.49	11.64
t_w	10.24	1.80	0.67	8.29
Среднее	13.56	1.45	2.57	7.66
Ранг	2	8	5	3

Продолжительность плана сокращается в большинстве стратегий, когда при применении допустимого распределения заданий. MLpa и Randoma уменьшают конкурентные факторы более чем на 20%. Стратегии MSTa и MCTa с наименьшими начальными конкурентными факторами имеют небольшие улучшения (0.78% и 1.25%). Видно, что улучшение производительности при ограниченном замедлении сильно варьируется от 0.8% до 62.69%, а время

ожидания от 0.67% до 57.88%.

MLp и Random показывают лучшие результаты при допустимых ограничениях на распределение работ. Эти ограничения включают требования работ к числу процессоров при принятии решений, помогая добиться лучшего баланса нагрузки. Другие стратегии, использующие размеры работ в их первоначальном определении, имеют значительно меньшие преимущества.

В таблице 3.11 показан рейтинг стратегий с допустимым распределением и без него в Grid3. В этом сценарии, при $a = 1$, MST, MCT, LBal_S и MPL превосходят остальные алгоритмы, обеспечивая плавную деградацию производительности, и способны справляться с различными требованиями. Однако, при допустимом распределении $a = 0.5$, MLp превосходит другие алгоритмы, показывая лучшие результаты.

Таблица 3.11 – Ранг деградации производительности стратегий без ($a = 1$) и с допустимым распределением ($a = 0.5$), Grid3

Метрика	Стратегия				
	MCT	*MLB*	*LBal_S*	*MLp*	*MPL*
$a = 1$	2	5	3	8	4
$a = 0.5$	3	7	4	1	5

Метрика	Стратегия			
	Random	*MST*	*MWWT_S*	*MWT*
$a = 1$	9	1	7	6
$a = 0.5$	8	2	9	6

В таблице 3.12. показаны проценты деградации производительности стратегий при $a = 0.5$ и $a = 1$ с учетом усреднения по трем метрикам.

Таблица 3.12 – Процент деградации производительности стратегий и их ранг для $a = 0.5$ и $a = 1$ (3 метрики), Grid3

Метрика	Стратегия					
	MCT_a	MCT	MLB_a	MLB	$LBal\_S_a$	$LBal\_S$
ρ	1	2	20	23	20	19
SD_b	62	63	114	115	73	79
t_w	44	45	87	90	52	57
Среднее	36	37	74	76	48	52
Ранг	4	5	11	12	6	8

Метрика	Стратегия					
	MLp_a	MLp	MPL_a	MLP	$Random_a$	$Random$
ρ	41	87	15	17	41	83
SD_b	0	168	81	91	118	136
t_w	0	137	58	66	93	115
Среднее	14	131	51	58	84	111
Ранг	1	18	7	9	13	17

Метрика	Стратегия					
	MST_a	MST	$MWWT\_S_a$	MWWT_S	MWT_a	MWT
ρ	0	0	43	45	22	26
SD_b	59	62	142	143	112	130
t_w	41	44	110	111	85	102
Среднее	33	35	98	100	73	86
Ранг	2	3	15	16	10	14

На рисунках 3.5–3.8 показаны рейтинги деградации производительности 18 стратегий: 9 с допустимой схемой распределения допустимом факторе $a = 0.5$ и 9 без нее $a = 1$.

Как и ожидалось, стратегии MSTa, MST, MCTa и MCT обеспечивают хорошую производительность и превосходят другие алгоритмы по коэффициенту аппроксимации. MLpa является лучшим алгоритмом для метрик среднего ограниченного замедления и среднего времени ожидания. MSTa, MCTa, MST и MCT хорошо работают для минимизации этих двух метрик. Они демонстрируют схожее поведение и отличаются на 60% в ухудшении производительности от лучшей стратегии по среднему ограниченному замедлению и на 45% по среднему времени ожидания.

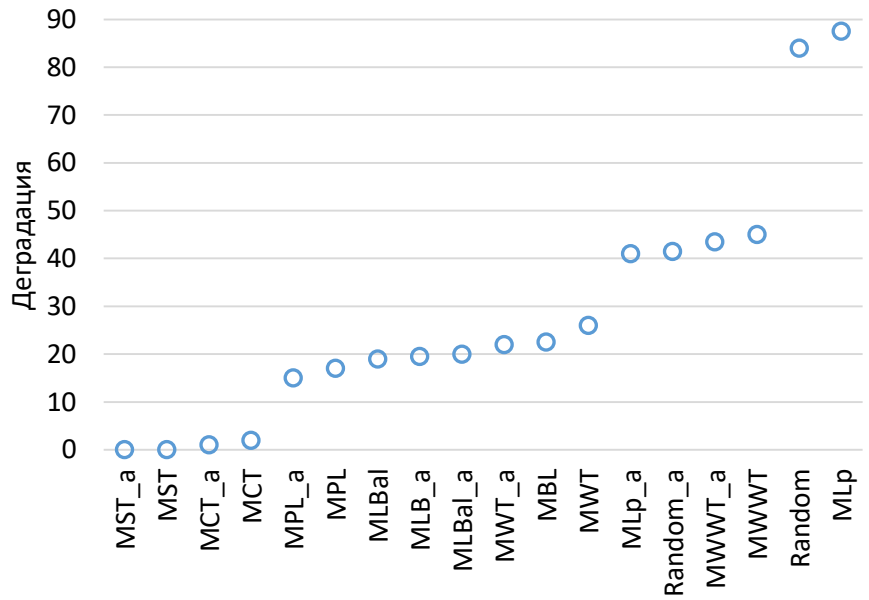


Рисунок 3.5 – Средняя деградация конкурентного фактора (%)

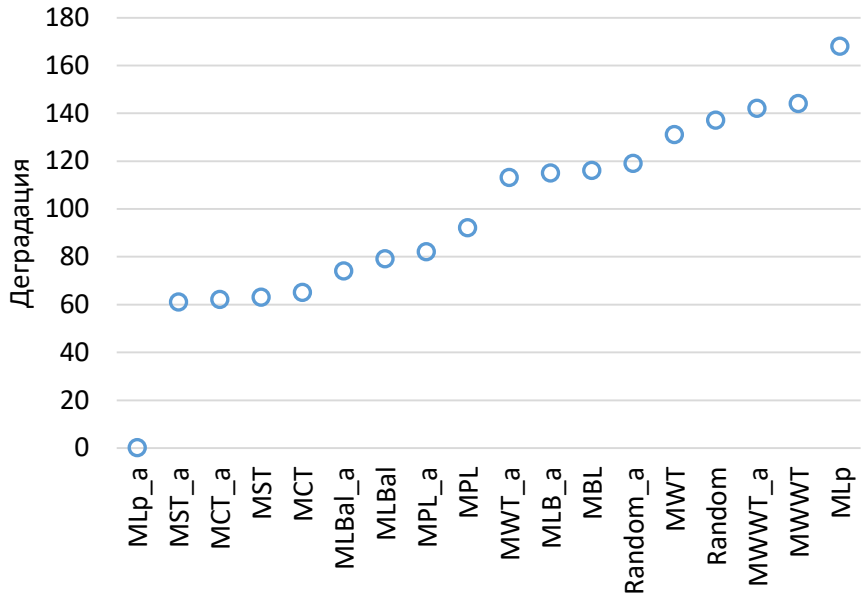


Рисунок 3.6 – Средняя деградация ограниченного замедления (%)

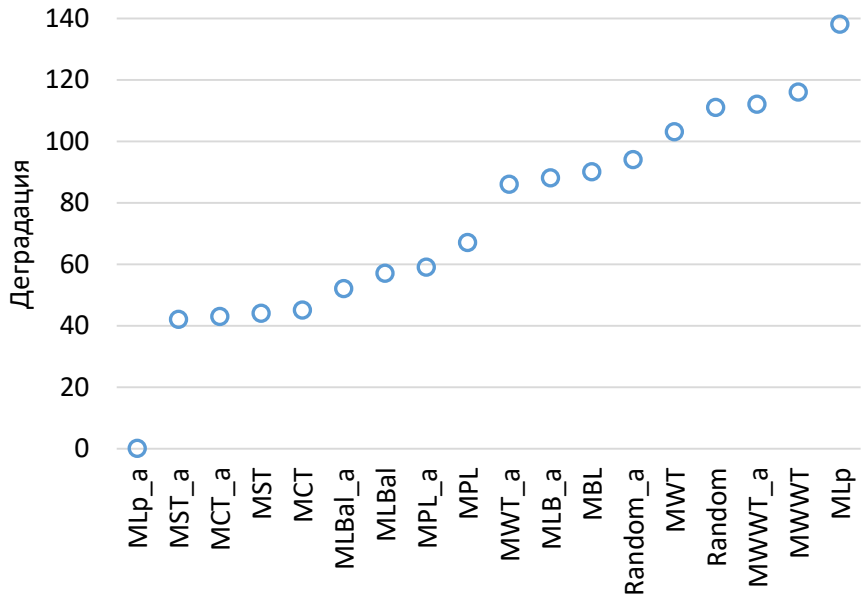


Рисунок 3.7 – Средняя деградация времени ожидания (%)

На рисунке 3.8 показан рейтинг средней деградации с учетом всех тестовых случаев. Видно, что три лучшие стратегии достигаются при допустимом факторе $a = 0.5$.

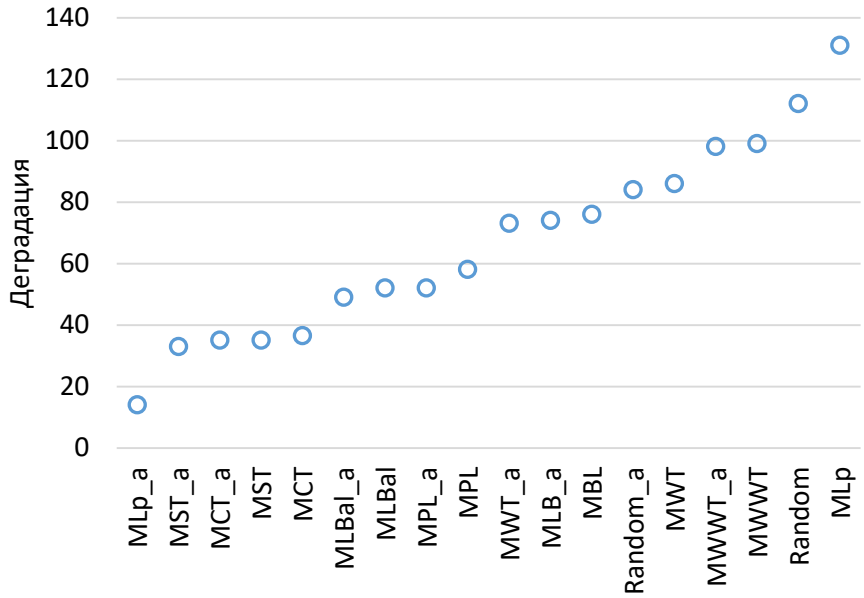


Рисунок 3.8 – Средняя деградация по всем случаям (%)

3.8 Выводы по третьей главе

По мере того, как распределенные системы становятся все более распространенными, проблема распределения ресурсов и планирования имеет решающее значение. Начинают появляться теоретические модели планирования, основанные на анализе наихудших случаев. В этой главе рассматривается онлайн-планирование параллельных работ без прерываний на гетерогенных системах, а также представлена подробная оценка работы алгоритмов.

В результате исследования получены следующие результаты.

- Предложены адаптивные онлайн алгоритмы планирования MLBa+PS и MCTa+PS с конкурентным фактором 11, который улучшает известную границу 17, и аппроксимационным фактором 9.
- При адаптации задачи к задаче, описанной в [78], где все работы могут быть выполнены на самой маленькой машине, получен фактор аппроксимации 3 и конкурентный фактор 5. Алгоритмы требуют минимальной информации и небольшой вычислительной сложности.

- Представлено подробное исследование влияния коэффициент допустимости, включенного в политику планирования, на общую эффективность системы.
- Изучены девять стратегий планирования в зависимости от типа и количества информации, которую они требуют, вместе с алгоритмом локального планирования EASY backfilling. Проведена всесторонняя оценка их практической эффективности с помощью моделирования для трех различных сценариев по десяти метрикам.
- Показано, что с точки зрения рассматриваемых критериев, допустимые стратегии распределения работ превосходят алгоритмы, которые используют все доступные сайты для распределения работ. Адаптивные допустимые стратегии планирования надежны и стабильны даже в сильно различающихся условиях и способны успешно справляться с различными рабочими нагрузками. Коэффициент допустимости может быть динамически скорректирован для того, чтобы справиться с динамической рабочей нагрузкой.

Глава 4. АДАПТИВНОЕ ПЛАНИРОВАНИЕ ПРОСТОЯ МАШИН

Оптимизация параллельных программ труднодостижима классическими методами оптимизации из-за их многообразия и разнообразия реальных параллельных и распределенных платформ и/или сред. Адаптивные алгоритмические схемы, способные динамически изменять распределение работ во время их исполнения для оптимизации поведения глобальной системы, являются лучшей альтернативой для решения этой проблемы. Планирование неклаирвоянтных параллельных работ с известными требованиями к ресурсам, но с неизвестным временем выполнения и акцентом на регулирование периодов простоя предложено в [20, 51].

В данной главе предлагается семейство стратегий планирования, основанных на двух фазах, которые последовательно сочетают последовательное и параллельное выполнение работ. Обобщаются известные предельные границы производительности в наихудшем случае путем введения двух дополнительных параметров, помимо числа процессоров и максимальных требований к процессору, рассматриваемых в литературе, штрафа за распараллеливание работ и коэффициента регулирования простоя процессора. Кроме того, доказывается, что регулирование простоя улучшает гарантию производительности планировщика параллельных работ в режиме разделения пространства.

4.1 Введение

Различные типы параллельных программ обуславливают разнообразные ограничения на политики планирования в зависимости от их динамических и статических характеристик. К сожалению, в большинстве случаев не хватает знаний о программах, чтобы эффективно планировать выполнение работ на основе моделей, используемых в классической теории. Более того, реальные

характеристики приложений слишком сложны, чтобы использовать их в качестве формальных параметров для планировщика. Если параметры программы являются непредсказуемыми или неопределенными, то выделение ресурсов для выполнения работ может быть произведено планировщиком на основе доступной информации, такой как параметры процессоров или операционной системы. Однако их так же очень сложно определить формально.

4.1.1 Параллелизм работ

Проблемы планирования изучались десятилетиями [76, 77, 74, 114]. Внутренний параллелизм приложений добавляет новое измерение к проблеме планирования. Способ разбиения работы на части (когда для выполнения работы требуются несколько процессоров одновременно) зависит от ее свойства делимости. Согласно классификации, предложенной в [93, 112], работы, в зависимости от требований к процессору, разделяются на два общих класса, включающих так называемые жесткие и гибкие работы.

Жесткие работы представляют собой параллельные задачи, требующие для своего параллельного выполнения фиксированного числа процессоров, которое не меняется до тех пор, пока задача не будет завершена.

Молдинговые работы (англ., moldable jobs) могут выполняться на любом числе процессоров. Однако после того, как работе выделено определенное число процессоров, оно остается неизменным на протяжении всего времени ее выполнения [109].

Гибкие (пластичные) работы (англ., malleable jobs) обладают гибким свойством делимости, позволяющем разделять их на любое число сегментов произвольных размеров. Число процессоров, выделенных для работы, не назначается заранее. Оно зависит от числа процессоров, доступных на момент выделения, а также от изменения требований или нагрузки. В любое время, при наличии большего числа процессоров, одна и та же работа может быть прервана,

перераспределена и возобновлена с использованием другого числа процессоров.

Эволюционные работы (англ., evolving jobs) требуют разного фиксированного числа процессоров в различные моменты времени, заданного до начала их выполнения [111].

4.1.2 Концепция управления ресурсами

По мере развертывания параллельных и распределенных систем для поддержки широкого спектра параллельных работ с различными характеристиками и различными требованиями к качеству обслуживания, планирование становится сложной научно-исследовательской проблемой. Разнообразие политик планирования, предназначенных для различных приложений/систем, было изучено в литературе [155]. Очевидно, что практические решения в области планирования работ требуют наличия разнообразных моделей и методик. Управление производительностью систем путем применения непосредственного администрирования или пользовательских настроек политик планирования является непрактичным. Одним из возможных способов решения реалистичных проблем планирования является использование инфраструктуры, которая может поддерживать различные решения и динамически адаптироваться к ним в режиме онлайн.

Общая адаптивная стратегия планирования, называемая (a, b, c)-Scheme предложена в [50, 51, 52]. Эта схема унифицирует планирование последовательных работ, а также работ, для выполнения которых может потребоваться более одного процессора. Она автоматически адаптируется к нужной гранулярности приложения, используя набор из трех параметров: a, b и c, отражающих соответственно характеристики системы, свойства работ и стратегии планирования. Этот механизм является хорошей отправной точкой для понимания процесса унификации различных стратегий динамического планирования. Предварительные результаты показывают, что при использовании этих стратегии

можно разработать хорошие аппроксимационные алгоритмы для планирования параллельных работ.

4.1.3 Штрафной фактор распараллеливания

Для нахождения компромисса между сложностью реальных параллельных систем и желаемой простотой их моделей для теоретического изучения рассмотрим модель параллельных работ, основанную на штрафном факторе. Такая модель использовалась в различных реальных программах [75, 99]. Идея заключается в том, чтобы добавить к времени параллельного выполнения накладные расходы, которые включают в себя время, потерянное на связь, синхронизацию, прерывания и любые дополнительные факторы, которые возникают при управлении параллельным выполнением работы.

Штрафной фактор неявно учитывает некоторые ограничения, когда они неизвестны или их очень сложно определить формально. Он скрывает реальные природу и характеристики приложения или параметры компьютера, хотя и известные, но слишком сложные для использования в качестве формальных ограничений планирования. Его можно рассматривать как отношение идеального ускорения исполнения к наблюдаемому. Штраф за выполнение одной работы или всей рабочей нагрузки также можно оценить на основе эмпирического анализа, сравнительного анализа, расчетов, профилирования, оценки производительности с помощью моделирования, измерения, или на основе информации, предоставленной пользователем.

В модели работы, предложенной в [100], функция ускорения, необходимая для оценки штрафной функции, моделируется с использованием трех параметров: среднего параллелизма работы, дисперсии распараллеливания и числа процессоров. С учетом этих параметров можно построить гипотетический профиль параллелизма и рассчитать ускорение.

В литературе были рассмотрены несколько качественных моделей штрафа

как функции от числа процессоров, выделенных на выполнение работы: константный, линейный и выпуклый [75]. Это наиболее распространенные классы при распараллеливании реальных программ.

Константная форма штрафной функции соответствует системе, в которой приложения достигают почти линейного ускорения с увеличением числа процессоров.

Линейная штрафная функция соответствует логарифмической форме функции ускорения задания, которая демонстрирует ускорение, в основном монотонно поднимающееся до определенного порога и замедляющееся за пределами определенного числа выделенных процессоров.

Выпуклая штрафная функция соответствует небольшим накладным расходам при запуске. Добавление дополнительных процессоров обходится минимально до определенного порога. При превышении порога увеличивается стоимость управления параллельностью, что приводит к снижению скорости работы. Это коррелирует с вогнутой функцией ускорения.

Из анализа исключаются приложения с суперлинейным ускорением, которые могут быть получены для некоторых больших приложений, превышающих кэш или доступную память, а также приложения с ускорением менее единицы (которые не содержат достаточно параллелизма или имеют большие накладные расходы на распараллеливание).

4.1.4 Динамическое регулирование простоя процессоров

Основной принцип планирования – жадный (англ., greedy) метод, который обеспечивает работу процессоров, если есть работы, готовые к выполнению. Стратегии планирования, использующие подходы динамического регулирования простоев для улучшения поведения системы, в последнее время вызвали определенный интерес. Разбиение работ на части для удержания некоторых процессоров в нерабочем состоянии предложено в [193]. Анализ показал его

эффективность для не масштабируемых работ, возможно, с большой вариацией в скорости поступления и исполнения.

В [136] представлена разработка и реализация политики планирования, которая предотвращает снижение производительности из-за слишком раннего назначения процессоров. Она держит некоторые процессоры в нерабочем состоянии в течение некоторого времени перед их назначением. Уменьщение неиспользуемых временных интервалов в расписании также рассматривается в [201].

В этой главе делается упор на простую политику, основанную на планировании, параметризованную для регулирования простоев.

Рассмотрим схему планирования (a, b, c)-Scheme, где стратегии описываются набором параметров [52]. Значение первого параметра a представляет число процессоров, допущенных к простою в системе перед переходом на вторую фазу, b определяет тип работ (жесткие, молдинговые, гибкие работы и др.), c отражает стратегию планирования второй фазы (список, LPT, First Fit, Backfill и т. д.).

Важно отметить, что оптимальные настройки этих параметров очень специфичны для конкретного объекта и зависят от рабочей нагрузки, ресурсов и других факторов вычислительной среды. В реальных параллельных системах значение параметра a может быть выбрано в соответствии с текущей нагрузкой системы и предоставлено планировщиком. Оно зависит от штрафного фактора приложений и, основываясь на измерении характеристик рабочей нагрузки, может адаптироваться к изменению качества рабочей нагрузки.

Изменение параметра a позволяет найти компромисс между отказом от простаивающих процессоров путем более быстрого запуска распараллеливания при большем числе работ (с большими накладными расходами) и задержкой их распараллеливания (вызывающей простои процессоров) до тех пор, пока не будет доступно меньшее число работ.

Улучшение известных наихудших границ производительности может быть достигнуто за счет ограничения максимального числа процессоров, которые

могут оставаться в режиме ожидания.

Известно несколько моделей рабочей нагрузки с наличием информации о выполнении заданий [100]. Однако очевидно, что выбор соответствующей политики должен, по крайней мере частично, определяться поведением фактической параллельной рабочей нагрузки, которая отличается разнообразием и может быть изменена во время исполнения. В (a, b, c)-Scheme используется идея настройки системного параметра a во время выполнения для оптимизации поведения системы, путем измерения характеристик рабочей нагрузки для оценки штрафных факторов. Это может быть сделано с помощью посмертного анализа характеристик работы за определенный промежуток времени.

В последние годы значительное внимание уделяется измерению характеристик рабочей нагрузки для улучшения планирования работы. В работе [179] обсуждается вопрос об измерении в режиме реального времени параметров выполнения задания и ускорения принятия решений планировщиком. В [97] изучено использование измерений рабочего времени для улучшения планирования работы на параллельной машине с акцентом на стратегии, основанные на групповом (англ., gang) планировании.

4.1.5 Регулирование распределения процессоров

Число процессоров, выбираемых пользователем для выполнения работы, как правило, основано на характере задачи или желании пользователя и не учитывает рабочие характеристики многопроцессорной системы. Это число, как правило, подходит для условий легкой нагрузки, но приводит к неприемлемо низкой производительности системы при большой нагрузке [112]. Поскольку пользователи практически не могут знать о загрузке системы, одним из возможных способов оптимизации поведения системы является поддержка различных решений и адаптация к ним во время выполнения.

В работе [122] предложена идея выделять все меньше и меньше

процессоров для каждой работы при увеличении нагрузки, тем самым увеличивая производительность. Максимальное ускорение работы приложения за счет выбора соответствующего числа выделенных процессоров обсуждается в [178], где предложено использовать систему, которая динамически измеряет эффективность работы при различных распределениях и автоматически регулирует процессорное распределение для максимального ускорения выполнения работы.

В [167] представлено исследование динамического планировщика, использующего информацию о простоях работ для динамической настройки распределения процессоров с целью повышения эффективности использования общей системы памяти.

В работе [131] рассмотрена самоадаптирующаяся стратегия планирования в парадигме мастера-рабочего, которая динамически регулирует число процессоров на основе показателей производительности, собранных во время выполнения работ.

В [100] показано, что из нескольких стратегий с эквивалентным временем выполнения, стратегия, уменьшающая распределение при высокой нагрузке, дает наименьшее замедление.

Общая методика виртуализации, имитирующая виртуальную машину с p процессорами на $p'<p$ процессорах и позволяющая выполнять параллельную работу, в которой запрашиваются p процессоров, в то время как на нее выделяются только p' процессоров, приведена в [113]. Данная методика дает хорошие результаты для различных топологий. Двухфазная стратегия регулирования распределения процессоров введена в [189]. Предложен ряд политик группового планирования для параллельных и распределенных систем с различной схемой распределения ресурсов [107, 108, 110, 140, 204, 214].

4.2 Проблема планирования работ с прерываниями

В этой главе анализируются системы планирования, в которых все работы поступают в момент времени 0 и обрабатываются в одной группе. Набор доступных и готовых работ будет выполнен до завершения последней. Все работы, поступившие за это время, будут обработаны в следующей группе. Связь между этой схемой и схемой, когда все работы поступают в разное время, либо в момент их высвобождения, либо в соответствии с зависимостями от предшествующих работ, известна. Она изучается для различных стратегий планирования для общих или ограниченных случаев и приводит к дополнительному коэффициенту 2 [197].

4.2.1 Задача двухэтапного планирования

Рассмотрим набор независимых параллельных работ с целью минимизации общего времени исполнения (makespan) в рамках стратегии двухэтапного планирования. Изучим следующую проблему: задана параллельная машина с m одинаковыми процессорами и набором из n независимых параллельных работ $J = \{J_1, \dots, J_n\}$, время обработки которых $p_1, \dots, p_n$ неизвестно до их завершения. При этом число процессоров k_i, необходимых для выполнения работы J_i, не задано (см. раздел 3) или фиксируется, как только работа поступит в систему (см. раздел 4).

Предположим, что работа независимо от k_i может быть выполнена как на одном процессоре, так и на m процессорах (если k_i не задано) и/или на запрошенных процессорах k_i (если k_i фиксировано).

Реальные системы поддерживают прерывания, поэтому предполагаем, что работа может быть прервана и перераспределена при необходимости. К сожалению, это может привести к большим накладным расходам. Чтобы свести к минимуму число прерываний, ограничим проводимый в диссертации анализ случаем, когда работу можно прервать один раз и назначить ей требуемое число

процессоров. После этого работу нельзя прервать и/или запустить на другом наборе и/или другом числе процессоров. Следовательно, алгоритмы выполняют максимум m прерываний.

Параллельная работа J_i характеризуется тройкой $J_i = \{p_i, k_i, \mu_k^i\}$, а именно: временем ее выполнения на одном процессоре (суммарный объем работы), числом запрошенных процессоров k_i, а также штрафным фактором за ее параллельное выполнение на k_i процессорах.

Время ее параллельного выполнения – $p_k^i = \frac{\mu_k^i p_i}{k_i}$. Предполагаем, что p_k^i не увеличивается при увеличении k_i, по крайней мере, для разумного числа процессоров, и $p_i = p_k^i$. Пусть $\bar{p} = \max_{1 \le i \le n}\{p_i\}$, $\overline{p_\parallel} = \max_{1 \le i \le n}\{p_k^i\}$, $\underline{k} = \max_{1 \le i \le n}\{k_i\}$, и $\bar{k} = \max_{1 \le i \le n}\{k_i\}$.

Вычисляя ускорение S_k^i при выполнении работы J_i на k процессорах как $\frac{p_1^i}{p_k^i}$, штраф μ_k^i учитывается как отношение идеального ускорения к фактическому $\mu_k^i = \frac{k}{S_k^i}$. Предполагается, что штрафной фактор работы J_i не уменьшается от числа выделенных процессоров k: $\mu_k^i \le \mu_{k+1}^i$ для любого $1 \le k < m$. Если работа назначена на один процессор, то $\mu_k^i = 1$.

Пусть $\bar{\mu} = \max_{1 \le i \le n}\{\mu_k^i\}$ и $\underline{\mu} = \max_{1 \le i \le n}\{\mu_k^i\}$. Согласно свойству монотонности работ [154], назначение большего числа процессоров работе уменьшает время ее выполнения, по крайней мере, до определенного порога, но увеличивает ее вычислительную площадь, следовательно

$$\frac{\underline{\mu}\bar{p}}{\bar{k}} \le \frac{\bar{p}\bar{\mu}}{\bar{k}} \le \overline{p_\parallel} \le \frac{\underline{\mu}\bar{p}}{\underline{k}} \tag{4.1}$$

Пусть $W = \sum_{i=1}^n p_i$ – это суммарный объем всех работ и $W_\parallel = \sum_{i=1}^n \mu_k^i p_i$ – суммарный объем работ, выполненных в параллельном режиме, где

$$\underline{\mu} W \le W_\parallel \le \overline{\mu W} \tag{4.2}$$

Пусть $C_\parallel^*$ обозначает продолжительность оптимального расписания, когда допускается распараллеливание работ, а C^* – продолжительность оптимального

расписания, когда распараллеливание работ не допускается. Для стратегии A введем C_A – продолжительность соответствующего расписания. Отметим, что

$$\frac{W}{m}, \bar{p} \tag{4.3}$$

это нижняя граница C^*, и

$$\frac{W_{\parallel}}{m}, \overline{p_{\parallel}} \tag{4.4}$$

это нижняя граница $C_{\parallel}^*$

Предполагая, что рабочая нагрузка состоит, по крайней мере, из m больших рабочих мест (с размером $\bar{k}$ и временем обработки $\overline{p_{\parallel}}$), можно определить, что $W_{\parallel} \geq m\overline{p_{\parallel}}\,\bar{k}$. Тогда справедливо следующее неравенство

$$C_{\parallel}^* \geq \overline{p_{\parallel}}\,\bar{k}. \tag{4.5}$$

Из (4.1) получаем

$$C_{\parallel}^* \geq \overline{p_{\parallel}}\,\bar{k} \geq \underline{\mu}\bar{p}. \tag{4.6}$$

Все стратегии планирования анализируются в соответствии с их аппроксимационными факторами. Пусть $p_{\parallel}(I) = \frac{C_A(I)}{C_{\parallel}^*(I)}$ и $p(I) = \frac{C_A(I)}{C^*(I)}$ обозначают параллельный аппроксимационный фактор алгоритма A для случая I, когда в оптимальном расписании допускается распараллеливание работ, и последовательный аппроксимационный фактор, когда распараллеливание работ когда в оптимальном расписании не допускается.

Для измерения общего качества A определяем его производительность по формулам $\overline{p_{\parallel}} = \sup\left\{\frac{C_A(I)}{C_{\parallel}^*(I)}\right\}$ и $p^* = \sup\left\{\frac{C_A(I)}{C^*(I)}\right\}$ соответственно. Первая величина показывает наихудший случай соотношения между временем исполнения параллельных работ алгоритма A и временем исполнения оптимального варианта.

Последовательный аппроксимационный фактор подчеркивает возможный выигрыш, который можно получить от многопроцессорных работ, в сравнении с классическим подходом, при котором работы являются чисто последовательными.

4.2.2 Двухфазный алгоритм

Рассмотрим влияние регулирования простоев на общее планирование на основе двухэтапной стратегии. Эта стратегия введена в [189] и рассматривает две последовательные фазы. На первом этапе, когда число работ велико, в стратегии последовательно распределяются процессоры без простоев и коммуникаций. Когда достаточное число работ выполнено, она переходит во вторую фазу с многопроцессорным исполнением. При последующем анализе предполагаем, что в первой фазе каждый процессор выполняет одну работу, каждая работа закреплена за одним процессором, таким образом, загрузка процессоров равна 1, а расписание является оптимальным. Неэффективность проявляется, когда остается менее m работ.

В [189] применяется другая стратегия, когда первый процессор простаивает. В этой главе этот подход обобщается путем введения механизма регулирования в режиме ожидания. Таким образом, когда число незадействованных процессоров становится больше a, где $0 \leq a \leq m$, все оставшиеся работы прерываются, и применяется другая стратегия, чтобы избежать слишком большого числа простаивающих процессоров.

Рисунок 4.1 иллюстрирует этот алгоритм, когда на втором этапе (политика распределения $1 - m$) рассматривается специальный тип расписания, называемый групповым расписанием (англ., gang scheduling), см. раздел 4.3. Использование параллельного планирования работ на втором этапе (политика распределения $1 - k$) проиллюстрировано на рисунке 4.6, см. раздел 4.4.

На рисунке 4.1 показаны последовательные фазы: фаза 1a, когда все процессоры заняты, и фаза 1b, когда работают как минимум $m - a + 1$ процессоров (обе фазы используют известную списочную стратегию Грэма); фаза 2, когда один или несколько процессоров простаивают, и, следовательно, переходят ко второй стратегии с параллельными работами.

Параметр a равен наибольшему числу процессоров, которые могут простаивать в системе перед переходом ко второй фазе. Он определяет баланс

между числом работ, обрабатываемых первой и второй стратегиями. Последовательное выполнение работ на первой фазе под большой нагрузкой является оптимальной стратегией, т. к. нет лишних накладных расходов. На втором этапе, когда нагрузка снижается и число простаивающих процессоров становится равным или больше a, система меняет стратегию, чтобы избежать слишком большого количества простоев.

При таком регулировании простоя существует компромисс между более ранним началом распараллеливания, когда a мало (следовательно, распараллеливается больше работ, что приводит к большим накладным расходам на распараллеливание), и задержкой их распараллеливания, когда a большое (что приводит к тому, что больше процессоров остаются без работы), до тех пор, пока не будет доступно меньшее число работ. Эта схема балансирует потребности пользователя (работ) с потребностями компьютерной системы

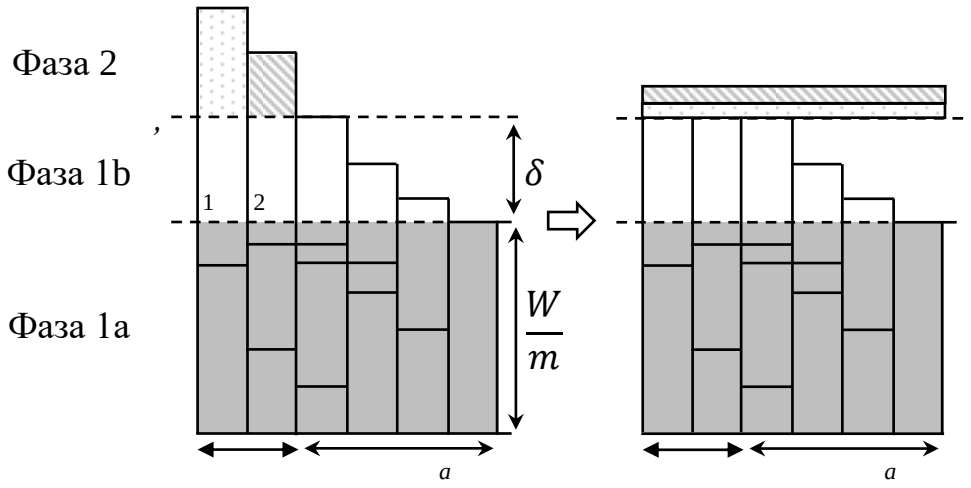


Рисунок 4.1 – Двухфазная стратегия с a=4 и m=6

Предполагаем, что все параллельные работы могут выполняться последовательно, независимо от их типа и числа процессоров, необходимых для выполнения. Это разумное предположение для многих приложений с распределенной и общей памятью, в которых параллельные процессы могут выполняться последовательно (в частности, процессы передачи сообщений, выполняемые одним процессором, или схема балансировки нагрузки, основанная

на общем пуле работ), легко адаптируясь к числу процессоров. Оно подходит даже для жестких работ, которые не могут выдержать сокращения или увеличения числа выделенных процессоров. Такие работы могут эффективно выполняться на одном процессоре за счет использования одного коммуникационного узла процессора.

Но это предположение исключает из нашего рассмотрения некоторые классы параллельных работ, например, синхронные приложения с общей памятью, где параллелизм не может быть эффективно преобразован в последовательное выполнение.

4.3 Анализ политики планирования $1 - m$

В данном разделе приводится анализ двухфазного алгоритма с регулированием простоя для случая, рассматривающего конкретное распределение параллельных работ на 1 и m процессоров. Прежде чем вывести общую границу, рассмотрим некоторые известные результаты для ограниченных случаев $a = 0$ и $a = 1$, рассмотренных в [131].

Случай $a = 0$ соответствует стратегии, которая начинается со второй фазы с распределением по m процессорам. Случай $a = m$ соответствует планированию последовательных работ (применяется только первая фаза).

4.3.1 Коэффициент регулирования простоя $a = 0$

Гарантии производительности любого списочного алгоритма для построения расписания независимых параллельных работ на m процессорах можно найти в [189]. Показано, что $\rho_{\parallel}^{*} \leq \frac{\bar{\mu}}{\underline{\mu}}$ и $\rho^{*} \leq \bar{\mu}$. Если все работы в оптимальном решении распределены на m процессоров $\left(\underline{\mu} = \bar{\mu}\right)$, то $\rho_{\parallel}^{*} = 1$, а если

на один процессор (по нашему предположению $\underline{\mu} = 1$), то $\rho_{\|}^{*} = \rho^{*}$.

Лемма 4.1. Гарантии производительности любого из перечисленных алгоритмов планирования независимых работ на одном процессоре ограничены следующим образом: $\rho^{*} \leq 2 - \frac{1}{\mathrm{m}}$ и $\rho_{\|}^{*} \leq \frac{1}{\underline{\mu}} + \frac{\bar{k}}{\underline{\mu}}(1 - \frac{1}{m})$, а также $\rho_{\|}^{*} \leq \frac{1}{\underline{\mu}}(1 - \frac{1}{m})$ при большой нагрузке.

Доказательство. Доказательство первой границы является простым и опирается в основном на аргументы Грэхема. Интервал простоя возникает, когда процессор не обрабатывает работу. Напомним, что в жадном расписании интервал простоя может возникнуть только в том случае, если ни одна из работ не доступна для обработки.

Пусть W обозначает сумму длин интервалов простоя в расписании в промежутке времени от 0 (врememени начала расписания) до C (конца расписания).

Учитывая что $W \leq (m-1)\bar{p}$ [125], $C = \frac{W+W^{\circ}}{m} \leq \frac{w+(m-1)\overline{p}}{m}$, что аналогично лемме [84]. Поэтому

$\rho_{\|}^{*} \leq \frac{W}{mC_{\|}^{*}} + \frac{\bar{p}}{C_{\|}^{*}}(1 - \frac{1}{m})$, и $\rho_{\|}^{*} \leq \frac{W}{mC_{\|}^{*}} + \frac{\bar{p}}{C_{\|}^{*}}(1 - \frac{1}{m})$.

Из формул (4.3), (4.4) и (4.1) следует, что $\rho^{*} \leq 2 - \frac{1}{m}$ и $\rho_{\|}^{*} \leq \frac{1}{\underline{\mu}} + \frac{\bar{k}}{\underline{\mu}}(1 - \frac{1}{m})$.

Следуя допущению (4.6), получаем $\rho_{\|}^{*} \leq \frac{1}{\underline{\mu}}(2 - \frac{1}{m})$, что доказывает лемму.

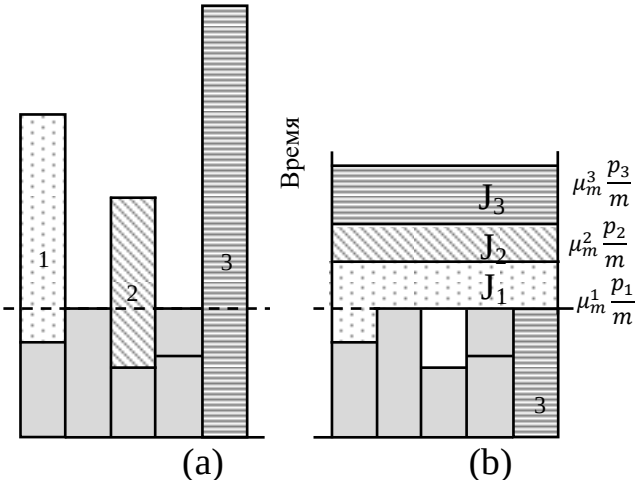


Рисунок 4.2 – Двухфазная стратегия с $a = 1$ и $m = 5$

4.3.2 Коэффициент регулирования простоя $a = 1$

Предположим, что стратегия переключается со стратегии Грэма на стратегию Gang, когда первый процессор простаивает, так что $a = 1$ (рисунок 4.2). Этот случай изучен в [189]. Стратегия использует комбинацию двух экстремальных стратегий планирования Грэма и Gang, комбинацию выделения одного процессора на одну работу и назначения всей машины на одну работу (скоординированная стратегия).

На первом этапе, когда $n \geq m$, каждая работа обрабатывается на одном процессоре (рисунок 4.2a). В момент времени t остается менее m работ. Во избежание простоев процессоров, оставшиеся работы прерываются и назначаются на m процессоров в соответствии со стратегией Gang (рисунок 4.2b).

Гарантии производительности ограничиваются следующим образом:

$$\rho_{\|}^{*} \leq \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{1}{m}\right) + \frac{1}{\underline{\mu} m} \text{ и } \rho^{*} \leq \bar{\mu}\left(1 - \frac{1}{m}\right) + \frac{1}{m}$$

(см. Теорему 4.1, $a = 1$).

4.3.3 Регулирование простоя при $a > 1$

Предположим, что стратегия переключается со стратегии Грэма на стратегию *Gang*, когда a или больше процессоров простаивают.

Теорема 4.1. *Для заданного набора из n независимых работ, каждая из которых имеет штрафной коэффициент от $\underline{\mu}$ до $\bar{\mu}$, назначенных на один и/или на m процессоров, гарантии производительности двухфазной стратегии с регулировкой простоев по параметру a могут быть оценены как:*

$\rho_{\|}^{*} = max\{\rho_{\|}^{*1}, \rho_{\|}^{*2}\}$ *и* $\rho^{*} = max\{\rho^{*1}, \rho^{*2}\}$, *где*

$$\rho_{\|}^{*1} \leq \frac{1}{\underline{\mu}}\left(1 + \frac{a-1}{m}\right), \rho_{\|}^{*2} \leq \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{a}{m}\right) + \frac{a}{\underline{\mu} m},$$

$\rho^{*1} \le 1 + \frac{a-1}{m}$, *и* $\rho^{*2} \le \bar{\mu}\left(1 - \frac{a}{m}\right) + \frac{a}{m}$.

Кроме того, $\rho_{\|}^{*1}$ *и* ρ^{*1} *достижимы в случае* $m \le \frac{a\bar{\mu}-1}{\bar{\mu}-1}$ *и* ρ^{*2} *достижимы в случае* $m > \frac{a\bar{\mu}-1}{\bar{\mu}-1}$.

Доказательство. Ниже приведены фазы алгоритма (рисунок 2):

$1a$ $[0, t)$ Все процессоры заняты

$1b$ $[t, t')$ По крайней мере" работают процессоры m-a+1

2 $[t', t'')$ процессоры простаивают и применяется стратегия Gang

Объем работы, выполненной в фазе 1a, обозначается $W^i = m \cdot t$. Начиная с момента t, остается менее m незавершенных работ. Каждая незавершенная работа, будет выполнена обязательно в фазе 2. Затем, предполагая, что часть этих h работ не была обработана в фазе 1a или 1б, оставшийся объем работы равен $W^2 = \sum_{i=1}^{h}(p_i^{rem} - \delta) \le h(\bar{p} - \delta)$, где p_i^{rem} оставшийся объем работы для работы i, $h \le m - a$, и $\delta = t' - t$.

Время завершения работ $C \le \frac{1}{m}W^1 + \delta + \bar{\mu}\frac{1}{m}W^2$. Два нижних предела оптимального расписания C^* равны δ и $\frac{W^1+(m-a+1)\delta+W^2}{m}$. Следовательно, $\frac{W^1}{m} \le C^* - \frac{(m-a+1)}{m}\delta - \frac{1}{m}W^2$ и $C \le C^* - \frac{m-a+1}{m}\delta + \frac{1}{m}W^2 + \delta + \bar{\mu}\frac{1}{m}W^2$.

Предположем, что h работ, выполнены после t'. Отсюда следует, что

$$C \le C^* - \frac{m-a+1}{m}\delta + \frac{\bar{\mu}-1}{m}(\bar{p} - \delta)h + \delta = C^* + \left(\frac{a-1}{m} - \frac{\bar{\mu}-1}{m}h\right)\delta + \frac{\bar{\mu}-1}{m}h\bar{p}$$

Пусть $B(a) = \frac{a-1}{m}$, $A(h) = \frac{\bar{\mu}-1}{m}h$ и $J(a,h) = (B(a) - A(h))\delta + A(h)\bar{p}$.

Рассмотрим две различные возможности для значений $A(h)$ и $B(a)$.

I. Если $A(h) \ge B(a)$ тогда $J(a,h) \le A(h)\bar{p}$. Следовательно, $C \le C^* + \frac{\bar{\mu}-1h}{m}\bar{p}$,

$C \le \frac{C_{\|}^*}{\underline{\mu}} + \frac{h}{m}\left(\frac{\bar{\mu}}{\underline{\mu}} - \frac{1}{\underline{\mu}}\right)\bar{p_{\|}}\bar{k}$, и $\rho^* \le 1 + \frac{(\bar{\mu}-1)(m-a)}{m} = \bar{\mu}\left(1 - \frac{a}{m}\right) + \frac{a}{m}$.

Из (5) следует, что $\rho_{\|}^* \le \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{a}{m}\right) + \frac{a}{m}$.

II. Если $A(h) < B(a)$ тогда $J(a,h) \leq (B(a) - A(h))\bar{p} + A(h)\bar{p}$. Следовательно, $C \leq C^* + \frac{a-1}{m}\bar{p}$ и $C \leq \frac{C_{\parallel}^*}{\underline{\mu}} + \frac{(a-1)\bar{k}}{\underline{\mu}m}\bar{p}_{\parallel}$.

Из этого следует что $\rho^* \leq 1 + \frac{a-1}{m}$, и $\rho_{\parallel}^{*1} \leq \frac{1}{\underline{\mu}}\left(1 + \frac{a-1}{m}\right)$

$A(h) < B(a)$ когда $\frac{a-1}{m} - \frac{\bar{\mu}-1}{m}h > 0$, следовательно $h < \frac{a-1}{\bar{\mu}-1}$.

Для $h = m - a$, $m - a < \frac{a-1}{\bar{\mu}-1}$ и $m < \frac{a\bar{\mu}-1}{\bar{\mu}-1}$.

Примечание

Для $a = 0$, $\rho^* \leq \bar{\mu}$ и $\rho_{\parallel}^* \leq \frac{\bar{\mu}}{\underline{\mu}}$ соответствуют гарантиям производительности любого алгоритма планирования работ, назначенных на m процессоров (см. раздел 3.1).

Для $= 1$, $\rho^* \leq \bar{\mu}\left(1 - \frac{1}{m}\right) + \frac{1}{m}$ и $\rho_{\parallel}^* \leq \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{1}{m}\right) + \frac{1}{m\underline{\mu}}$ равны границам полученным в [189]. Для $\bar{\mu} = 1$, $\rho^* = \rho_{\parallel}^* = 1$ работы распараллеливаются без накладных расходов. Расписание не имеет интервалов простоя и является оптимальным. Для $a = m$, $\rho^* \leq 2 - \frac{1}{m}$ и применяется только стратегия Грэма, В этом случае, $\rho_{\parallel}^* \leq \frac{1}{\underline{\mu}}\left(2 - \frac{1}{m}\right)$.

На рисунке 4.3 показан аппроксимационный фактор $\rho_{\parallel}^*$ относительно варьирования числа процессоров при фиксированном a, логарифмической форме функции ускорения m (линейный штрафной фактор) и линейной $m/2$ форме ускорения как функции от m (постоянный штрафной фактор 2).

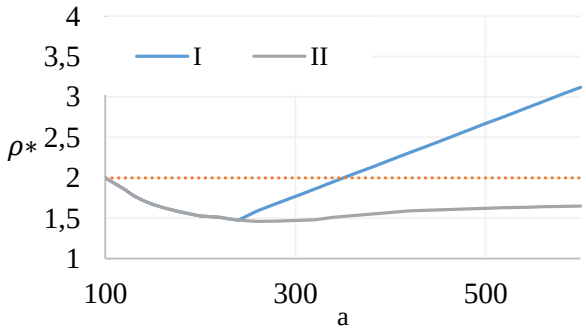


Рисунок 4.3 – ρ^* относительно изменения m; $a = 30$, (I) логарифмическая функция ускорения и (II) линейное ускорение $m/2$

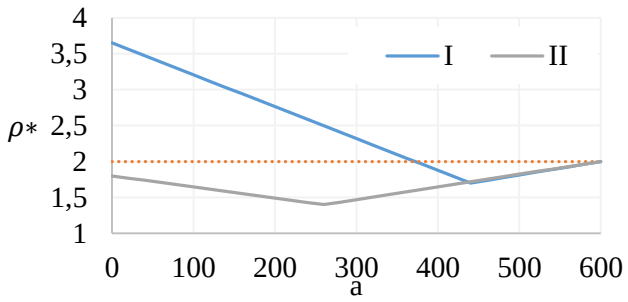


Рисунок 4.4 – $\rho_{\|}^{*}$ относительно изменения a; $m = 600$; (I) логарифмическая функция ускорения и (II) линейное ускорение $m/2$

На рисунке 4.4 представлена производительность, относительно варьирования параметра a, при фиксированном числе процессоров.

Оба рисунка показывают, что настройка простоя по параметру a позволяет получить минимум наихудшей ошибки, связанной с двумя различными характеристиками рабочей нагрузки.

4.4 Анализ политики планирования 1−k

В главе приводится анализ двухфазного алгоритма с распределением по 1 и/или фиксированному числу процессоров с $\underline{k}$ по $\bar{k}$ при регулировании простоя $0 \leq a \leq m$. Анализируется производительность с точки зрения числа процессоров, выделенных на выполнение работы, и его штрафного фактора.

Прежде чем вывести общую границу, рассмотрим известные результаты для ограниченного случая $a = 0$, что соответствует стратегии, которая начинается со второго этапа (планирование жестких работ). Случай $a = m$ соответствует планированию списка для последовательных заданий (применяется только первая фаза).

4.4.1 Регулирование простоя при $a = 0$

Прежде чем углубиться в детали двухфазного алгоритма, сделаем некоторые общие замечания, хорошо известные в литературе.

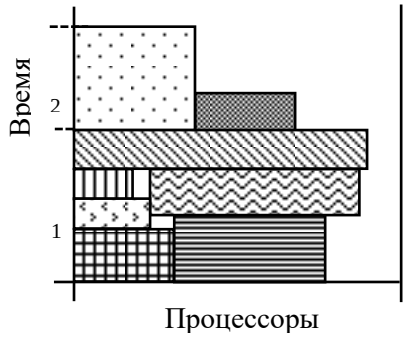


Рисунок 4.5 – Планирование жестких заданий в режиме совместного использования пространства

Лемма 4.2. Гарантия производительности любого списочного алгоритма для планирования независимых параллельных жестких работ имеет следующее ограничение:

$$\rho_{\parallel}^{*} \leq \frac{2m-\bar{k}}{m-\bar{k}+1} \text{ и } \rho^{*} \leq \frac{1}{m-\bar{k}+1}\left(\bar{\mu}m + \frac{\mu}{\underline{k}}\left(m-\bar{k}\right)\right).$$

Доказательство. В расписании есть два вида временных интервалов, которые концептуально могут быть объединены в два последовательных интервала, а именно C_1 и C_2 (рисунок 4.5) [153, 161]. Оба интервала соответствуют временным слотам, когда в нем более чем $\bar{k}-1$ процессора простаивают, и строго более $\bar{k}-1$ процессоров простаивают.

Пусть $W_{\parallel}$ будет общим объемом работы. Учитывая, что C_2 ограничен максимальным временем выполнения работы $\overline{p_{\parallel}}$ (или длиной критического пути в случае ограничений зависимости между работами [153, 161]), а $W_{\parallel} \geq \left(m-\bar{k}+1\right)C_1 + C_2$, получаем верхнюю границу общего времени завершения работ.

$C = C_1 + C_2 \leq \frac{W_{\parallel} - C_2}{m - \bar{k} + 1} + C_2 \leq \frac{W_{\parallel} + (m - \bar{k})\overline{p_{\parallel}}}{m - \bar{k} + 1}$.

Из (4.4) следует, что $\overline{p_{\parallel}} \leq \frac{2m - \bar{k}}{m - \bar{k} + 1}$.

Используя (4.2), (4.3) и (4.1), получаем

$$\rho^* \leq \frac{\bar{\mu} m}{m - \bar{k} + 1} + \frac{\underline{\mu}(m - \bar{k})}{\underline{k}(m - \bar{k} + 1)} = \frac{1}{m - \bar{k} + 1}\left(\underline{\mu} m + \frac{\underline{\mu}}{\underline{k}}(m - \bar{k})\right)$$

Примечание: Результат является обобщением известной границы Грэма. Для последовательных работ получаем границу $2 - 1/m$ (без штрафов: $\underline{\mu} = \bar{\mu} = 1$ и $\underline{k} = \bar{k} = 1$). Оба коэффициента совпадают.

Лемма 4.3. Последовательная гарантия производительности любого списочного алгоритма планирования независимых параллельных и последовательных работ ограничена следующим образом: $\rho^* \leq \bar{\mu} \frac{2m - \bar{k}}{m - \bar{k} + 1}$.

Доказательство. Доказательство следует непосредственно из Леммы 4.2 и предположения, что $\underline{k} = 1$:

$$\rho^* \leq \frac{1}{m - \bar{k} + 1}\left(\underline{\mu} m + \frac{\underline{\mu}}{\underline{k}}(m - \bar{k})\right) \leq \bar{\mu} \frac{2m - \bar{k}}{m - \bar{k} + 1}$$

4.4.2 Регулирование простоя при $a > 0$

Теперь перейдем к случаю, когда стратегия переключается со стратегии Грэма на параллельное планирование работ, когда процессоры начинают простаивать (рисунок 4.6).

Теорема 4.2. *(Последовательная конкурентность). Для заданного набора из n независимых параллельных работ с изменением штрафных коэффициентов, от $\underline{\mu}$ до $\bar{\mu}$, назначенных на фиксированное число процессоров от $\underline{k}$ до $\bar{k}$, последовательный аппроксимационный фактор двухфазной стратегии может быть оценен как:*

$$\rho^* = max\{\rho^{*1}, \rho^{*2}\}, \text{ где}$$

$$\rho^{*1} \le 1 + \frac{a-1}{m}\, u$$

$$\rho^{*2} \le \frac{a}{m} + \frac{1}{m-\bar{k}+1}\left(\underline{\mu}(m-a) + \frac{\underline{\mu}}{\underline{k}}\left(m-\bar{k}\right)\right)$$

Доказательство. Обозначим W^1 объем работ, выполненный до момента t. Каждая работа, не завершенная до t', обязательно выполняется в фазе 2. Затем, предполагая, что часть этих h работ не была обработана в фазе 1a или 1b, оставшийся объем работ будет определяться следующим образом:

$$W^2 = \sum_{i=1}^{h} (p_i^{rem} - \delta) \le h(\bar{p} - \delta),$$

где $h \le m - a$ и $\delta = t' - t$.

Let $p_{\|}^{rem} = \max_{1\le i\le m}\left\{\frac{\mu_k^i (p_i^{rem} - \delta)}{k_i}\right\}$. Следуя Лемме 4.2 в отношении времени завершения фазы 2, верхняя граница общего времени завершения

$$C = \frac{W^1}{m} + \delta + \frac{\bar{\mu} W^2 + (m-\bar{k})\bar{p}_{\|}^{rem}}{m-\bar{k}+1}.$$

Известно, что $W \ge W^1 + (m - a + 1)\delta + W^2$, следовательно

$$C \le \frac{W}{m} - \frac{m-a+1}{m}\delta + \delta + W^2\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right) + \frac{(m-\bar{k})\bar{p}_{\|}^{rem}}{m-\bar{k}+1}.$$

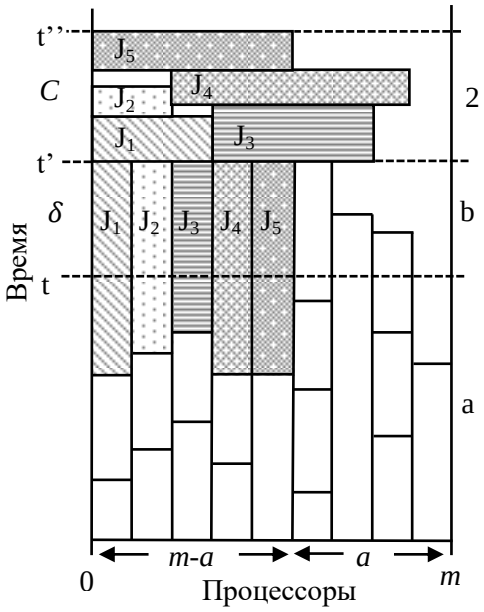


Рисунок 4.6 – Двухфазная стратегия для $a = 4$ и $m = 9$

Рассматривая h работ, выполненных после t', и принимая $\bar{p}_{\parallel}^{rem} \leq \frac{\underline{\underline{\mu}}(\bar{p}-\delta)}{\underline{k}}$, получим, что

$$C \leq \frac{W}{m} + \left(\frac{a-1}{m} - \left(\frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}\right)\right)\delta + \left(\frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}\right)\bar{p}.$$

Пусть $B(a) = \frac{a-1}{m}$, $A(h) = \frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}$, и

$J(a,h) = (B(a) - A(h))\delta + A(h)\bar{p}$. Следовательно, $C \leq \frac{W}{m} + J(a,h)$

Существуют две разные возможности для значений $A(a)$ и $B(a)$.

I. Если $A(h) \geq B(a)$ тогда $J(a,h) \leq A(h)\bar{p}$, и

$C \leq \frac{W}{m} + \left(\frac{h\bar{\mu}}{m-\bar{k}+1} + \frac{(m-\bar{k})\underline{\mu}}{\underline{k}(m-\bar{k}+1)} - \frac{h}{m}\right)\bar{p}.$

На основании (4.3), $\rho^* \leq \frac{a}{m} + \frac{1}{m-\bar{k}+1}\left(\bar{\mu}(m-a) + \frac{\underline{\underline{\mu}}}{\underline{k}}(m-\bar{k})\right)$.

II. Если $A(h) < B(a)$ тогда $J(a,h) \leq \big(B(a) - A(h)\big)\bar{p} + A(h)\bar{p} \leq B(a)\bar{p}$, и $C \leq \frac{W}{m} + \frac{a-1}{m}\bar{p}$. Следовательно $\rho^* \leq 1 + \frac{a-1}{m}$.

Примечание. Для $a = 0$, $\rho^* \leq \frac{1}{m-\bar{k}+1}\underline{\mu}m + \frac{\underline{\underline{\mu}}}{\underline{k}}(m-\bar{k})$ соответствует последовательной гарантии производительности любого списочного алгоритма для расписания параллельных работ (Лемма 4.3).

Для $a = m$ и $\underline{k} = \bar{k} = 1$, $\rho^* \leq 2 - \frac{1}{m}$ применяется только стратегия Грэма (фаза 1).

На рисунке 4.7 показан последовательный аппроксимационный фактор ρ^* относительно варьирования числа процессоров, при фиксированных a, $\bar{k}$ и $\underline{k}$, линейной $\bar{\mu}$ (логарифмической форме функции ускорения работы от m), и постоянной $\bar{\mu}$ (линейной форме функции ускорения работы от m, равной $m/2$).

Предполагается, что число незавершенных работ в фазе 2 равно $m - a$.

На рисунке 4.8 представлен последовательный аппроксимационный фактор при изменении параметра a, фиксированном числе процессоров $\bar{k}$ и $\underline{k}$, линейной и константной $\bar{\mu}$. Оба рисунка показывают, что минимальная граница наихудшего случая ошибки может быть достигнута путем настройки времени простоя процессоров.

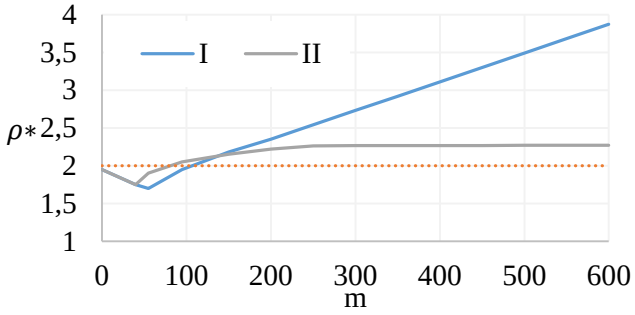


Рисунок 4.7 – ρ^* относительно исменения числа процессоров, $a = 120$, $\bar{k} = 60$ и $\underline{k} = \bar{k}/30$, (I) линейный $\bar{\mu}$ и (II) постоянный $\bar{\mu}$

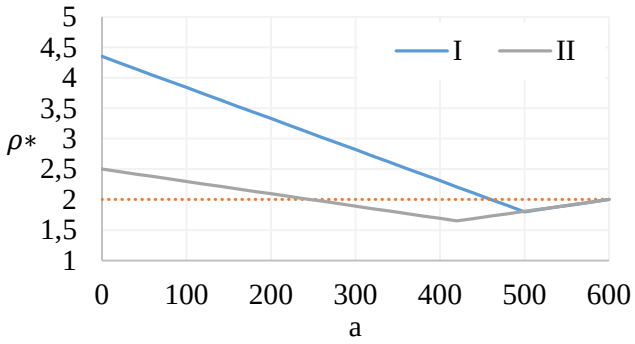


Рисунок 4.8 – ρ^* относительно изменения числа процессоров, $m = 600$, $\bar{k} = 60$ и $\underline{k} = \bar{k}/30$, (I) линейный $\bar{\mu}$ и (II) постоянный $\bar{\mu}$

Теорема 4.3 Последовательная конкурентность (англ., sequential equential competitivity) для смеси работ. Для заданного набора n независимых параллельных и последовательных работ с вариациями штрафных факторов, от 1 до $\bar{\mu}$, назначенных на фиксированное число процессоров, от 1 до $\bar{k}$, последовательный аппроксимационный фактор двухфазной стратегии может быть

оценена как:

$$\rho^* \le \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right).$$

Доказательство. Доказательство вытекает непосредственно из Теоремы 4.2, а также из предположения, что $\underline{k} = 1$. Отметим также, что для любого $m, \bar{k}, \bar{\mu}, \underline{k}$, $A(h) - B(a) = \frac{1}{m-\bar{k}+1}\left(h\bar{\mu} + \frac{(m-\bar{k})\bar{\mu}}{\underline{k}}\right) - 1 + \frac{1}{m} \ge 0$, так что $A(h) \ge B(a)$. Следовательно $\rho^* \le \frac{a}{m} + \frac{1}{m-\bar{k}+1}\left(\bar{\mu}(m-a) + \frac{\bar{\mu}}{\underline{k}}(m-\bar{k})\right)$ и $\rho^* \le \bar{\mu}\frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right)$.

Теорема 4.4. *Для заданного набора из n независимых параллельных работ, назначенных на фиксированное число процессоров, от $\underline{k}$ до $\bar{k}$, с минимальным штрафным фактором $\underline{\mu}$, параллельный аппроксимационный фактор двухфазной стратегии может быть оценен как:*

$$\rho_{\parallel}^* = max\{\rho^{*1}, \rho^{*2}\},$$

где

$$\rho_{\parallel}^{*1} \le 1 + \frac{a-1}{m}u$$

$$\rho_{\parallel}^{*2} \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{\bar{\mu}}{m-\bar{k}+1} - \frac{1}{m}\right).$$

Доказательство. Как показано в Теореме 4.2, время выполнения расписания $C \le \frac{W^1}{m} + \delta + \frac{W_{\parallel}^2 + (m-\bar{k})\bar{p}_{\parallel}^{rem}}{m-\bar{k}+1}$.

Обратите внимание, что

$$W_{\parallel} \ge \underline{\mu}W^1 + (m-a+1)\delta\underline{\mu} + W_{\parallel}^2.$$

Из этого следует

$$C \le \frac{W_{\parallel}}{\underline{\mu}m} + \frac{a-1}{m}\delta + \frac{W_{\parallel}^2}{m-\bar{k}+1} - \frac{W_{\parallel}^2}{\underline{\mu}m} + \frac{(m-\bar{k})\bar{p}_{\parallel}^{rem}}{m-\bar{k}+1}.$$

Отметим, что $W^2 \le h\bar{k}\bar{p}_{\parallel}^{rem}$ and $\bar{p}_{\parallel}^{rem} \le \frac{\underline{\mu}(\bar{p}-\delta)}{\underline{k}}$.

Пусть $\bar{\bar{p}}_{\parallel} = \frac{\mu\bar{p}}{\underline{k}}$, $\bar{p}_{\parallel}^{rem} \le \bar{\bar{p}}_{\parallel} - \frac{\underline{\mu}}{\underline{k}}\delta$, следовательно

$$C \le \frac{W_\parallel}{\underline{\mu} m} + \frac{a-1}{m}\delta + \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\bar{p}_\parallel^{rem},$$

$$C \le \frac{W_\parallel}{\underline{\mu} m} + \frac{a-1}{m}\delta + \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\frac{\underline{\mu}}{\underline{\underline{k}}}\delta + \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\bar{\bar{p}}_\parallel.$$

Пусть $B(a) = \frac{a-1}{m}$, $A(h) = \frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}$, и $J(a,h) = \left(B(a) - A(h)\frac{\underline{\mu}}{\underline{\underline{k}}}\right)\delta + A(h)\bar{\bar{p}}_\parallel$. Значит $C \le \frac{W_\parallel}{\underline{\mu} m} + J(a,h)$.

Рассмотрим две различные возможности для значений $A(h)$ и $B(a)$.

I. Если $A(h)\frac{\underline{\mu}}{\underline{\underline{k}}} \ge B(a)$ тогда $J(a,h) \le A(h)\bar{\bar{p}}_\parallel$ и $C \le \frac{W_\parallel}{\underline{\mu} m} + \left(\frac{h\bar{k}}{m-\bar{k}+1} - \frac{h\bar{k}}{\underline{\mu} m} + \frac{(m-\bar{k})}{m-\bar{k}+1}\right)\bar{\bar{p}}_\parallel$.

Нижними границами оптимального расписания являются (4.4) и (4.6). Следовательно,

$$C_\parallel^* \ge \underline{\mu}\bar{p} \ge \frac{\underline{\mu}}{\underline{\underline{k}}}\bar{p} = \bar{\bar{p}}_\parallel, \text{ и}$$

$$\rho_\parallel^* \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{\underline{\mu} m}\right) \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right).$$

II. Если $A(h)\frac{\underline{\mu}}{\underline{\underline{k}}} < B(a)$ тогда $J(a,h) \le \left(B(a) - A(h)\frac{\underline{\mu}}{\underline{\underline{k}}}\right)\bar{p} + A(h)\bar{\bar{p}}_\parallel$.

Отметим, что $\bar{\bar{p}}_\parallel = \frac{\mu\bar{p}}{\underline{\underline{k}}}$. Тогда получаем $J(a,h) \le B(a)\bar{p}$, и $C \le \frac{W_\parallel}{\underline{\mu} m} + \frac{a-1}{m}\bar{p} \le \frac{W_\parallel}{m} + \frac{a-1}{m}\bar{p}$.

Из (4.6) и (4.4), получаем $C_\parallel^* \ge \underline{\mu}\bar{p} \ge \bar{p}$ и $\rho_\parallel^* \le 1 + \frac{a-1}{m}$, что доказывает теорему.

Примечание. Для $a = 0$, $\rho_\parallel^* \le \frac{2m-\bar{k}}{m-\bar{k}+1}$ соответствует известному аппроксимационному фактору любого списочного алгоритма планирования параллельных работ. Для $\bar{k} = 1$, $\rho_\parallel^* \le 2 - \frac{1}{m}$, применяется только стратегия Грэма.

На рисунке 4.9 показан параллельный аппроксимационный фактор $\rho_\parallel^*$ для различного числа процессоров, с фиксированным a, $\bar{k} = {}^{m}/_{2}$ и константой $\underline{\mu}$ (линейная форма функции ускорения работы от m, равная $m / 2$). Здесь

предполагаем, что число незавершенных работ в фазе 2 равно $m - a$.

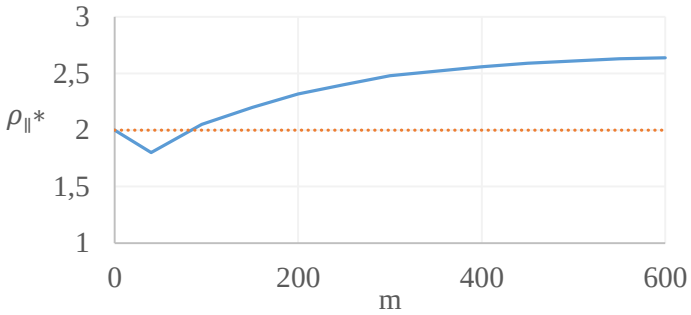


Рисунок 4.9 – $\rho_{\|}^{*}$ относительно изменения числа процессоров, $a = 120$, $\bar{k} = m/2$ и $\underline{\mu}$ – константа

На рисунке 4.10 приведен параллельный аппроксимационный фактор относительно варьирования параметра a, при фиксированном числе процессоров $\bar{k} = {}^{m}/_{2}$ и константе $\underline{\mu}$. Изменение параметра a позволяет получить минимальный предел наихудшей ошибки.

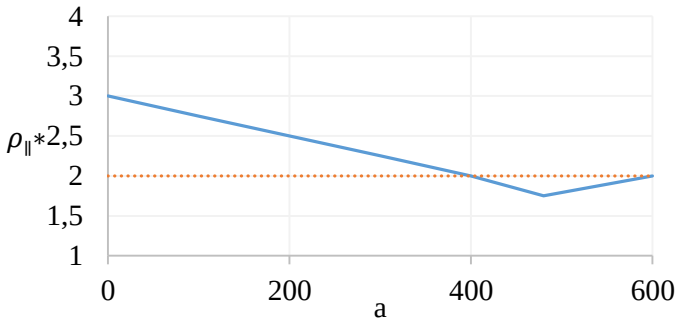


Рисунок 4.10 – $\rho_{\|}^{*}$ варьируя число процессоров, a, $m = 600$, $\bar{k} = m/2$ и константа $\underline{\mu}$

Теорема 4.5. *Конкурентоспособность для смеси работ. Для заданного набора из n независимых параллельных и последовательных работ, назначенных на фиксированное число процессоров, от 1 до* $\bar{k}$*, параллельный аппроксимационный фактор двухфазной стратегии может быть оценен как:*

$$\rho_{\|}^{*} \leq \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right).$$

Доказательство. Доказательство следует непосредственно из Теоремы 4.4 и предположения, что $\underline{k} = \underline{\mu} = 1$. Отметим также, что $A(h) \geq B(a)$ (см. Теорему 4.3). Это означает, что

$$\rho_{\parallel}^{*} \leq \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{\underline{\mu}m}\right) \leq \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right).$$

4.5 Выводы по четвертой главе

В этой главе рассмотрено планирование параллельных работ, с неизвестными временами выполнения, фокусируясь на регулирование периодов простоя машин. Предложены адаптивные схемы, способные динамически изменять алгоритм планирования во время выполнения для оптимизации поведения системы.

Рассмотрены алгоритмы, которые не знают о работах ничего, кроме числа незавершенных работ в системе и требований к процессору. Используется пакетное планирование в рамках общего двухфазного подхода. Оно последовательно сочетает последовательное и параллельное выполнение работ в режиме разделения пространства.

Основной вклад в решение проблемы динамического планирования периодов простоя на параллельных машинах заключается во разработке модели, отражающей некоторые важные аспекты практических задач. Поэтому рассматриваются работы, распараллеливаемые не идеально. Для снижения сложности реальных параллельных систем и упрощения их теоретического изучения применяется модель параллельных работ, основанная на штрафном факторе.

Обобщены известные пределы производительности по худшему случаю, путем введения двух дополнительных параметров: штрафа за распараллеливание работ и фактора регулирования простоя в дополнение к числу процессоров и максимальным требованиям к процессорам, рассмотренным в литературе.

Показано, что настройка параметра для регулирования времени простоя улучшает гарантии производительности планирования. Можно найти компромисс между предотвращением простоев процессоров путем более раннего начала распараллеливания, когда распараллеливается больше задач (увеличивая общие накладные расходы), и задержкой их распараллеливания (приводя к тому, что процессоры остаются без работы).

Таблица 4.1 – Аппроксимационные факторы двухфазного алгоритма при минимизации общего времени выполнения

a	Распределение	ρ^*	Стратегия
0	Gang	$\rho^* = \bar{\mu}/\underline{\mu}$.	Распределение по m процессорам
$1 \le a \le m$	1-m-Gang	$\rho_{\parallel}^* = max\{\rho_{\parallel}^{*1}, \rho_{\parallel}^{*2}\}$ where $\rho_{\parallel}^{*1} \le \frac{1}{\underline{\mu}}\left(1 + \frac{a-1}{m}\right)$ and $\rho_{\parallel}^{*2} \le \frac{\bar{\mu}}{\underline{\mu}}\left(1 - \frac{a}{m}\right) + \frac{a}{\underline{\mu}m}$.	Распределение на 1 и/или m процессоров с холостой регулировкой a
0	Rigid	$\rho_{\parallel}^* = \frac{2m-\bar{k}}{m-\bar{k}+1}$.	Распределение на k (от $\underline{k}$ до $\bar{k}$) процессоров
$1 \le a < m$	1-k-Rigid	$\rho_{\parallel}^* = \max\{\rho_{\parallel}^{*1}, \rho_{\parallel}^{*2}\}$, where $\rho_{\parallel}^{*1} \le \left(1 + \frac{a-1}{m}\right)$, and $\rho_{\parallel}^{*2} \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{\underline{\mu}m}\right)$.	Распределение на 1 и/или k (от $\underline{k}$ до $\bar{k}$) процессоров изменяя a
$1 \le a < m$	1-k-Rigid & serial	$\rho_{\parallel}^* \le \frac{2m-\bar{k}}{m-\bar{k}+1} - a\left(\frac{1}{m-\bar{k}+1} - \frac{1}{m}\right)$.	Распределение на 1 и/или k (от 1 до $\bar{k}$) процессоров изменяя a

Параметр a может быть рассчитан на основе измерения характеристик рабочей нагрузки и адаптирован к изменению качества рабочей нагрузки в режиме онлайн.

Основные результаты можно найти в таблице 4.1. Они показывают, что в рамках предложенной модели можно разработать хорошие алгоритмы аппроксимации планирования параллельных работ, качественное поведение которых можно оценить.

Остаются открытыми несколько проблем: во-первых, теоретический и

экспериментальный анализ регулирования простоев с большим числом вариаций стратегий планирования работ (сначала крупная работа, обратное заполнение и т. д.), а также критерии оптимизации, как системно-ориентированные (утилизация и пропускная способность и т. д.), так и ориентированные на пользователя (время ожидания, время выполнения работ и т. д.). Во-вторых, анализ системы в практической среде планирования, которая поддерживает зависимые работы, а также работы, которые могут прийти в любой момент. Также представляет интерес применение предложенной схемы к набору гибких произвольно распараллеливаемых работ.

Глава 5. ОНЛАЙН ПЛАНИРОВАНИЕ В ОБЛАЧНЫХ СРЕДАХ С РАЗЛИЧНЫМИ УРОВНЯМИ ОБСЛУЖИВАНИЯ

В этой главе рассматриваются проблемы планирования виртуализированных вычислительных ресурсов, предоставляемых пользователям через Интернет в форме облачного сервиса (англ., Infrastructure as a Service – IaaS). В типичном сценарии IaaS поставщик инфраструктуры предлагает свои ресурсы по требованию и с различными уровнями обслуживания своим клиентам. Эти уровни в основном отличаются вычислительной мощностью, которую заказчик гарантированно получит в течение определенного периода времени.

В [15, 42] каждый уровень обслуживания описывается слак-фактором и ценой за единицу времени обработки. Если провайдер принимает работу, она гарантированно выполняется к установленному сроку, т. е. время ее подачи плюс время обработки, умноженное на слак-фактор заданного уровня обслуживания. После того, как работа получена, провайдер должен немедленно и безоговорочно решить, принимает ли он эту работу или отказывается от нее. Предлагаются различные алгоритмы и приводится конкурентный анализ для обсуждения различных сценариев для данной модели. Эти сценарии объединяют фиксированные уровни обслуживания с одной машиной или параллельной машиной.

5.1 Введение

Облачные вычисления в последнее время стали важной вычислительной парадигмой. В рамках IaaS провайдеры облачных вычислений предлагают компьютерные ресурсы клиентам на платной основе. Чтобы удовлетворить потребности различных клиентов и получить достаточный доход, они могут предлагать различные уровни обслуживания. Тогда цена за единицу ресурса

зависит от услуг, выбранных заказчиком. Взамен клиент получает гарантии в отношении предоставляемых ресурсов. Степень этих гарантий определяется ограничениями QoS. Кроме того, соглашения об уровне обслуживания SLA вводятся для того, чтобы юридически оформить отношения клиента с поставщиком. Для соблюдения обещанных гарантий ограничения QoS должны учитываться при планировании выполнения работ.

С одной стороны, поставщик хочет предложить своим клиентам большую гибкость. С другой стороны, провайдеру вряд ли удастся, как правило, включить в соглашение все возможные ограничения QoS. Поэтому провайдер может выбирать различные категории услуг или уровни обслуживания. В то время как большое число уровней обслуживания приводит к высокой гибкости для клиентов, это также приводит к значительным накладным расходам на управление. Следовательно, необходимо найти и при необходимости динамически корректировать подходящий компромисс. Например, системы могут предлагать небольшое число конкретных уровней обслуживания (например, бронзовый, серебряный, золотой) и позволять своим клиентам выбирать эти уровни для каждой работы индивидуально в соответствии со своими потребностями. Поэтому гарантии QoS могут предоставляться только в течение относительно короткого периода времени. Чтобы справиться с такой ситуацией, провайдеру необходимы алгоритмы планирования ресурсов на основе этих нескольких уровней обслуживания. В данной главе предлагаются и анализируются несколько алгоритмов для такого типа планирования.

Смещение акцентов в сторону сервис-ориентированной парадигмы привело к принятию SLA как очень важной концепции. Поставщики и заказчики ищут варианты SLA, наилучшим образом отвечающие их потребностям. Были проведены обширные исследования по различным темам, связанным с SLA. Тем не менее, эти исследования, как правило, выполняются специалистами-практиками, которые ищут подходящие варианты реализации. Мало что известно об эффективности решений по планированию SLA в наихудшем случае.

Несмотря на то, что теоретических результатов по планированию уровней

обслуживания очень мало, результаты из области планирования в режиме реального времени часто имеют существенное сходство. Поэтому модели планирования в реальном времени могут быть использованы в качестве базы для планирования уровней обслуживания. Тем не менее, эти модели должны быть адаптированы таким образом, чтобы, с одной стороны, обеспечить подходящую абстракцию реальности, а с другой стороны, должны соблюсти ключевые свойства вычислений в соответствии с определенными уровнями обслуживания, чтобы обеспечить преимущества для реальных инсталляций.

Поскольку уровни обслуживания часто рассматриваются как продолжение парадигмы реального времени, ориентированной на процессы со сроками, начнем с простой модели, которая использует относительный срок выполнения работы, как функцию времени обработки работы с постоянным фактором уровня обслуживания. Более того, чтобы избежать правовых коллизий, ограничения QoS в соглашениях об уровне обслуживания должны быть легко отслеживаемы. Поэтому такая информация, как крайний срок выполнения работы (директивный срок), зарезервированное время выполнения работы, число предоставленных процессоров и цена за единицу времени, хорошо подходит для включения в SLA. Таким образом, модель действительно представляет собой базовую абстракцию уровня обслуживания в виртуализированной инфраструктуре. Более того, она может быть формализована и обработана автоматически.

5.2 Обзор литературы

Многие исследования рассматривают уровни обслуживания и качество предоставляемого сервиса в сочетании с проблемами планирования. Некоторые авторы описывают и оценивают эвристики [217], другие предлагают и оценивают архитектуры программного обеспечения [182]. Однако не известны исследования, которые бы сосредоточились на теоретическом анализе планирования с уровнями обслуживания в контексте облачных вычислений.

Некоторые теоретические исследования по планированию затрагивают онлайн-планирование с отказом от работ, например, см. [105]. В ней используются работы с равным временем обработки, и поэтому она включает в себя некоторые формы ограничения по времени обработки. Однако целевая функция отличается от предложенных в диссертационном исследовании.

Наиболее близкое соответствие с рассматриваемой проблемой можно найти в области планирования систем реального времени, где моделируются уровни обслуживания с помощью предельных сроков выполнения работ.

Онлайн-планирование последовательных независимых работ в системах реального времени предложено в [72]. В этой работе представлен алгоритм ROBUST, гарантирующий минимальный коэффициент задержки выполнения для каждой работы.

Коэффициент задержки f работы определяется как отношение относительного срока выполнения к требуемому времени исполнения. Он является количественным показателем соблюдения срока выполнения работы. Поэтому он соответствует нашему определению проблемы. ROBUST гарантирует эффективное использование процессора равное $(f-1)f$ в интервале с переполнением, который соответствует предложенному в разделе 5.4 результату. Однако существуют различия в целевых функциях и методах проверки.

В [128] системы жесткого реального времени основываются на концепции С. Баруа и др. Вместо слак-фактора они используют стрейч-фактор. В дальнейшем неоднократно применяются результаты из [128]. Однако в предложенных сценариях каждая работа имеет индивидуальный стрейч-фактор с минимальным значением, в то время как в диссертационном исследовании используется ограниченное число фиксированных стрейч-факторов. Более того, на разных уровнях обслуживания существуют разные цены на вычислительные устройства, в то время в [128] оптимизируются суммарное время обработки, соответствующее только одной цене. Наконец, в диссертации расширяются предложенные в [128] доказательства путем учета ограниченного времени обработки.

5.3 Планирование облачного сервиса с учетом уровня обслуживания

В этой главе рассмотрим одно машинную модель и параллельную однородную модель (P_m). Система предлагает один или несколько уровней обслуживания S_i. Уровень обслуживания S_i связан со слак-фактором $f_i > 1$ и ценой единицы времени обработки u_i.

Поскольку работы поступают со временем ($r_{j,online}$), время обработки p_j и запрашиваемый уровень обслуживания S_i работы J_j становятся известны только при ее поступлении $r_j \geq 0$. Кроме того, работа J_j имеет крайний срок выполнения $d_j = r_j + f_i \cdot p_j$. При этом f_i является слак-фактором или стрейч-фактором уровня обслуживания S_i, см. рисунок 5.1.

Интуитивно понятно, что работа с f_i будет получать в среднем не менее $1/f_i$ ресурсов машины в интервале времени от поступления до завершения работы. Непосредственно в момент получения работы поставщик инфраструктуры должен решить, будет ли это задание принято к исполнению или же будет отклонено. Чтобы сохранить ранее предоставленные гарантии уровня обслуживания, работа J_j может быть принята только в том случае, если существует такое расписание, что ни J_j, ни какая-либо ранее принятая работа не выходят за пределы своих сроков, т.е. для всех принятых работ J_j выполняется $C_i \leq d_i$, при этом C_i - это время окончания работы J_j.

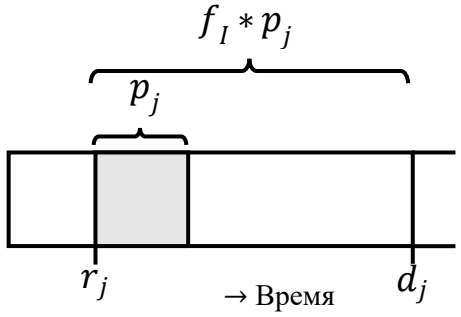


Рисунок 5.1 – Работа и слак-фактор

Целевая функция представляет собой функцию поставщика инфраструктуры, который хочет максимизировать свой общий доход. Работа J_i с уровнем обслуживания S_i генерирует доход $v_j = x_j \cdot p_j \cdot u_i$. Двоичная переменная $x_j \in \{0, 1\}$ обозначает, принимается ли работа J_j $(x_j = 1)$ или нет $(x_j = 0)$.

Используя обозначение трех полей, можно описать проблему либо как $1|prmp, r_{j,online},\ S_i|\sum v_j$ либо как $P_m|prmp, r_{j,online},\ S_i|\sum v_j$ в зависимости от выбранной модели машины. Заметим, что проблема требует максимизации целевой функции вопреки многим другим проблемам планирования.

Для решения этих задач применим конкурентный анализ, введенный в [202] для обсуждения алгоритмов. Метод максимизации A имеет конкурентный фактор $c_V(A) \in [0,1]$ если $c_V(A) = \min_I \frac{V(A)}{V^*} \leq 1$ для всех входных экземпляров I. При этом $V(A)$ и V^* являются совокупным доходом, полученным методом A, и оптимальным совокупным доходом соответственно.

За счет максимизации дохода больший конкурентный фактор лучше, чем меньший. Всякий раз, когда обсуждаются общие верхние пределы для конкурентных факторов, обсуждаются c_V.

После того, как поставщик инфраструктуры принял решение о приемке и размещении работы на машине, используется алгоритм выполнения, основанного на приоритете работ с самым ранним сроком окончания (англ., Earliest Due Date first – EDD) для каждой машины в отдельности. Следует обратить внимание, что EDD алгоритм назначает машину на работу в любой момент времени, в то время как на практике ресурсы машины могут быть назначены на различные работы параллельно, например, если машина является многоядерным процессором.

Тем не менее, алгоритм EDD с прерываниями хорошо подходит для нашей цели, т. к. он прост в применении и дает оптимальное решение для проблемы $1\left|prmp, r_{j,online}\right|L_{max}$ [183].

В общем случае задержка L_j работы J_j в расписании S определяется как $\max\{C_j - d_j, 0\}$. Кроме того, $L_{max} = \max_j\{L_j\}$. Помните, что для всех расписаний

машин в наших проблемах, $L_{max} = 0$, т. к. ни одно задание не может опоздать. Более того, EDD алгоритм производит жадное расписание и поэтому не откладывает использование ресурсов на будуще, когда они могут понадобиться.

5.4 Один уровень обслуживания – одна машина

Сначала рассмотрим простой сценарий с одним уровнем обслуживания S_I и одной машиной, – сокращенно SSL-SM. Результаты этого сценария полезны для более реалистичных сценариев. Этот сценарий является особым случаем планирования в жестком режиме реального времени.

5.4.1 Конкурентный фактор SSL-SM

Для рассмотрения практических IaaS сред введем минимально возможное время обработки $p_{min} > 0$ и максимально возможное время обработки p_{max} любой заданной работы, т.е. предполагаем, что провайдер устанавливает некоторые ограничения на время выполнения работы. Более того, допускаем рациональные слак-факторы, но ограничиваем их степень так, что либо $f_I = \lfloor f_I \rfloor$ либо $f_I - \lfloor f_I \rfloor \gg \frac{p_{min}}{p_{max}}$. За исключением использования p_{min} и p_{max}, Теорема 5.1 очень похожа на Теорему 2 в работе [128] и использует одну и ту же идею доказательства. Тем не менее, повторяем доказательство, т. к. используем аналогичную концепцию в более поздних разделах.

Теорема 5.1. *$c_V \leq 1 - \left(1 - \frac{p_{min}}{p_{max}}\right) \cdot \frac{1}{f_I}$ справедливо для SSL-SM с сервисным уровнем S_I.*

Доказательство. Оппонент отправляет в момент времени $r_0 = 0$ работу J_0 со временем обработки $p_0 = p_{min}$. Если J_0 не принимается, то оппонент не подает другой работы, в результате чего $c_V = 0$.

Если J_0 принята, $f_I \neq \lfloor f_I \rfloor$, то оппонент немедленно представляет другую работу $J_{\lceil f_I \rceil}$

$$r_{\lceil f_I \rceil} = 0$$

$$p_{\lceil f_I \rceil} = (f_I - \lfloor f_I \rfloor) \cdot p_{max}$$

$$d_{\lceil f_I \rceil} = f_I \cdot (f_I - \lfloor f_I \rfloor) \cdot p_{max} \gg d_0 = f_I \cdot p_{min}.$$

Если работа $J_{\lceil f_I \rceil}$ не принимается, то оппонент не подает дополнительную работу, и получаем

$$c_V \leq \frac{p_{min} \cdot u_I}{\left((f_I - \lfloor f_I \rfloor) \cdot p_{max}\right) \cdot u_I} = \frac{p_{min}}{(f_I - \lfloor f_I \rfloor) \cdot p_{max}}.$$

Впоследствии предполагаем, что работа $J_{\lceil f_I \rceil}$ принята. Если $f_I = \lfloor f_I \rfloor$, просто удаляем условия, соответствующие работе $J_{\lceil f_I \rceil}$. Далее оппонент представляет $\lfloor f_I \rfloor$ идентичные работы J_j с $1 \leq j \leq \lfloor f_I \rfloor$ и

$$r_j = 0$$

$$p_j = p_{max}$$

$$d_j = f_I \cdot p_{max} > d_{\lceil f_I \rceil}$$

Одна работа J_j с $1 \leq j \leq \lfloor f_I \rfloor$ должна быть отклонена, так как

$$p_0 + p_{\lceil f_I \rceil} + \lfloor f_I \rfloor \cdot p_{max}$$

$$= p_{min} + (f_I - \lfloor f_I \rfloor) \cdot p_{max} + \lfloor f_I \rfloor \cdot p_{max}$$

$$= p_{min} + f_I \cdot p_{max}$$

$$= d_1 + p_{min} = \cdots = d_{\lfloor f_I \rfloor} + p_{min}.$$

Однако в оптимальном графике работа J_0 будет отклонена, а работы J_1 до $J_{\lceil f_I \rceil}$ принимаются, т. к. все они могут быть завершены вовремя. Поэтому имеем

$$c_V \leq \frac{\left((f_I - 1) \cdot p_{max} + p_{min}\right) \cdot u_I}{f_I \cdot p_{max} \cdot u_I} = 1 - \left(1 - \frac{p_{min}}{p_{max}}\right) \cdot \frac{1}{f_I}$$

Комбинация из трех вариантов дает

$$c_V \leq \max\left\{0, \frac{p_{min}}{(f_I - \lfloor f_I \rfloor) \cdot p_{max}}, 1 - \left(1 - \frac{p_{min}}{p_{max}}\right) \cdot \frac{1}{f_I}\right\} = 1 - \left(1 - \frac{p_{min}}{p_{max}}\right) \cdot \frac{1}{f_I}.$$

5.4.2 Алгоритм SSL-SM

Для SSL-SM в [128] предложен простой алгоритм, который называется жадным принятием. Этот алгоритм принимает каждую новую работу, если эта работа и все ранее принятые работы могут быть выполнены вовремя. Теорема 5.2 похожа на Теорему 3 в [128] и показывает, что конкурентный фактор жадного принятия соответствует верхней границе Теоремы 5.1 для неограниченного времени обработки. Однако в этом разделе рассматривается только один фиксированный слак-фактор, в то время как в [128] допускаются произвольные факторы до тех пор, пока они превышают минимальное значение. Поскольку рассматриваемая в диссертации проблема является частным случаем проблемы жесткого реального времени, то можно применить более простое доказательство, идея которого также будет использована в более поздних разделах.

Теорема 5.2. *Жадное принятие имеет конкурентный фактор* $1 - \frac{1}{f_I}$ *для SSL-SM с уровнем обслуживания* S_I.

Доказательство. Рассмотрим интервал $[0, \max_j d_j)$. Этот интервал может быть разбит на последовательность интервалов таким образом, что каждый интервал $[a, e)$ состоит из субинтервала $[a, b)$ без простоя в расписании и субинтервала $[b, e)$ с простоем.

Любой интервал простоя, начинающийся в момент времени 0, игнорируется, т. к. в этом интервале нет работы, которая может быть выполнена из-за жадной политики принятия и свойства EDD расписания.

Рассмотрим произвольный интервал $[a, e)$, как определено выше. Допустим, имеется работа J_0 с $a \le r_0 < C'_{max}$. C'_{max} это конец расписания в момент подачи J_0. Очевидно, что $C'_{max} \le b$.

$$r_0 + p_0 \cdot f_I < C'_{max} + p_0 \Rightarrow p_0 < \frac{C'_{max} - r_0}{f_I - 1} \Rightarrow$$

$$p_0 \cdot f_I < (C'_{max} - r_0) \cdot \frac{f_I}{f_I - 1} \qquad (5.1)$$

является необходимым условием для отказа от работы J_0.

Следует обратить внимание, что условие 5.1 не является достаточным, т. к. прерывание может допускать принятие J_0 хотя условие 5.1 не выполняется.

В связи с условием 5.1, работа J_1 с $a \le r_1 \le C'_{max} \le b$ и дедлайном

$$d_1 > r_1 + (C'_{max} - r_1) \cdot \frac{f_I}{f_I - 1} = C'_{max} \cdot \frac{f_I}{f_I - 1} - \frac{r_1}{f_I - 1}$$

не отвергается жадным принятием.

Следовательно, в интервале $[a, e)$ не может быть ни одного суб-интервала больше интервала $\left[a, \max\left\{e, b \cdot \frac{f_I}{f_I - 1} - \frac{a}{f_I - 1}\right\}\right)$ для любой политики принятия. В результате для всех интервалов у нас есть

$$c_V(G_{SSL-SM}) \ge \frac{b - a}{b \cdot \frac{f_I}{f_I - 1} - \frac{a}{f_I - 1} - a} = \frac{f_I - 1}{f_I} = 1 - \frac{1}{f_I}.$$

5.5 Один уровень обслуживания - несколько машин

Далее расширим инфраструктуру на множество идентичных машин, в то время как существует только один уровень обслуживания. В [128] показано, что верхняя граница одной машины также распространяется на параллельную идентичную модель машины. Поэтому лишь набросаем альтернативное простое доказательство для нашего особого случая одного слак-фактора.

5.5.1 Конкурентный фактор SSL-PM

Теорема 5.3. *$c_V \le 1 - \frac{1}{f_I}$ справедливо для SSL-PM с уровнем обслуживания S_I, если соотношение между самым большим и наименьшим временем обработки работы может быть произвольно большим.*

Доказательство. Рассмотрим только работы, которые представлены в

момент 0. Однако надо помнить, что провайдер должен решить, принимается ли работа, прежде чем он узнает о следующей работе. Если каждая машина имеет хотя бы одну работу, то оппонент представляет $m \cdot f_I$ работ с достаточно большим временем обработки p_∞ и сроком $f_I \cdot p_\infty$. Только $f_I - 1$ из этих работ могут быть выполнены на каждой машине с $c_V \leq 1 - \frac{1}{f_I} + \delta$ с $\delta \to 0$ для $p_\infty \to \infty$ (см. Теорему 5.1).

Таким образом, алгоритм должен держать по крайней мере одну машину в режиме простоя и гарантировать $c_V > 1 - \frac{1}{f_I} m - 1$ используя только оставшиеся машины. Далее неоднократно применяем один и тот же аргумент к оставшимся машинам, пока не получим одну машину, вызывающую противоречие с Теоремой 5.1.

Если время обработки работ ограничено, то нельзя применить концепцию доказательства предложенную в [128]. Тем не менее, можем получить немного большую верхнюю границу для SSL-PM с ограниченным временем обработки.

Теорема 5.4. $c_V \leq \frac{f_I}{1+f_I \cdot (1-\frac{p_{min}}{p_{max}})}$ справедливо для SSL-PM с уровнем обслуживания S_I и слак-фактором $f_I < \frac{p_{max}}{p_{min}}$

Доказательство. В момент 0, оппонент представляет идентичные работы с небольшим временем обработки p_{min} пока $k \cdot f_I$ работ были приняты для $k < m$ или $m \cdot f_I$ работ были посланы. Следует обратить внимание, что f_I из этих небольших работ могут быть выполнены на каждой машине. Если меньше, чем $k \cdot f_I$ работ принимаются, то противник не представляет каких-либо дополнительных работ, в результате

$$c_V < \frac{k}{m}. \tag{5.2}$$

В противном случае оппонент немедленно посылает $m \cdot f_I$ *(больших*) работ со временем обработки p_{max} и датой выпуска 0. f_I больших работ могут быть назначены на машину, которая не выполняет какую-либо небольшую работу, в то время, как только $f_I - 1$ из этих работ может быть назначены на любую другую

машину.

Таким образом, максимум $m \cdot f_I - k$ больших работ могут быть приняты, в то время как в оптимальном расписании все малые работы отклоняются, а все большие - принимаются. Это приводит к

$$c_V \leq \frac{(m \cdot f_I - k) \cdot p_{max} + k \cdot f_I \cdot p_{min}}{m \cdot f_I \cdot p_{max}} = \frac{m \cdot f_I - k}{m \cdot f_I} + \frac{p_{min}}{p_{max}} \cdot \frac{k}{m} \quad (5.3)$$

Вместе из неравенств 5.2 и 5.3 получается

$$c_V \leq \min\left\{\frac{k}{m}, \frac{m \cdot f_I - k}{m \cdot f_I} + \frac{p_{min}}{p_{max}} \cdot \frac{k}{m}\right\}.$$

Для $k = \frac{f_I}{1 + f_I \cdot \left(1 - \frac{p_{min}}{p_{max}}\right)} \cdot m$, правые стороны неравенства 5.2 и 5.3 имеют одно и то же выражение, и получаем

$$c_V \leq \frac{f_I}{1 + f_I \cdot \left(1 - \frac{p_{min}}{p_{max}}\right)}.$$

5.5.2 Алгоритм SSL-PM

Жадное принятие имеет тот же конкурентный фактор для SSM-SM и SSL-PM, см. [128]. Неизвестно, может ли алгоритм использовать возможное ограничение по времени обработки. Хотя результат известен, предоставляем расширение доказательства теоремы 5.1 для параллельной однородной машинной модели, т. к. это доказательство нам понадобится позже в разделе 5.7.

Теорема 5.5. *Жадное принятие имеет конкурентный фактор* $1 - \frac{1}{f_I}$ *для SSL-PM с сервисным уровнем* S_I.

Доказательство. Используем тот же подход, что и в доказательстве Теоремы 5.2. Интервал расписания $[0, \max_j d_j)$ разбит на несколько интервалов. Всякий раз, когда работа поступает на простаивающую машину, начинается новый интервал.

Поэтому в пределах такого интервала невозможно, чтобы за полностью

субинтервалом простоя следовал субинтервал без простоя, т. е. для машины i, интервал $[a, e)$ может быть либо интервалом без простоя $(b_i = e)$, либо интервалом простоя $(b_i = a)$, либо субинтервалом без простоя $[a, b_i)$, за которым следует субинтервал простоя $[b_i, e)$.

Рассмотрим произвольный интервал $[a, e)$, как определено выше, и предположим, что работа J_0 поступает в момент r_0 с $\leq r_0 < b_i$ для всех машин i и что работа J_0 отвергается.

Из доказательства Теоремы 5.2 известно, что на доказательство не влияет различие между субинтервалом без простоя в момент времени r_0 и конечным субинтервалом без простоя.

Поэтому рассматриваем только конечные значения b_i.

$$r_0 + p_0 \cdot f_I < b_i + p_0 \Rightarrow$$

$$p_0 < \frac{b_i - r_0}{f_I - 1} \Rightarrow$$

$$p_0 \cdot f_I < (b_i - r_0)\frac{f_I}{f_I - 1} \leq (b_i - a)\frac{f_I}{f_I - 1} \tag{5.4}$$

должно быть справедливо для всех машин, чтобы отклонить задание J_0.

В связи с условием 5.4, работа J_1 с

$$d_1 > r_1 + (b_i - r_1) \cdot \frac{f_I}{f_I - 1} = b_i \cdot \frac{f_I}{f_I - 1} - \frac{r_1}{f_I - 1}$$

не отклоняется жадным принятием по крайней одной машиной i.

Т. к. $a \leq r_1$, в интервале $[a, e)$ не может быть более крупного субинтервала без простоя, чем интервал $\left[a, \max\left\{e, \min_i\{b_i\} \cdot \frac{f_I}{f_I - 1} - \frac{a}{f_I - 1}\right\}\right)$. Т. к. этот результат соблюдается в течение всех промежутков времени, получаем

$$c_V(G_{SSL-PM}) \geq 1 - \frac{1}{f_I},$$

см. доказательство Теоремы 5.2.

5.6 Несколько уровней обслуживания – одна машина

В целом провайдеры IaaS предлагают несколько уровней обслуживания. В нашей модели эти уровни отличаются слак-факторами и ценой за единицу времени обработки. В отличие от проблемы жесткого реального времени, используем фиксированное число уровней обслуживания и слак-факторов. Кроме того, уровень обслуживания, рассматриваемый в диссертационном исследовании, включает в себя цену единицы времени.

Следует обратить внимание, что утверждения относятся только к двум разным уровням обслуживания S_I и S_{II}. Требуется $1 < f_I < f_{II} < \infty$ и $u_I > u_{II}$ для слак-факторов и ценовых значений соответственно. Иногда вводятся дополнительные ограничения. В большинстве случаев возможно расширение до более чем двух уровней обслуживания, но это приведет к увеличению сложности, не давая дополнительных знаний.

5.6.1 Конкурентный фактор MSL-SM

В этом разделе снова ограничиваемся одной машиной. Этот сценарий сокращенно назван MSL-SM.

Для каждого уровня обслуживания можно использовать Теорему 5.1 для определения верхней границы конкурентного фактора. Очевидно, что наименьшая из этих верхних границ — это верхняя граница конкурентного фактора для MSL-SM. Более того, можем определить еще более сильную границу для некоторых слак-факторов.

Как и в разделе 5.4, предполагаем, что $f_I - \lfloor f_I \rfloor \gg \frac{p_{min}}{p_{max}}$ справедливо для слак-фактора f_I уровня обслуживания S_I.

Теорема *5.6.*

$$c_V \le \max\left\{\frac{\frac{p_{min}}{p_{max}}}{(f_I-1)}, \frac{f_I-1+\frac{p_{min}}{p_{max}}}{f_I-1+\frac{u_I}{u_{II}}}\right\}$$

справедливо для MSL-SM с двумя уровнями обслуживания S_I и S_{II} и $f_{II} < 2$.

Доказательство. Доказательство аналогично доказательству Теоремы 5.1. Оппонент в момент времени $r_0 = 0$ отправляет работу J_0 с временем обработки $p_0 = p_{min}$ и уровнем обслуживания S_{II}. Если J_0 не принимается, то оппонент не представляет другой работы, в результате чего получается

$$c_V = 0 \tag{5.5}$$

В противном случае оппонент немедленно отправляет другое задание J_1 с уровнем обслуживания S_{II} и

$$r_1 = 0$$

$$p_1 = (f_I - 1) \cdot p_{max}$$

$$d_1 = f_{II} \cdot (f_I - 1) \cdot p_{max} \gg d_0 = f_{II} \cdot p_{min}$$

Если работа J_1 не принимается, то оппонент не представляет никакой дополнительной работы, и имеем

$$c_V \le \frac{p_{min} \cdot u_{II}}{(f_I-1) \cdot p_{max} \cdot u_{II}} = \frac{p_{min}}{(f_I-1) \cdot p_{max}} \tag{5.6}$$

В противном случае оппонент отправляет последнюю работу J_2 с уровнем сервиса S_I и

$$r_2 = 0$$

$$p_2 = p_{max}$$

$$d_2 = f_I \cdot p_{max}$$

Отметим, что удовлетворяется отношение

$$d_2 - d_1 = \left(f_I - f_{II} \cdot (f_I - 1)\right) \cdot p_{max} > 0$$

$$\Leftrightarrow f_{II} < \frac{f_I}{f_I - 1}$$

Т. к. функция $\frac{x}{x-1} = 1 - \frac{1}{x-1}$ убывает для $x > 1$, то имеем $d_2 > d_1$ из-за $1 < f_I < f_{II} < 2$ и $\frac{2}{2-1} = 2$.

Следовательно, работа J_2 должна быть отклонена из-за

$$p_0 + p_1 + p_2 = p_{min} + (f_I - 1) \cdot p_{max} + p_{max} = p_{min} + f_I \cdot p_{max} > \max\{d_1, d_2\} = d_2.$$

Однако при оптимальном графике работа J_0 будет отклонена, а работы J_1 и J_2 будут завершены вовремя. Поэтому

$$c_V \leq \frac{((f_I - 1) \cdot p_{max} + p_{min}) \cdot u_{II}}{(f_I - 1) \cdot p_{max} \cdot u_{II} + p_{max} \cdot u_{II}} = \frac{f_I - 1 + \frac{p_{min}}{p_{max}}}{f_I - 1 + \frac{u_I}{u_{II}}} \tag{5.7}$$

Сочетание трех неравенств 5.5, 5.6 и 5.7 дает результат

$$c_V \leq \max\left\{0, \frac{\frac{p_{min}}{p_{max}}}{f_I - 1}, \frac{f_I - 1 + \frac{p_{min}}{p_{max}}}{f_I - 1 + \frac{u_I}{u_{II}}}\right\}$$

Для неограниченного времени обработки,

$$c_V \leq \frac{f_I - 1}{f_I - 1 + \frac{u_I}{u_{II}}} = 1 - \frac{\frac{u_I}{u_{II}}}{f_I - 1 + \frac{u_I}{u_{II}}} < 1 - \frac{1}{f_I} < 1 - \frac{1}{f_{II}}$$

в связи с $u_I > u_{II}$ и $1 < f_I < f_{II}$.

Поэтому граница теоремы 5.6 для сценария с двумя уровнями сервиса и $f_I < f_{II} < 2$ жестче, чем граница теоремы 5.1.

Обратите внимание, что результат теоремы 5.1 получается для $u_I = u_{II}$. Эта связь между теоремой 5.1 и теоремой 5.6 предполагает, что граница теоремы 5.6 также имеет все значения f_I и f_{II}. Однако, эта проблема все еще остается открытой. Определение такой границы становится более сложным для $f_{II} > 2$. Тогда сценарий теоремы 5.6 включает, по крайней мере, три работы, имеющие уровень обслуживания S_{II}: Работа J_0, работа $J_{\lceil f_{II} \rceil}$, и работы с J_1 до $J_{\lfloor f_{II} \rfloor}$. Таким образом, увеличивается число вариантов определения верхней границы для конкурентного фактора.

5.6.2 Алгоритм MSL-SM

Во-первых, оценим производительность многократно используемого жадного алгоритма принятия для нескольких уровней обслуживания.

Теорема 5.7. Жадный алгоритм принятия имеет конкурентный фактор $\frac{u_{II}}{u_I}\left(1-\frac{1}{f_I}\right)$ для MSL-SM с уровнями обслуживания S_I и S_{II}.

Доказательство. Доказательство аналогично доказательству теоремы 5.2. В нем используются те же определения для интервалов $[a,e)$, $[a,b)$ и $[b,e)$. Однако субинтервал $[a,b)$ может содержать работы с различными уровнями обслуживания.

Пусть работа J_0 – это работа с уровнем обслуживания S_I, которая подается в момент r_0 с $a \le r_0 < b$. Обратите внимание, что условие 1 в доказательстве теорема 5.2 также является необходимым условием для отказа от выполнения работы.

В худшем случае существует оптимальный график, в котором интервал $[a,e)$ содержит не пустой субинтервал $\left[a, b+\frac{b-a}{f_I-1}\right)$, в котором выполняются только работы с уровнем сервиса S_I, в то время как алгоритм присваивает интервалу $[a,b)$ только работы с уровнем сервиса S_{II}. Таким образом, имеем

$$c_V(G_{MSL-SM}) \le \frac{u_{II}}{u_I\left(1+\frac{1}{f_I-1}\right)} = \frac{u_{II}}{u_I}\cdot\left(1-\frac{1}{f_I}\right).$$

Конкурентный фактор теоремы 5.7 является достижимым для жадного распределения, если время обработки не ограничено, см. пример 5.8. Но в отличие от SSL-SM и SSL-PM, сравнение между теоремами 5.6 и 5.7 указывает на то, что конкурентный фактор жадного распределения в целом может не быть жестким.

Пример 5.8. Пусть f_I и f_{II} будут целыми числами. В момент времени 0, поступает f_{II} идентичных работ с уровнем обслуживания S_{II} и временем

обработки $p_1 = \frac{(f_I - 1) \cdot p_{max} + \epsilon}{f_{II}}$. Жадное принятие принимает все работы, т. к. их срок $d_1 = f_{II} \cdot p$ равен общему времени обработки. Сразу после этого подается f_I идентичных работ с уровнем обслуживания S_I, временем обработки $p_2 = p_{max}$ и моментом подачи 0. Из-за $d_2 = f_I \cdot p_{max} < d_1 + p_2$ принимаются любые работы с уровнем сервиса S_I, что приводит к соотношению

$$\frac{\big((f_I - 1) \cdot p_{max} + \epsilon\big) \cdot u_{II}}{f_I \cdot p_{max} \cdot u_I} = \frac{u_{II} \cdot (f_I - 1)}{u_I \cdot f_I} + \frac{u_{II} \cdot \epsilon}{f_I \cdot p_{max} \cdot u_I}.$$

Поэтому рассмотрим другой алгоритм под названием ограниченное принятие для MSL-SM. Этот алгоритм использует коэффициент принятия $h_{II} \geq 1$ и принимает работу J_j с уровнем обслуживания S_{II} только в том случае, если $d_i - C_i \geq h_{II} \cdot p_i$ для всех работ J_i с уровнем обслуживания S_{II} и $d_i \geq d_j$. Здесь C_i обозначает время завершения работы J_i в EDD-расписании с прерываниями после принятия работы J_i.

Интуитивно заменяем слак-фактор f_{II} на $h_{II} \leq f_{II}$ при принятии решения о приеме новой работы с уровнем обслуживания S_{II}, в то время как первоначальный слак-фактор активен при приеме работы с уровнем обслуживания S_I.

Рассмотрим работу J_j с уровнем обслуживания S_{II}, которая завершается в момент C_j в расписании, который не выполняет ни одну работу с уровнем обслуживания S_I после времени r_j. Эту работу всегда можно отложить как минимум на время $p_j \cdot (f_{II} - h_{II}) \geq \big(C_j - r_j\big) \cdot \frac{f_{II} - h_{II}}{h_{II}}$, см. рисунок 5.2.

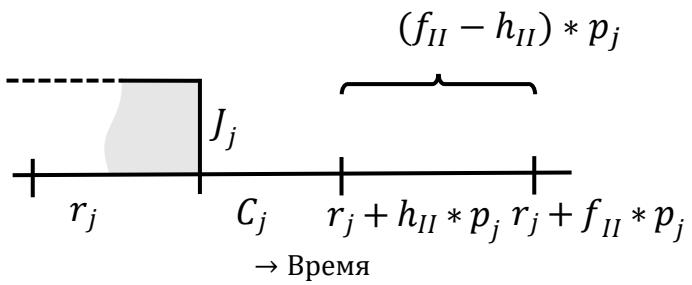


Рисунок 5.2 – Возможная задержка работы с уровнем обслуживания S_{II} из-за коэффициента принятия

Теорема 5.8. *Ограниченное принятие имеет конкурентный фактор* $min\left\{\frac{h_{II}-1}{f_{II}}, \frac{f_I-1}{f_I} - \frac{h_{II}}{f_{II}} \cdot \left(1 - \frac{u_{II}}{u_I}\right)\right\}$ *для MSL-SM с уровнями обслуживания* S_I *и* S_{II}*, и* $h_{II} < f_{II} \cdot \left(1 - \frac{1}{f_I}\right)$.

Доказательство. В доказательстве снова используется та же базовая концепция и те же определения интервалов $[a, e), [a, b), [a, C'_{max})$, и $[b, e)$, что и в доказательстве теоремы 5.2.

Во-первых, рассматриваем пример, в котором представлены только работы с уровнем обслуживания S_{II}. Допустим, работа J_0 с $a \le r_0 < C'_{max}$.

$$r_0 + p_0 \cdot h_{II} < C'_{max} - r_0 + p_0 \Rightarrow$$

$$p_0 < \frac{C'_{max} - r_0}{h_{II} - 1} \Rightarrow$$

$$p_0 \cdot f_{II} < (C'_{max} - r_0) \cdot \frac{f_{II}}{h_{II}-1} \tag{5.8}$$

Условие (5.8) является необходимым для того, чтобы отклонить J_0. В связи с этим, $a \le r_0$, $C'_{max} \le b$ и никакая другая политика принятия не может генерировать в интервале $[a, e)$ субинтервал без простоя больший, чем интервал $\left[a, \max\left\{e, a + (b-a) \cdot \frac{f_{II}}{h_{II}-1}\right\}\right)$. Поэтому в расписании, содержащем только работы с уровнем сервиса S_{II},

$$c_V(R_{MSL-SM}) \ge \frac{h_{II}-1}{f_{II}}. \tag{5.9}$$

Теперь рассмотрим работу J_1 с уровнем обслуживания S_I и $a \le r_1 \le b$. Допустим далее, что в данном расписании все работы в интервале $[r_1, b)$ не должны заканчиваться позже d_1, т. к. части работы, которые могут быть выполнены после d_1, могут быть проигнорированы относительно работы J_1. Кроме того, работы с уровнем обслуживания S_I занимают периоды с общей длиной g в интервале $[a, b)$, см. Условие 5.10.

$$(b - r_0) \cdot \frac{f_{II}}{h_{II}} \le p_0 \cdot f_I < b - r_0 + p_0 + g \Rightarrow \tag{5.10}$$

$$b - r_0 \le p_0 \cdot f_I \cdot \frac{h_{II}}{f_{II}} \Rightarrow$$

$$g > p_0 \cdot \left(f_I \cdot \left(1 - \frac{h_{II}}{f_{II}}\right) - 1\right) \tag{5.11}$$

Условие (5.11) является необходимым для отклонения J_1. Обратите внимание, что использование h_{II} выгодно только в том случае, если g в Условии 5.11 является положительным. Это приводит к неравенству $h_{II} < f_{II} \cdot \left(1 - \frac{1}{f_I}\right)$.

Используя те же аргументы, что и в доказательстве теоремы 5.2, имеем

$$c_V(R_{MSL-SM}) \ge \min_{\text{g}} \frac{u_I \cdot g + u_{II} \cdot (p_0 \cdot (f_I - 1) - g)}{u_I \cdot p_0 \cdot f_I}$$

$$= \min_{g} \frac{g \cdot (u_I - u_{II}) + u_{II} \cdot p_0 \cdot (f_I - 1)}{u_I \cdot p_0 \cdot f_I}$$

$$\ge 1 - \frac{1}{f_I} - \frac{h_{II}}{f_{II}} \cdot \left(1 - \frac{u_{II}}{u_I}\right), \tag{5.12}$$

если в интервале поступает и отклоняется хотя бы одно задание с уровнем обслуживания S_I .

Неравенства 5.12 и 5.9 вместе дают результат этого утверждения.

Обратите внимание, что не легко решить, является ли сильнее неравенство 5.12 или неравенство 5.9.

Давайте сравним результаты теорем 5.7 и 5.9 на примере.

Пример 5.10. Рассмотрим сценарий с двумя уровнями обслуживания S_I, S_{II}, $u_I = 10,\ u_{II} = 1,\ f_I = 2,\ f_{II} = 6,$ и $h_{II} = 2 < \left(1 - \frac{1}{2}\right) \cdot 6$. Жадный прием имеет конкурентный фактор $\frac{1}{10} \cdot \left(1 - \frac{1}{2}\right) = \frac{1}{20}$, в то время как конкурентный фактор для ограниченного приема равет $\min\left\{\frac{2-1}{6} = \frac{1}{6}, 1 - \frac{1}{2} - \frac{2}{6}, \left(1 - \frac{1}{10}\right) = \frac{1}{5}\right\} = \frac{1}{6}$.

5.7 Несколько уровней обслуживания - несколько машин

В этом разделе рассматривается сценарий MSL-PM с различными уровнями обслуживания и параллельными одинаковыми машинами. MSL-PM близко

соответствует реальным требованиям стандартов IaaS, которые, как правило, состоят из множества машин для поддержки масштабирования. Чтобы эффективно использовать большое число машин, поставщик IaaS нуждается во многих независимых клиентах. Для повышения гибкости предлагаются различные уровни обслуживания.

Для анализа алгоритмов MSL-PM используются результаты, полученные в разделах 5.6 и 5.5. Сначала рассмотрим жадное распределение. Как и в разделе 5.6, ограничимся двумя разными уровнями обслуживания, хотя результаты могут быть обобщены за счет повышенной сложности.

***Теорема* 5.9.** Жадное принятие имеет конкурентный фактор $\frac{u_{II}}{u_I}\left(1-\frac{1}{f_I}\right)$ для MSL-PM с уровнями обслуживания S_I и S_{II}.

Доказательство. Доказательство использует теорему 5.5 и теорему 5.7. Доказательство теоремы 5.5 расширяет определения интервалов с одной машины на несколько машин.

Теорема показывает, что жадное распределение не может дать выигрыш от использования нескольких машин, т. к. требуемые свойства интервала должны храниться для каждого интервала в отдельности. Для каждой машины применяем доказательство теоремы 5.7.

Более того, можно расширить пример 5.8, чтобы показать, что конкурентный фактор теоремы 5.11 достижим для жадного распределения.

Тем не менее, можно дать некоторым машинам более высокий приоритет в обслуживании, чтобы получить лучший результат. Точнее, работы с уровнем обслуживания S_I могут быть распределены на каждую машину, в то время как работы с уровнем обслуживания S_{II} могут быть распределены только на некоторые машины, т.е. используем ограничения допустимости для машин с так называемой иерархической топологией сервера или уровнем обслуживания [71].

Назовем этот алгоритм *Допустимое принятие*. Аналогичный подход также использовался в контексте планирования параллельных работ в ГРИД [17].

Теорема 5.10. *Допустимое принятие имеет конкурентный фактор* $\frac{1-\frac{1}{f_{II}}}{1+\frac{1-\frac{1}{f_{II}}}{1-\frac{1}{f_I}}-\frac{u_{II}}{u_I}}$ *для MSL-PM с уровнями сервисов* S_I *и* S_{II}.

Доказательство. Работы с уровнем обслуживания S_{II} могут быть распределены только на k выбранных машин, получая в соответствии с теоремой 2.5

$$c_V(E_{MSL-PM}) \geq \frac{k}{m} \cdot \left(1 - \frac{1}{f_{II}}\right), \tag{5.13}$$

В наихудшем случае большое число работ с уровнем сервиса S_I поступает сразу после работ с уровнем сервиса S_{II}. Работы с уровнем обслуживания S_I назначаются только на свой эксклюзивный набор машин $m-k$, в то время как на других k машинах их выполнение исключается за счет заданий с уровнем обслуживания S_{II}.

Используя теорему 5.5 для $m-k$ машин и теорему 5.7 для остальных k машин, получаем следующие условия

$$c_V(E_{MSL-PM}) \geq \frac{(k \cdot u_{II} + (m-k) \cdot u_I) \cdot (f_I - 1)}{m \cdot f_I \cdot u_I} \tag{5.14}$$

Для

$$k = \frac{m}{1 + \frac{1 - \frac{1}{f_{II}}}{1 - \frac{1}{f_I}} - \frac{u_{II}}{u_I}}$$

и достаточно большого m, так что k – целое число, правая сторона неравенств 5.13 и 5.14 дают один и тот же результат.

$$c_V(E_{MSL-PM}) \geq \frac{1 - \frac{1}{f_{II}}}{1 + \frac{1 - \frac{1}{f_{II}}}{1 - \frac{1}{f_I}} - \frac{u_{II}}{u_i}}.$$

Опять же, можем использовать расширение примера 5.8, чтобы показать, что конкурентный фактор теоремы 5.10 является достижимым для допустимого

принятия.

Конкурентный фактор допустимого принятия вместе с подходящим выбором k, (см. теорему 5.10) всегда больше, чем конкурентный фактор обычного жадного принятия, см. теоремы 5.09, т. к. неравенство

$$\frac{u_{II}}{u_I} < 1 < \frac{1-\frac{1}{f_{II}}}{1-\frac{1}{f_I}}$$

всегда соблюдается.

В следующем примере используем значения примера 5.10.

Пример 5.13. Рассмотрим сценарий с двумя уровнями обслуживания S_I и S_{II}, и $u_I = 10$, $u_{II} = 1$, $f_I = 2$, $f_{II} = 6$, и $m = 77$. В соответствии с теоремой 5.10 получаем $\frac{77}{1+\frac{1-\frac{1}{6}}{1-\frac{1}{2}}+\frac{1}{10}} = 30$. Следовательно допустимое принятие достигает конкурентного фактора $\frac{30}{77} \cdot \left(1-\frac{1}{6}\right) = \frac{25}{77}$.

Этот конкурентный фактор лучше, чем конкурентный фактор $\frac{1}{5}$ ограниченного принятия, применяемый ко всем машинам, см. пример 5.10.

5.8 Выводы по пятой главе

Основываясь на моделях расписаний реального времени, введена простая модель для назначения и планирования работ на основе уровней обслуживания.

Рассмотрены сценарии с одной или несколькими одинаковыми машинами. Для этих сценариев получены верхние пределы конкурентных факторов и простые алгоритмы, а также оценка их производительности с помощью конкурентного анализа.

Особенно интересно, что наиболее реалистичный сценарий с двумя уровнями обслуживания и параллельно работающими идентичными машинами

имеет лучший конкурентный фактор, чем сценарий с использованием только одной машины.

При решении многих других проблем планирования в режиме реального времени переход от модели одной машины к модели параллельно работающих идентичных машин приводит к худшему конкурентному фактору. Таким образом, этот результат показывает преимущество параллелизма, которое соответствует аналогичным утверждениям на практике.

Глава 6. ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ IAAS С УРОВНЯМИ ОБСЛУЖИВАНИЯ

В этой главе предлагается модификация модели облака IaaS, предложенной в главе 5. Чтобы показать практичность и конкурентоспособность алгоритмов, проведено комплексное исследование их производительности с помощью моделирования. Для оценки практичности алгоритмов планирования используются рабочие нагрузки, основанные на реальных производственных трассах гетерогенных HPC-систем.

Рассматривается проблема онлайн-планирования. Работы поступают одна за другой, и после поступления новой работы планировщик должен решить, отклонить ли ему эту поступившую работу или запланировать ее на одной из машин. Проблема является онлайновой, поскольку планировщик должен решать ее без информации о следующих работах.

Для этой задачи производительность алгоритмов оценивается набором метрик, включающих конкурентный фактор и число принятых работ.

В начале исследуются два жадных алгоритма SSL-SM и SSL-PM с одним уровнем обслуживания.

SSL-SM принимает каждую новую работу для одной машины, если эта работа и все ранее принятые работы могут быть выполнены в срок.

SSL-PM принимает работу с учетом всех доступных процессоров на параллельных машинах. Ключевые свойства SL должны быть соблюдены, чтобы обеспечить преимущества для реальных установок. Поскольку SL часто рассматривается как преемник парадигмы реального времени, ориентированной на обслуживание со сроками, начнем с простой модели с одним уровнем обслуживания на одном компьютере и расширим ее до множественных SL на нескольких компьютерах.

В дальнейшем рассматриваются один уровень обслуживания, четыре уровня обслуживания и более общий случай с множественным уровнем обслуживания и параллельными машинами (MSL-PM).

6.1 Модель работы

Пусть $SL = [SL^1, SL^2, \dots, SL^l, \dots, SL^k]$ – это набор уровней обслуживания, предлагаемых SLA. Для данного SL^l работа J_j требует производительность s_j^l, которая гарантируется предоставлением соответствующей виртуальной машины VM, и взимается стоимость u_j^l за единицу времени выполнения в зависимости от требуемой срочности. Эта срочность выражается через слак-фактор $f_j^l \geq 1$. $u_{max} = max\{ u_j^l \}$ обозначает максимальную стоимость для всех $l = 1..k$ и $j = 1..n$. Общее число работ, посланных в систему, равно n_r.

Каждая работа J_j описывается кортежем (r_j, w_j, d_j, SL_j^l), содержащим дату выпуска r_j, объем работы w_j, описывающий вычислительную нагрузку, которая должна быть выполнена до требуемого времени ответа, директивный срок выполнения d_j и уровень обслуживания $SL_j^l \in SL$.

Пусть $p_j = w_j / s_j^l$ – это гарантированное время, которое система потратит на обработку работы до окончания срока, согласно уровню обслуживания SL_j^l.

Пусть d_j – самое позднее время, за которое система должна выполнить работу J_j в случае ее принятия. Это значение вычисляется при принятии работы как $d_j = r_j + f_j^l \cdot p_j$. Максимальный срок выполнения работы равен $d_{max} = max_j\{d_j\}$. Когда работа поступает на исполнение, становятся известны ее характеристики.

Доход, который система получит за выполнение работы J_j, рассчитывается как $u_j^l \cdot p_j$. Как только работа поступила, планировщик должен решить до поступления других работ, принимается ли эта работа или нет.

Для того чтобы принять работу J_j, планировщик должен убедиться, что какая-то машина в системе способна выполнить ее до истечения срока. В случае принятия, позднее поданные работы не могут привести к тому, что работа J_j не

будет выполнена до окончания срока.

Как только работа принята, планировщик использует некоторое правило для составления плана выполнения. Множество принятых работ $J = [J_1, J_2, \ldots, J_n]$ является подмножеством поступивших работ, где $n \leq n_r$ – число принятых работ.

6.2 Модель машины

Рассмотрим набор из m гетерогенных машин $M = [M_1, M_2, \ldots, M_m]$. Каждая машина M_i описывается кортежем (s_i, eff_i), указывающим ее относительную скорость обработки s_i и ее энергоэффективность eff_i.

В момент времени t только подмножество всех машин может принять работу. Пусть $M^a(t) = [M_1, M_2, \ldots, M_{m^a}]$ – такое множество допустимых машин. Это множество определяется для каждой работы как подмножество доступных машин, которые могут выполнить эту работу без нарушения сроков и могут гарантировать вычислительную мощность s_j^l для обработки. Машины, скорость обработки которых меньше скорости, гарантированной SLA, не могут принять работу.

Значение s_i консервативно выбрано таким образом, что ускорение всех приложений превышает s_i. Таким образом, пользователи получают одинаковые гарантии, какие бы процессоры ни использовались. Сроки рассчитываются на основе уровня обслуживания и не могут быть изменены, а гарантированное время обработки не нарушается при более медленной обработке. C_{max} обозначает время выполнения всех работ.

6.3 Модель потребления энергии

В модели предполагается, что энергопотребление $P_i(t)$ машины M_i в момент времени t состоит из постоянной части P^{idle}, которая обозначает мощность, потребляемую машиной M_i в состоянии простоя, и переменной части P^{work}, которая зависит от рабочей нагрузки: $P_i(t) = o_i(t)\left(P^{idle} + w_i(t)P^{work}\right)$, где $o_i\ (t) = 1$, если машина включена в момент времени t, иначе $o_i(t) = 0$, и $w_i(t) = 1$, если машина занята, иначе $w_i(t) = 0$. Общая мощность, потребляемая системой, представляет собой сумму мощностей, потребляемых во время работы [174]: $E^{op} = \int_{t=0}^{C_{max}\int} P^{op}(t)dt$, причем $P^{op}(t) = \sum_{i=1}^{m} P_i(t) = \sum_{i=1}^{m} o_i(t) \cdot (P^{iddle} + w_i(t)P^{work})$.

6.4 Критерии оптимизации

Для оценки производительности системы используются метрики, которые полезны для систем с уровнями обслуживания, где традиционные критерии, такие как время выполнения, становятся неактуальными. Для такого рода систем метрики должны позволять провайдеру измерять производительность системы в терминах, помогающих установить маржу полезности, а также удовлетворенность пользователя услугой.

Рассмотрим два критерия: максимизацию дохода поставщика услуг и минимизацию энергопотребления E^{op}. Доход определяется как $V = \sum_{j=1}^{n}\left(u_j^l \cdot p_j\right)$.

В такой постановке необходимо обеспечить выгоду для поставщика услуг. Чтобы показать, как доход, генерируемый алгоритмом, приближается к значению, получаемому при оптимальном доходе $V(A)^*$, используем конкурентный фактор ρ, который определяется как:

$$\rho = \frac{\sum_{j=1}^{n}\left(u_j^l \cdot p_j\right)}{V(A)^*} \leq 1,$$

где оптимальный доход $V(A)^*$ аппроксимируется верхней границей $\hat{V}(A)^* = u_{max} \cdot min\left(\sum_{j=1}^{n_r} p_j\,, d_{max} \cdot m\right)$.

Чтобы получить верхнюю границу дохода, рассмотрим два возможных случая. Максимальный доход может быть получен, если все поступившие работы будут обработаны, или если принятые работы будут обработаны на всех машинах до директивного срока без простоя. В обоих случаях работа имеет максимальную цену за единицу времени.

Первый член $\hat{V}(A)^*$ – это сумма времени обработки всех полученных работ, умноженная на максимальную цену за единицу выполнения всех доступных SL. Второй член – это максимальный срок выполнения всех полученных работ, умноженный на максимальную цену за единицу выполнения и число машин в системе. Благодаря предложенной в диссертации политике контроля допуска, система не выполняет работы, если их сроки не могут быть достигнуты. Поэтому второй член также является верхней границей общего времени обработки системы.

Оптимальный доход больше или равен верхней границе: $V(A)^* \geq \hat{V}(A)^*$, а нижняя граница для коэффициента конкурентоспособности $\hat{\rho} \leq \rho$ получается с помощью $\hat{V}(A)^*$.

6.5 Методы планирования

В этом разделе описываются предлагаемые методы планирования с учетом SL и энергопотребления. Используется двухуровневый подход к планированию, как показано на рисунке 6.1 [17, 56]. На верхнем уровне система проверяет, может ли работа быть принята или нет, используя политику принятия Greedy. Если

работа принята, то система выбирает машину из набора допустимых машин для ее выполнения на нижнем уровне.

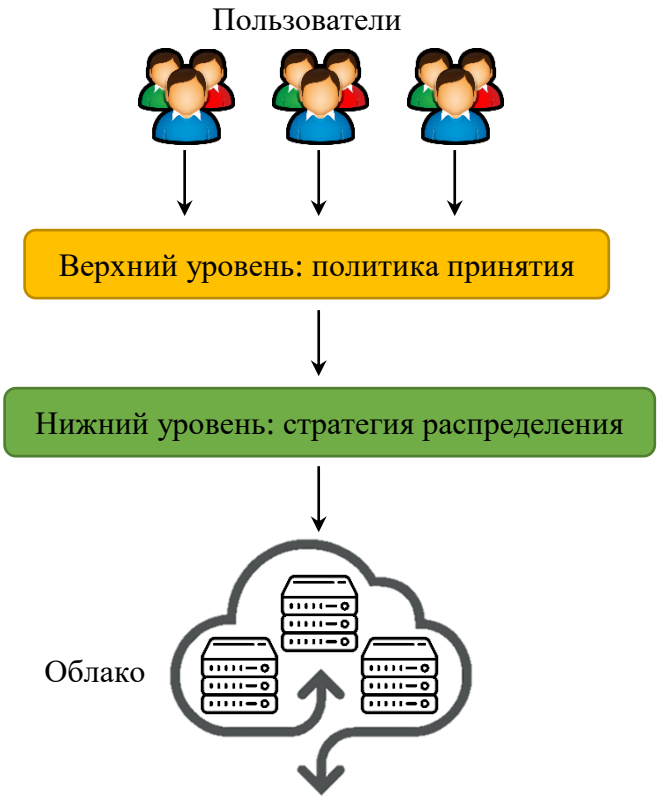


Рисунок 6.1 – Двухуровневый подход к планированию с использованием политики принятия (верхний уровень) и стратегий распределения (нижний уровень)

6.5.1 Политика принятия на верхнем уровне

Используется жадная политика приемки на верхнем уровне. Она основана на алгоритме EDD, который отдает приоритет работам в соответствии с их директивными сроками. Когда в систему поступает работа J_j, чтобы определить, принять ее или отклонить, система ищет набор машин, способных выполнить работу J_j до истечения срока, гарантируя, что ни одна работа в машине не

опоздает к сроку. Если набор доступных машин не пуст $\left(\left|M^a(r_j)\right| \geq 1\right)$, то работа J_j принимается, в противном случае она отклоняется. На этом первый этап планирования завершен.

На нижнем уровне используется алгоритм Preemptive EDD с прерываниями для каждой машины отдельно. Этот алгоритм прост в применении и дает оптимальное решение для проблемы 1 | prmp, r_j, online $| L_{max}$. В общем случае задержка L_j работы J_j определяется как $max(c_j - d_j, \cdot 0)$. Тогда получаем $L_{max} = max\{ L_j\}$.

Для всех планов в наших задачах L_{max} должно соблюдаться, т. к. ни одна из работ не может опоздать. Более того, алгоритм Preemptive EDD создает расписание без задержек (жадное планирование) и поэтому не откладывает использование ресурсов на будущее, когда они могут понадобиться еще неизвестным работам. С помощью Preemptive EDD проверяется, что все уже принятые работы, срок выполнения которых больше, чем срок выполнения поступающей работы, будут завершены до истечения срока их выполнения.

6.5.2 Стратегии распределения работ нижнего уровня

Машина для распределения работ может быть определена с учетом различных критериев. В данной главе рассматриваются восемь стратегий распределения (см. таблица 6.1). Они характеризуются типом и объемом информации, используемой для принятия решения о распределении.

Выделяется два уровня доступной информации. На уровне 1 предполагается, что время выполнения работы, скорость машин и политика приема известны. На уровне 2, кроме того, известна энергоэффективность машины и энергия, потребляемая при выполнении работы.

В таблице 6.1 приведены подробные сведения о стратегиях распределения, используемых в данной работе. Предложенные стратегии можно разделить на три

группы: 1) без знаний о системе, без информации о работах и ресурсах [16, 9, 56]; 2) с учетом энергопотребления, с информацией об энергопотреблении; 3) с информацией о скорости машин.

Таблица 6.1 – Стратегии распределения работ

Тип		Стратегия	Уровень	Описание
Knowledge	Free	Rand	1	Распределяет работу j на подходящую машину, выбранную случайным образом с помощью равномерного распределения в диапазоне $[1..m]$.
		FFit	1	Распределяет работу j на первую доступную машину, способную его выполнить.
		MLp	1	Распределяет работу j на машину с наименьшей нагрузкой в момент времени r_j: $min\{n_i\}$.
Energy	aware	Max-eff	2	Распределяет работу j на машину с наибольшей энергоэффективностью $max\{eff_i\}$.
		Min-e	2	Распределяет работу j на машину с минимальным полным энергопотреблением в момент времени r_j : $min\{\sum_{t=1}^{r_j} P_i^{op}(t)\}$.
		MCT-eff	2	Распределяет работу j на машину с наилучшим соотношением между временем выполнения и энергоэффективностью $min\{C_{max}^i/eff_i\}$, где $c_{max}^i = max\{c_k^i\}$ и c_k^i – это время выполнения и время завершения работы k на машине i, соответственно.
Speed	aware	Max-seff	2	Распределяет работу j на машину с $max\{s_i * eff_i\}$.
		Max-s	2	Распределяет работу j на самую быструю машину: $max\{s_i\}$.

6.5.3 Алгоритмы низкоуровневого исполнения

В экспериментах используются алгоритмы SSL-SM, SSL-PM, MSL-SM и MSL-PM. Работы, которые были приняты, но еще не завершены, ставятся в очередь. Работы упорядочиваются по убыванию директивных сроков. Для их выполнения работы берутся из головы очереди. Когда новая работа освобождается, она помещается в очередь в соответствии с ее сроком выполнения.

EDD – это оптимальный алгоритм для минимизации опозданий в одно машинной системе. В нашем случае это соответствует минимизации числа отклоненных работ.

6.5.4 Рабочая нагрузка

Чтобы оценить производительность планирования, проведем серию экспериментов с использованием работ HPC, полученных из архивов PWA и GWA. (таблица 6.2). Подробную информацию о характеристиках можно найти в [184] и [129].

Таблица 6.2 – Трассы работ

Система	Проц.	P^{Idle}	$P^{Working}$	Энергоэфф. MFLOPS/W	Скорость GFlOPS
DAS2—University of Amsterdam	64	17.8	35.35	1.36	126
DAS2—Delft University of Technology	64	17.8	35.35	1.36	126
DAS2—Utrecht University	64	17.8	35.35	1.36	126
DAS2—Leiden University	64	17.8	35.35	1.36	126
KTH—Swedish Royal Institute of Technology	100	17.8	26	1.75	18.6
DAS2—Vrije University Amsterdam	144	17.8	35.35	1.32	230
HPC2N—HPC Center North, Sweden	240	58.9	66	0.89	481
CTC—Cornell Theory Center	430	17.8	26	1.64	88.4
LANL—Los Alamos National Lab	1024	24.7	31	1.45	65.4

Система	Log	#работ	#пользов.
DAS2—University of Amsterdam DAS2—Delft University of Technology	Gwa-t-1-anon_jobs-reduced.swf	1124772	333

DAS2—Utrecht University			
DAS2—Leiden University			
KTH—Swedish Royal Institute of Technology			
DAS2—Vrije University Amsterdam	KTH-SP2-1996-2.swf	28489	204
HPC2N—HPC Center North, Sweden	HPC2N-2002-1.1-cln.swf	527371	256
CTC—Cornell Theory Center	CTC-SP2-1996-2.1-cln.swf	79302	679
LANL—Los Alamos National Lab	LANL-CM5-1994-3.1-cln.swf	201387	211

Эти трассы являются логами реальных параллельных компьютерных систем и дают хорошее представление о том, как предлагаемые схемы будут работать с реальными пользователями. Хорошо известно, что в реальных логах преобладают низкопараллельные работы. Несмотря на то, что некоторые работы, представленные в логах, требуют нескольких процессоров, мы считаем, что машины имеют достаточную мощность для их обработки.

Поскольку предполагается, что облака IaaS являются перспективной альтернативой вычислительным центрам, можно ожидать, что рабочая нагрузка, передаваемая в облака, будет иметь схожие характеристики с теми, которые передаются в реальные параллельные и грид-системы. В нашем логе рассмотрено девять трасс из следующих центров: DAS2 (Университет Амстердама), DAS2 (Университет Делфта), DAS2 (Университет Утрехта), DAS2 (Университет Лейдена), KTH, DAS2 (Университет Врие), HPC2N, CTC и LANL.

Хорошо известно, что распределение работ неравномерно по времени и зависит от времени суток и дня недели. Более того, каждый отдельный лог показывает разное распределение. Кроме того, они записываются в разных часовых поясах. Поэтому необходима нормализация используемых рабочих нагрузок путем смещения нагрузок на определенный временной интервал, чтобы представить более реалистичную ситуацию. Рабочие нагрузки преобразуются таким образом, чтобы все трассы начинались в один и тот же будний день и в одно и то же время суток. Для этого удалены все работы до первого понедельника

в полночь. Обратите внимание, что выравнивание связано с местным временем, поэтому разница во времени, соответствующая исходным часовым поясам, сохраняется.

Необходимо рассматривать нормализацию временных зон, нормализацию профилированных временных интервалов и фильтрацию недействительных работ. Для удаления определенных работ применяется несколько фильтров: время отправки < 0; время выполнения ≤ 0; число выделенных процессоров ≤ 0; запрошенное время ≤ 0; идентификатор пользователя ≤ 0; статус $= 0, 4, 5$ ($0 -$ работа не выполнена; $4 -$ частичное выполнение, работа не выполнена; $5 -$ работа была отменена, либо до начала, либо во время выполнения).

Чтобы получить достоверные статистические данные, для каждого SL моделируется 30 экспериментов с интервалом в одну неделю. Мы рассчитываем сроки выполнения работ на основе реального времени обработки работ.

6.6. Методология, использованная для анализа

6.6.1 Деградация производительности

Для того чтобы обеспечить эффективное ориентирование при выборе наилучшей стратегии, проведем совместный анализ двух метрик в соответствии с методологией деградации производительности (относительной ошибки) (Глава 3). При анализе средних значений необходимо исключить влияние небольшой части данных с большим отклонением на процесс сравнения и на интерпретацию данных. Для этого используем профили эффективности (производительности) стратегий.

6.6.2 Профиль эффективности

Профиль эффективности $\delta(\tau)$ – это неубывающая, кусочно-постоянная функция, которая представляет собой вероятность того, что отношение γ находится в пределах коэффициента τ от наилучшего отношения [79]. Функция $\delta(\tau)$ является кумулятивной функцией распределения. Стратегии с большой вероятностью $\delta(\tau)$ при малом τ будут предпочтительнее.

6.6.3 Би-критериальный анализ

Многокритериальная оптимизация обычно находит набор решений, известный как оптимальное множество Парето [87]. Одно решение может являться очень хорошим в отношении потребления энергии, в то время как другое решение может быть очень хорошим в отношении дохода.

Цель – выбрать наиболее адекватное решение и получить набор компромиссных решений, которые представляют собой хорошее приближение к фронту Парето. Двумя важными характеристиками хорошей многокритериальной методики являются сходимость к фронту Парето и разнообразие для максимально полной выборки фронта. Решение является оптимальным по Парето, если никакое другое решение не улучшает его с точки зрения всех целевых функций. Любое решение, не входящее во фронт, может считаться менее качественным, чем те, которые в него входят.

Выбор между решениями, входящими во фронт Парето, зависит от предпочтений системы. Если один критерий считается более важным, чем другой, то предпочтение отдается тем решениям, которые близки к оптимальным по предпочтительному критерию, даже если значения вторичного критерия не являются одними из лучших.

Часто результаты решения многоцелевых задач сравниваются путем визуального наблюдения пространства решений. Более формальный и статистический подход использует метрику покрытия (доминирования)

множества (англ. “set coverage”) [223]: даны два множества решений A и B, метрика $SC(A,B)$ вычисляет долю решений в B, которые слабо доминируют над решениями A:

$$SC(A,B) = \frac{|\{b \in B; |\exists a \in A: a \leq b\}|}{|B|}$$

Метрическое значение $SC(A,B) = 1$ означает, что все решения B доминируют A, тогда как $SC(A,B) = 0$ означает, что ни один член B не доминируется A. Таким образом, чем больше значение $SC(A,B)$, тем лучше фронт Парето A по отношению к B. Поскольку оператор доминирования не является симметричным, $SC(A,B)$ не обязательно равен $SC(A,B)$, и для того, чтобы понять, сколько решений A покрывается B и наоборот, необходимо вычислить как $SC(A,B)$, так и $SC(B,A)$.

6.7 Анализ одной машины

Для первого набора экспериментов со схемой одномашинной системы проведены эксперименты для 12 значений слак-фактора: 1, 2, 5, 10, 15, 20, 25, 50, 100, 200, 500 и 1000. Хотя не ожидается, что в реальном SL слак-фактор будет больше 50, большие значения важны для изучения ожидаемой производительности системы, когда слак-фактор стремится к бесконечности.

На рисунках 6.2-6.6 представлены результаты моделирования алгоритма SSL-SM. На них представлены процент отклоненных работ, общее время обработки принятых работ, среднее время ожидания, среднее число прерываний на одну работу и средний конкурентный фактор.

На рисунке 6.2 показан процент отклоненных работ для алгоритма SSL-SM. Видно, что число отклоненных работ уменьшается с увеличением слак-фактора.

Большие значения слак-фактора увеличивают гибкость принятия новых работ за счет задержки выполнения уже принятых. В случае, когда слак-фактор равен единице, система не может принимать новые работы до тех пор, пока не

завершится работа, находящаяся в процессе выполнения. Заметим, что процент отклоненных работ при слак-факторе, равном единице, несколько ниже, чем при значениях слак-фактора от 2 до 25. Однако это не означает, что данный фактор позволяет системе выполнять больше вычислительной работы (см. рисунок 6.2).

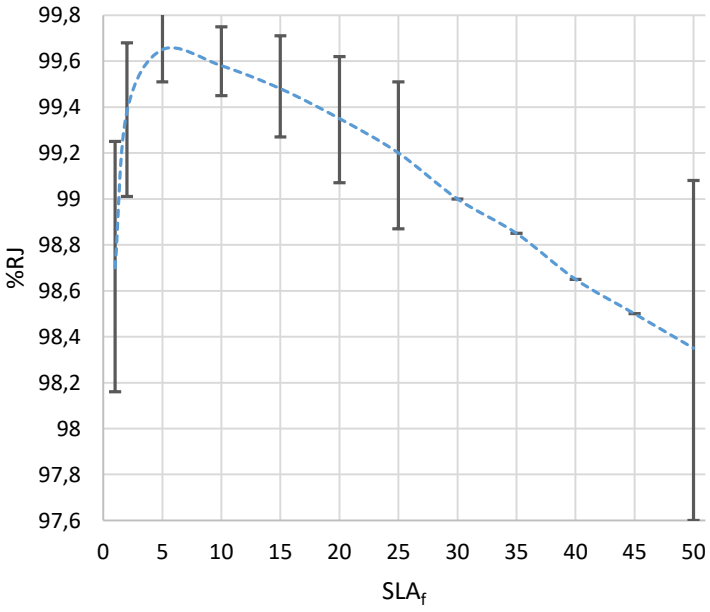


Рисунок 6.2 – Процент отклоненных работ для алгоритма SSL-SM.

На рисунке 6.3 показано общее время обработки принятых работ для заданных слак-факторов. Видно, что время обработки увеличивается по мере увеличения слак-фактора, что означает, что планировщик способен использовать увеличение гибкости работ.

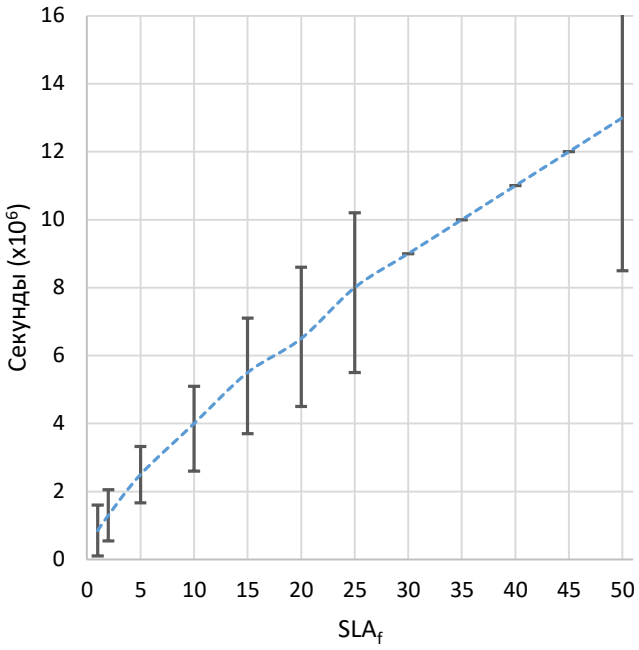


Рисунок 6.3 – Общее время обработки для алгоритма SSL-SM.

На рисунке 6.4 показано среднее время ожидания в зависимости от слак-фактора. Он показывает, что увеличение общего времени обработки приводит к увеличению времени ожидания.

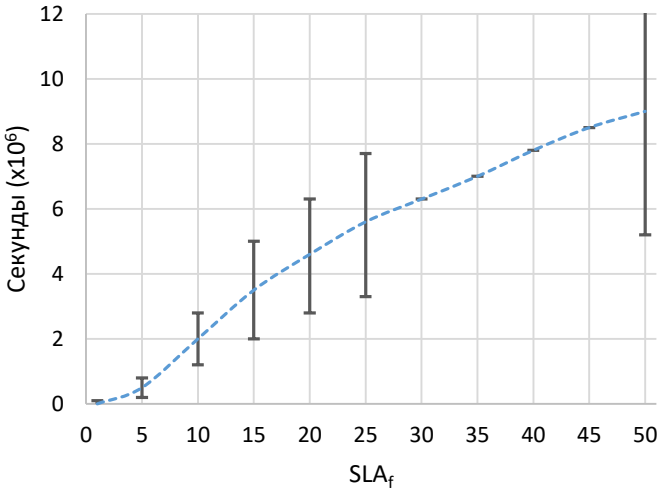


Рисунок 6.4 – Среднее время ожидания работ для алгоритма SSL-SM.

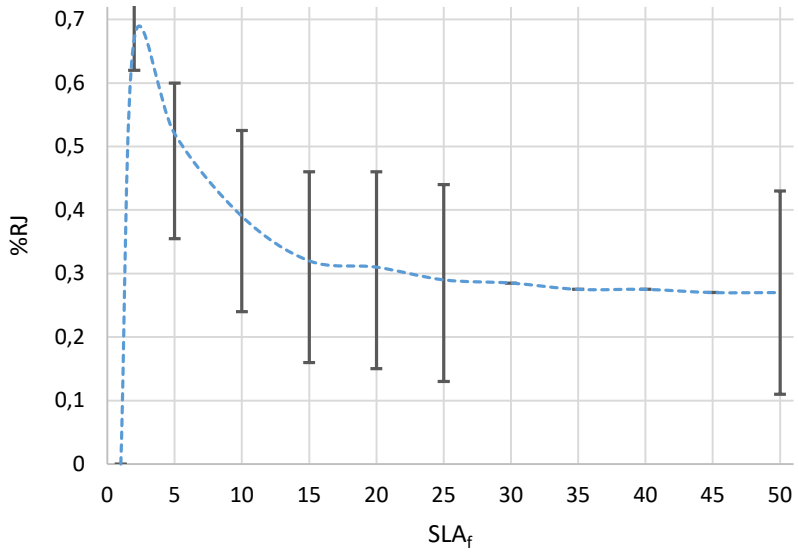


Рисунок 6.5 – Среднее число прерываний на работу для алгоритма SSL-SM

Среднее число прерываний на одну работу показано на рисунке 6.5. Видно, что при малых значениях слак-фактора число прерываний больше, чем при больших слак-факторах. Средние значения ниже одного прерывания на работу. Более того, если слак-фактор больше десяти, то число прерываний на одну работу стабильно между 0.2 и 0.3. Этот факт важен, т. к. поддержание низкого числа прерываний уменьшает накладные расходы системы.

На рисунке 6.6 показан средний конкурентный фактор. Он отражает цель поставщика инфраструктуры максимизировать свой общий доход.

.

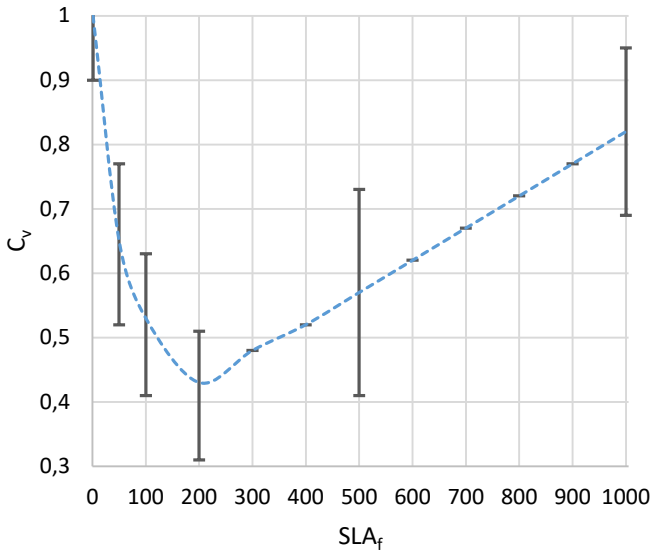


Рисунок 6.6 – Средний конкурентный фактор алгоритма SSL-SM.

Больший конкурентный фактор лучше, чем меньший. Когда слак-фактор равен единице, конкурентный фактор равен 0.85. При увеличении слак-фактора до 5 получаются лучшие конкурентные факторы. Когда слак-фактор равен 5, средний конкурентный фактор имеет максимальное значение 0.94. Пройдя эту точку, конкурентный фактор уменьшается до тех пор, пока слак-фактор не станет равным 200. Полагаем, что в этот момент сроки выполнения работ значительно превышают время их обработки. Если слак-фактор находится между 200 и 500, то конкурентный фактор снова увеличивается, т. к. максимальный срок приближается к сумме времени обработки.

Когда срок выполнения всех работ стремится к бесконечности, конкурентный фактор оптимален, как и ожидалось.

В реальном облачном сценарии слак-фактор может динамически регулироваться в ответ на изменения в конфигурации и/или рабочей нагрузке. Для этого можно проанализировать историческую рабочую нагрузку за определенный промежуток времени, чтобы определить соответствующий слак-фактор. Временной интервал для этой корректировки должен быть установлен в соответствии с динамическими характеристиками рабочей нагрузки и конфигурацией IaaS.

6.8 Анализ нескольких машин

В этом разделе представлены результаты моделирования алгоритма SSL-PM с двумя и тремя машинами. Представлены результаты и построены графики результатов SSL-SM для анализа изменения производительности системы в зависимости от числа машин.

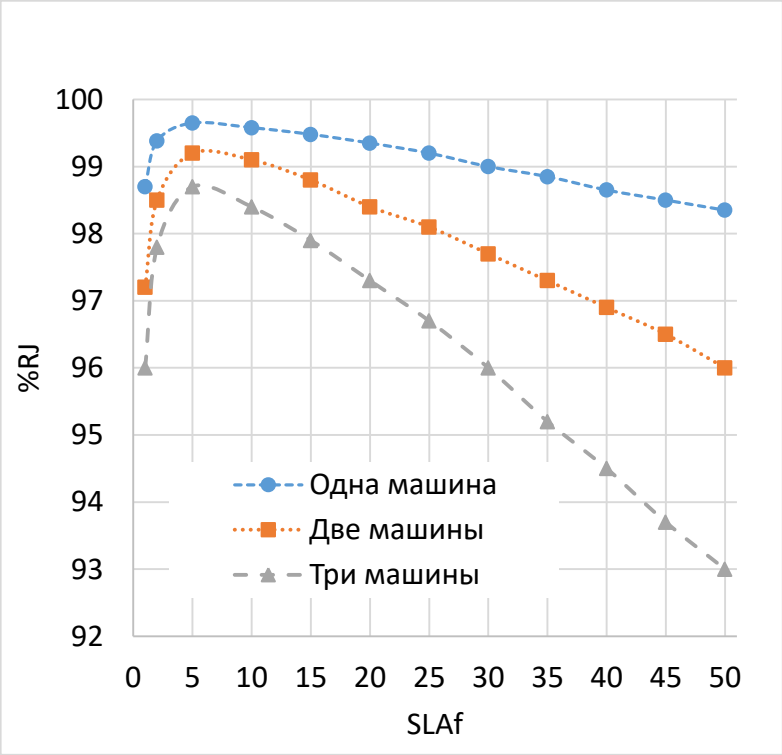


Рисунок 6.7 – Процент отклоненных работ для алгоритма SSL-PM.

На рисунке 6.7 представлен процент отклоненных работ. Видно, что увеличение числа машин оказывает ограниченное влияние на принятие работ, когда слак-фактор мал. Однако большие значения слак-фактор оказывают большее влияние на число принятых работ.

На рисунке 6.8 показано общее время обработки принятых работ. Оно увеличивается по мере добавления большего числа машин в систему. Однако удвоение и утроение мощности обработки не приводит к одинаковому увеличению времени обработки. Этот эффект хорошо виден при большом слак-

факторе. Можно сделать вывод, что увеличение мощности обработки будет более эффективным при меньших слак-факторах.

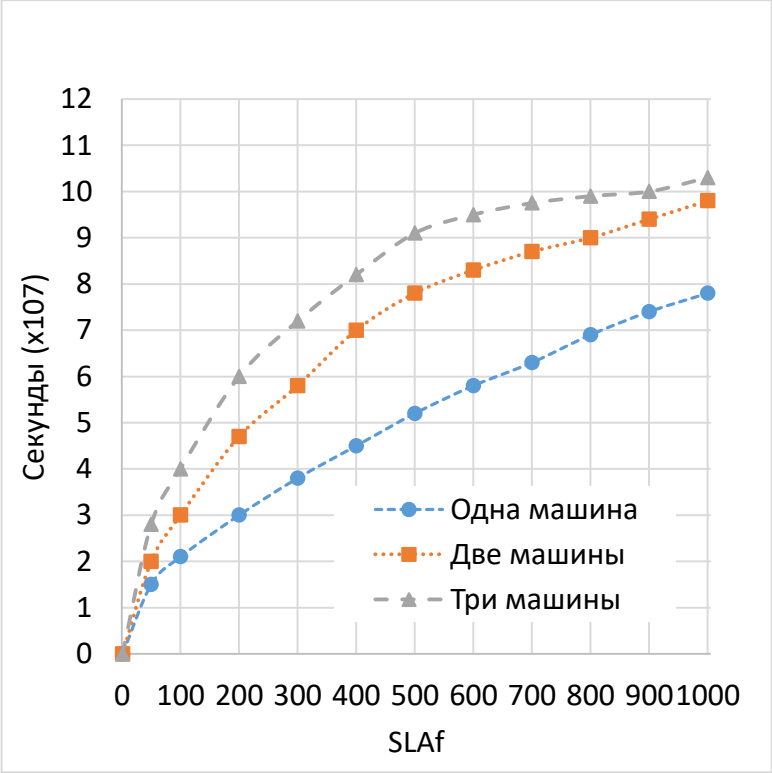


Рисунок 6.8 – Общее время обработки для алгоритма SSL-PM.

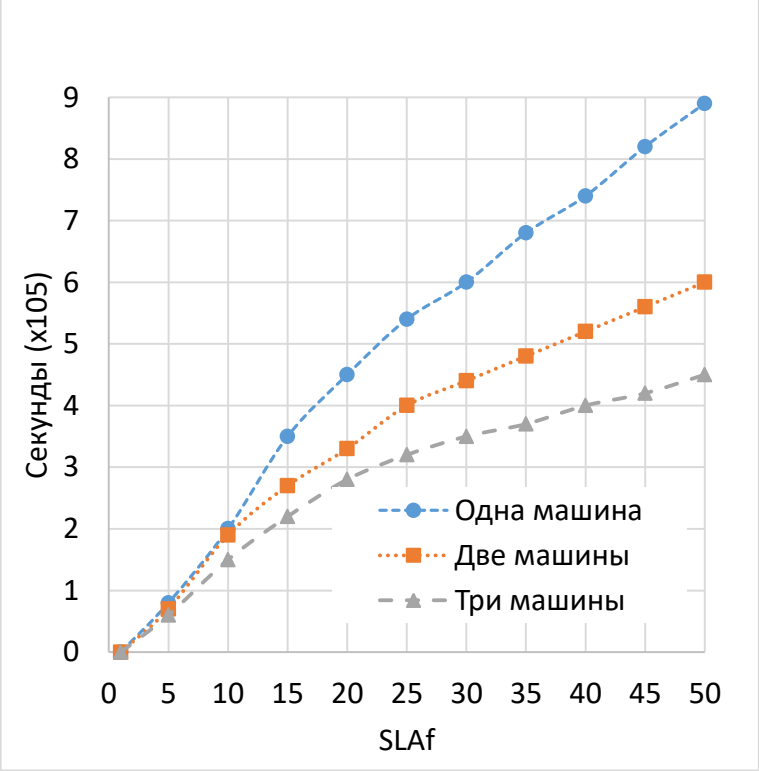


Рисунок 6.9 – Среднее время ожидания для алгоритма SSL-PM.

На рисунке 6.9 показано среднее время ожидания в зависимости от слак-фактора. Видно, что увеличение общего времени обработки при больших слак-факторах также приводит к увеличению времени ожидания. Кроме того, добавление большего числа машин в систему делает увеличение среднего времени ожидания менее значительным.

На рисунке 6.10 показано среднее число прерываний на одну работу. Видно, что увеличение числа машин увеличивает число прерываний.

Это увеличение незначительно и стабилизируется по мере увеличения слак-фактора. Число прерываний максимально при слак-факторе равном двум, для всех трех моделей.

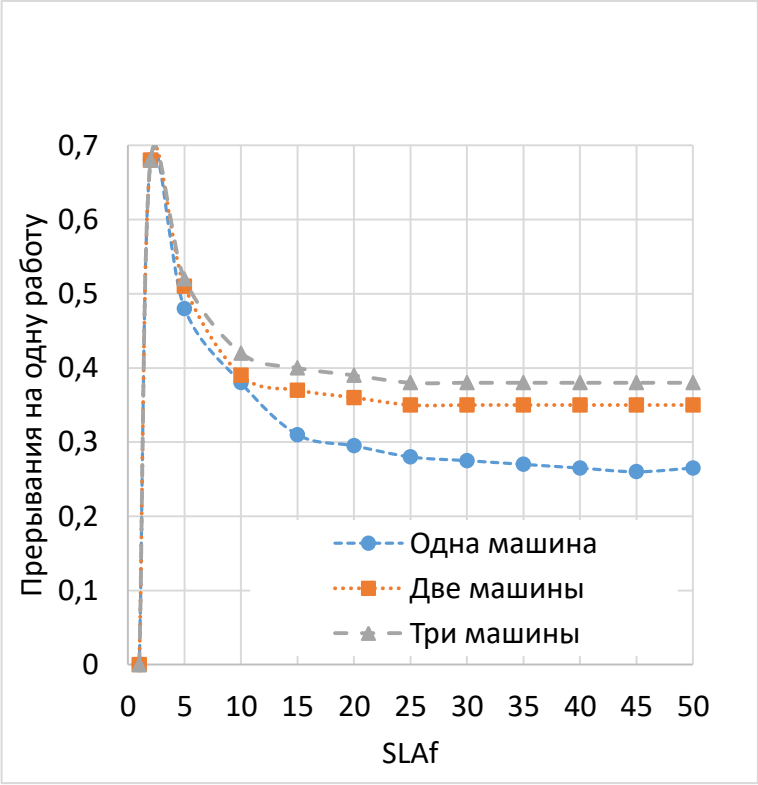


Рисунок 6.10 – Среднее число прерываний для алгоритма SSL-PM.

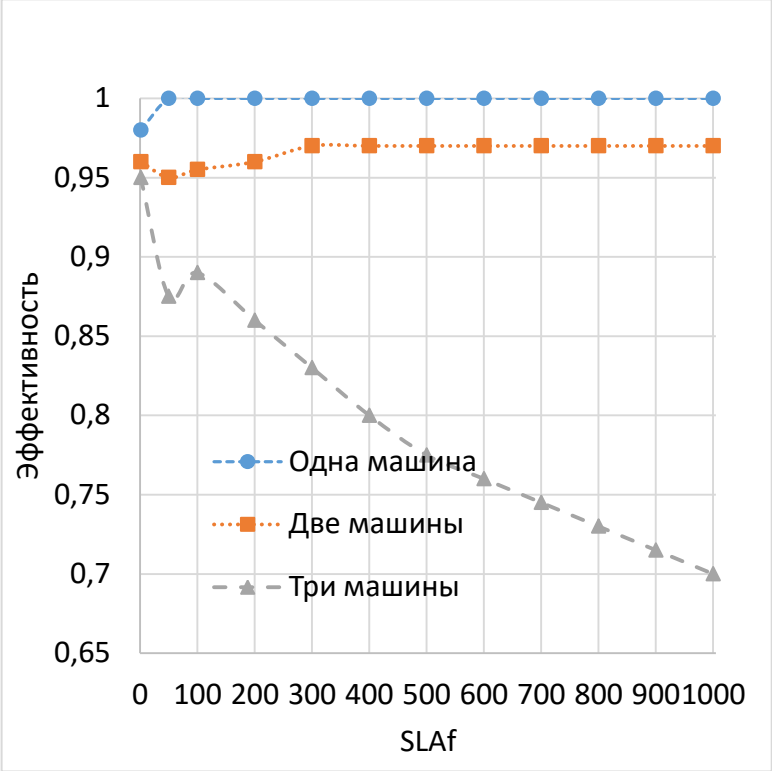


Рисунок 6.11 – Эффективность выполнения для алгоритма SSL-PM.

На рисунке 6.11 показана эффективность выполнения. Эта метрика показывает относительное количество полезной работы, которую система выполняет за промежуток времени между моментом получения первой работы и завершением последней.

Видно, что снижение эффективности, по крайней мере, при умеренных слак-факторах, в основном зависит от числа машин.

На рисунке 6.12 представлен конкурентный фактор, зависящий от слак-фактора. Видно, что для двух- и трех машинной конфигурации системы максимальный конкурентный фактор достигается при слак-факторе, равном двум. Как было отмечено выше, в случае одномашинной конфигурации наилучшие конкурентные факторы достигаются при слак-факторе пять и два. Также можно заметить, что при увеличении слак-фактора конкурентный фактор снижается. Это происходит до тех пор, пока слак-фактор не становится достаточно большим, чтобы создать значительную разницу между сроками выполнения работ и временем их обработки. Это хорошо видно, когда слак-фактор равен 200 для

конфигурации с одной машиной и 100 для двух и трех машин.

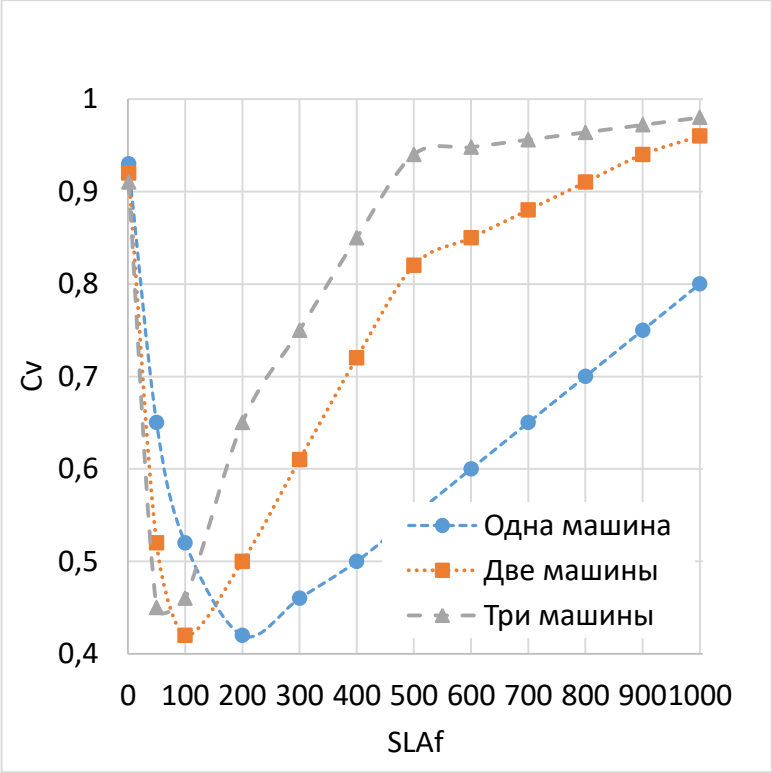


Рисунок 6.12 – Конкурентный фактор для алгоритма SSL-PM.

В случае конфигурации из двух и трех машин, при слак-факторе более 500 конкурентный фактор почти достиг оптимального значения.

6.9 Анализ затрат на выполнение

В сценарии IaaS облачные провайдеры предлагают компьютерные ресурсы клиентам на условиях оплаты по мере использования. Цена за единицу времени зависит от выбранных клиентом услуг. Эта плата зависит не только от цены, которую готов принять пользователь, но и от стоимости обслуживания инфраструктуры.

Чтобы оценить эту плату, предложена тарифная функция, которая зависит от слак-фактора. Для начала рассмотрим, случай, когда провайдеру необходимо

вернуть стоимость обслуживания от выполнения работ. Предполагается, что провайдер платит фиксированную ставку за использование/обслуживание ресурсов [70].

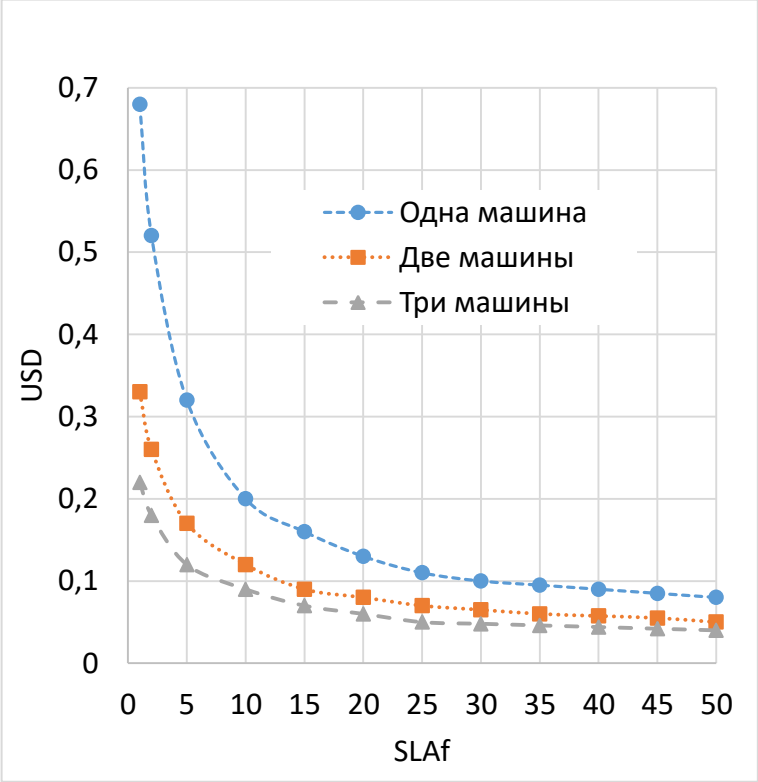


Рисунок 6.13 – Стоимость выполнения в час

Общие эксплуатационные расходы на обработку работ (co_t) могут быть рассчитаны по формуле:$\frac{\sum_{j=1}^{n_r} p_j}{m} \times u_u \times m$. А стоимость единицы времени co_u можно рассчитать как $co_u = \frac{co_t}{\sum_{j=1}^{n} p_j}$, где $\sum_{j=1}^{n_r} p_j$ – сумма времени обработки всех выпущенных работ, u_u – цена за единицу обслуживания, m - число машин, а $\sum_{j=1}^{n} p_j$ – сумма времени обработки, достигнутая алгоритмом. Исходим из того, что u_u равна 8,5 центам в час – это цена, которую Amazon EC2 взимает за небольшую вычислительную машину [91].

На рисунке 6.13 показана стоимость выполнения в час в зависимости от слак-фактора. Как видно, затраты системы на обработку работ с небольшим слак-фактором больше, чем на выполнение работ с меньшим слак-фактором. Более того, затраты больше, если используется меньше машин. Причина в том, что

система с меньшим числом машин и малым слак-фактором отклоняет большинство работ в интервале, поэтому их выполнение требует больших затрат. В то время как конфигурации, выполняющие больше работ, имеют меньшие затраты на единицу времени выполнения. Очевидная прибыль получается, если представленная стоимость за единицу времени увеличивается.

6.9 Алгоритмы с учетом энергопотребления

6.9.1 Сценарии

В этой главе расширяются предыдущие результаты для оценки гарантий QoS. Рассматриваются один уровень обслуживания, четыре уровня обслуживания и более общий вариант множественный уровень обслуживания – на множестве машин (MSL-PM) с пятью различными SL и до пяти уровней обслуживания.

Для всех сценариев определяются число машин и число SL, которые используются в экспериментальном анализе. Рассмотрим восемь размеров инфраструктуры с числом машин, равным степени 2 от 1 до 128. Это не точно соответствует всем машинам, на которых регистрируются рабочие нагрузки, и в некоторых случаях может вызвать артефакты в единичном прогоне. Для получения достоверных статистических значений моделируется 30 экспериментов по 7 дней.

Сценарии имеют следующие детали: семидневная рабочая нагрузка; жадная политика принятия на высшем уровне; гетерогенные машины; восемь размеров инфраструктуры, 20 SL; восемь эвристик распределения нижнего уровня, описанных в таблице 6.1.

В таблице 6.2 представлены скорость, энергоэффективность и энергопотребление машин, полученные из их спецификаций. Видно, что скорость находится в диапазоне [18.6, 481] GFLOP, энергоэффективность – в диапазоне

[0.89, 1.75] MFLOPS/W, энергопотребление – в диапазоне [17.8, 58.9] W для P^{idle} и в диапазоне [16, 66] W для P^{work}. Эти данные используются в экспериментах.

В таблице 6.3 представлены различные SL. SLi содержит i-й уровнь обслуживания. Каждый уровень обслуживания связан с ценой за единицу времени выполнения работы и со слак-фактором, который определяет максимальный промежуток времени для предоставления запрашиваемого объема вычислительных ресурсов.

Таблица 6.3 – Несколько уровней обслуживания и соответствующие слак-факторы

SLA	Слак-фактор
1	$SL^1 \rightarrow f_1 = 1$
2	$SL^1 \rightarrow f_1 = 1, SL^2 \rightarrow f_2 = 2$
3	$SL^1 \rightarrow f_1 = 1, SL^2 \rightarrow f_2 = 2, SL^3 \rightarrow f_3 = 3$
…	…
20	$SL^1 \rightarrow f_1 = 1, SL^2 \rightarrow f_2 = 2, \ldots, SL^{20} \rightarrow f_{20} = 20$

Сценарий 1: один уровень обслуживания. Представлены три исследования. В первом используется один SL для всех работ. При этом SL варьируется от 1 до 20, чтобы сравнить алгоритмы с различными SL и с наихудшей границей, найденной на основе теоретического анализа. Во втором, для более детального анализа, каждая работа может иметь один из четырех SL из набора $SL = [SL^1, \ldots, SL^4]$ со слак-фактором $f_1 = 1, \ldots, f_4 = 4$. И в третьем представлен более полный анализ, в котором SL варьируется от 1 до 5. Сначала рассматриваются деградации дохода и энергопотребления, для каждого SLA независимо, затем их средние значения и ранжирование.

На рисунке 6.14 представлен конкурентный фактор в однородной среде с вариацией слак-фактора f. Видно, что увеличение сроков для всех работ уменьшает общий доход, но увеличивает гибкость планировщика для построения расписаний, близких к оптимальному доходу. При $f = 20$ конкурентные

факторы близки к оптимальным. Это происходит, когда f становится достаточно большим, чтобы создать значительную разницу между сроками выполнения работ и временем их обработки. Для сценария с $m = 64$ конкурентный фактор увеличивается с $\rho = 0{,}8$ при $f = 1$ до $\rho = 1$ при $f = 5$.

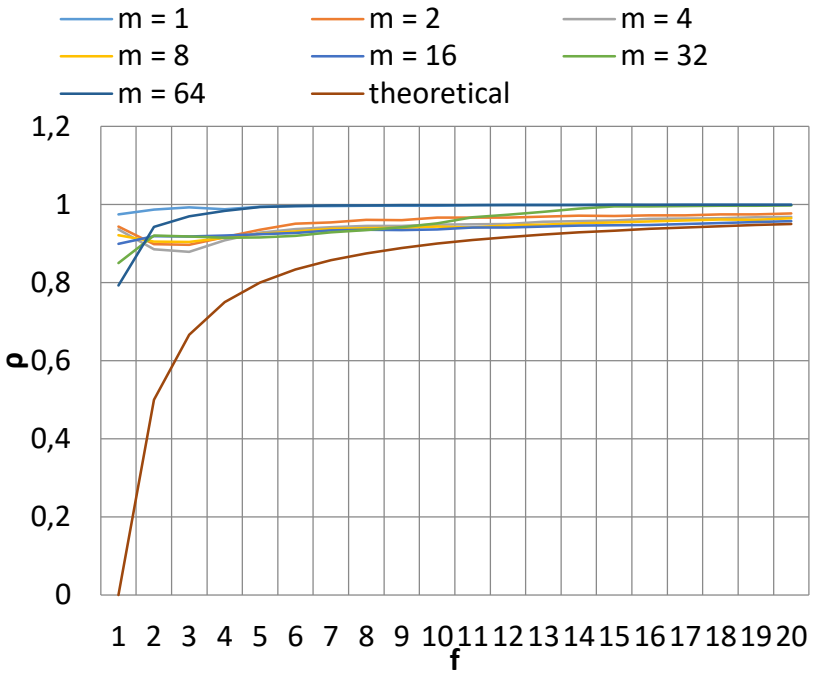


Рисунок 6.14 – Оценка производительности SSL-PM-hom

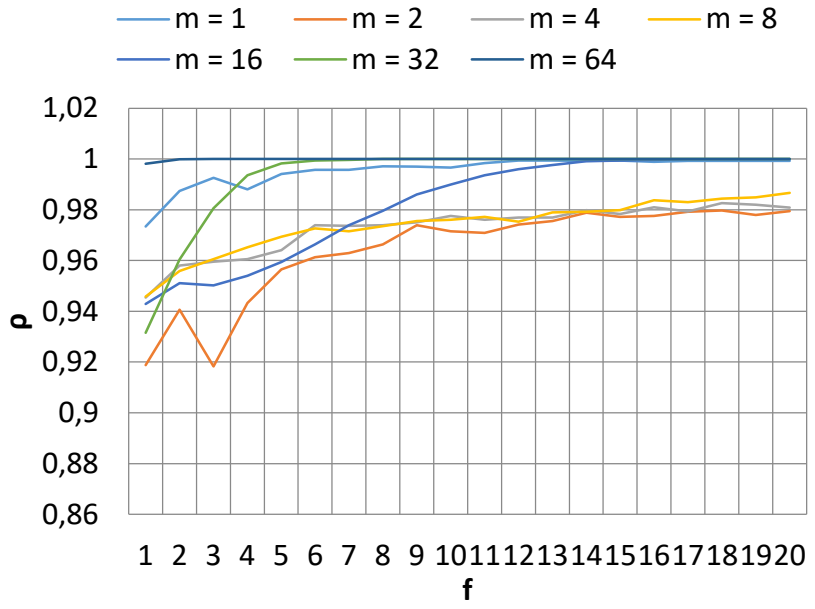


Рисунок 6.15 – Оценка производительности SSL-PM-het

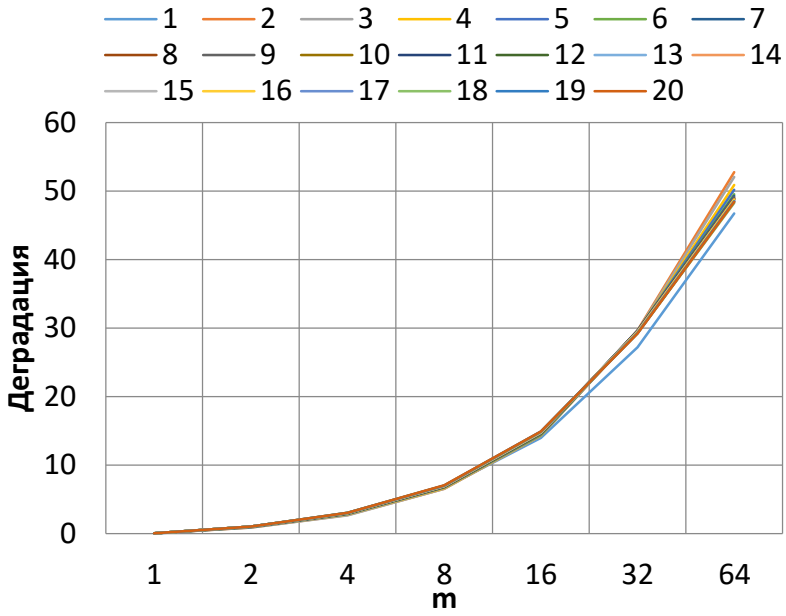


Рисунок 6.16 – Деградация энергопотребления. SSL-PM-hom

На рисунке 6.15 представлены ρ при изменении f в гетерогенной среде. Видна та же тенденция увеличения ρ с ростом f, однако с большим разбросом.

На рис. 6.16 и 6.17 показана деградация энергопотребления в однородной и неоднородной средах. Деградация увеличивается при росте f. Видно, что гетерогенность ухудшает производительность чуть больше, чем в два раза по энергопотреблению. В однородном сценарии стратегии работают практически одинаково, в то время как в гетерогенном сценарии производительность снижается с большим отклонением.

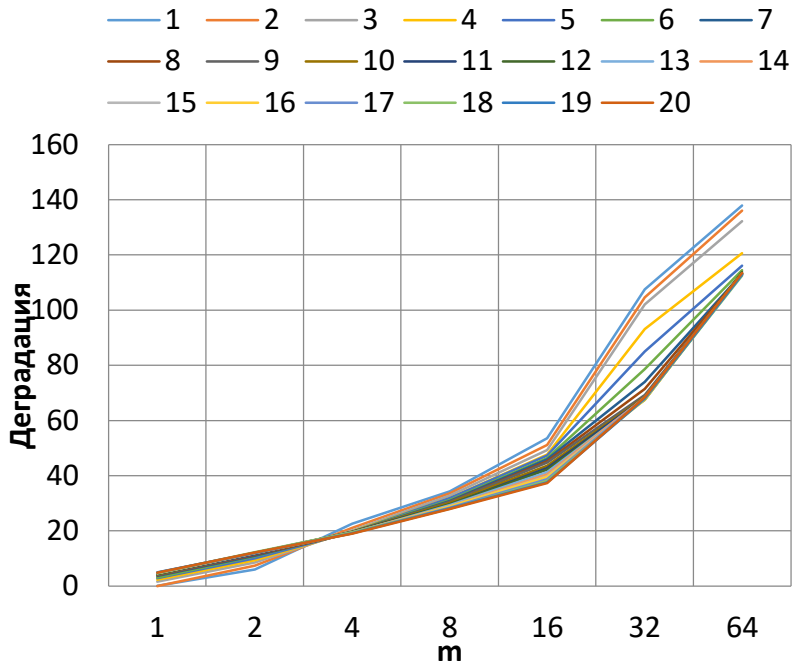


Рисунок 6.17 – Деградация энергопотребления. SSL-PM-het

Сценарий 2: четыре уровня обслуживания. Все пользователи используют SLA с четырьмя уровнями обслуживания с соответствующими слак-факторами $\boldsymbol{f_1 = 1, \ldots, f_4 = 4}$. Проведены четыре исследования: в первом анализируется деградация доходов; во втором изучается деградация энергопотребления; в третьем изучается средняя деградация производительности; и в четвертом представлен анализ профиля производительности.

6.9.2 Доход

В этом подразделе анализируется доход, полученный при использовании восьми стратегий распределения, изученных для восьми рассматриваемых инфраструктур.

На рисунках 6.18–6.20 показаны средний доход на машину, общий доход и деградация дохода, соответственно, при использовании четырех уровней обслуживания.

На рисунке 6.18 видно, что для заданных рабочих нагрузок лучший доход

от инфраструктуры на машину получается при использовании 4 машин. Когда число машин увеличивается, доход, генерируемый каждой машиной, снижается. Средний доход, генерируемый каждой машиной в сценарии с 4 машинами, примерно в 3 раза выше, чем в сценариях с 1, 64 и 128 машинами.

На рисунке 6.19 показано, что общий доход, генерируемый каждой стратегией, различен при использовании 8, 16 и 32 машин. При использовании 64 и 128 машин стратегии показывают схожие результаты независимо от того, какая машина используется для распределения, т. к. в этих случаях все работы принимаются, обеспечивая оптимальный доход. Например, в сценарии с $m = 16$, Max_eff , Max_s и Max_seff генерируют больше дохода, но разница с Min_e составляет менее 1%

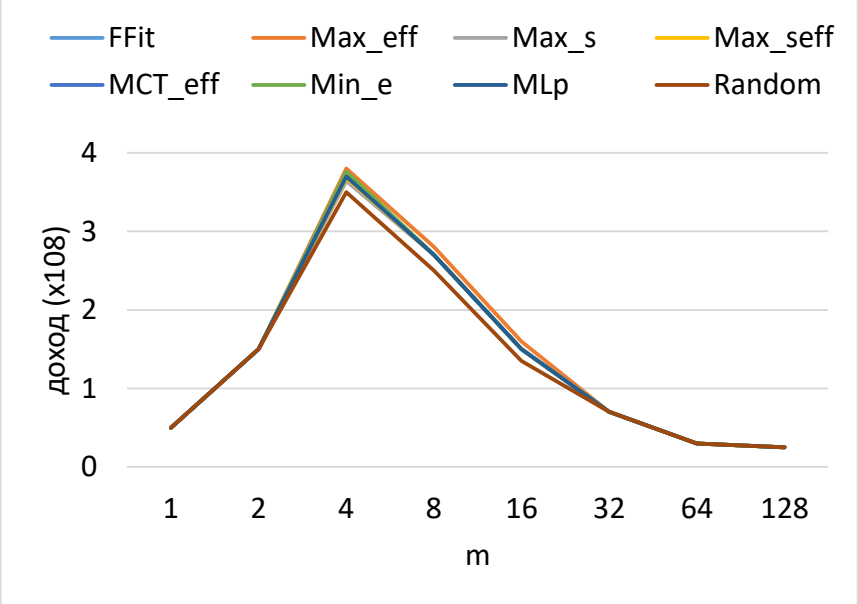


Рисунок 6.18 – Средний доход на машину при использовании SLA с 4 уровнями обслуживания

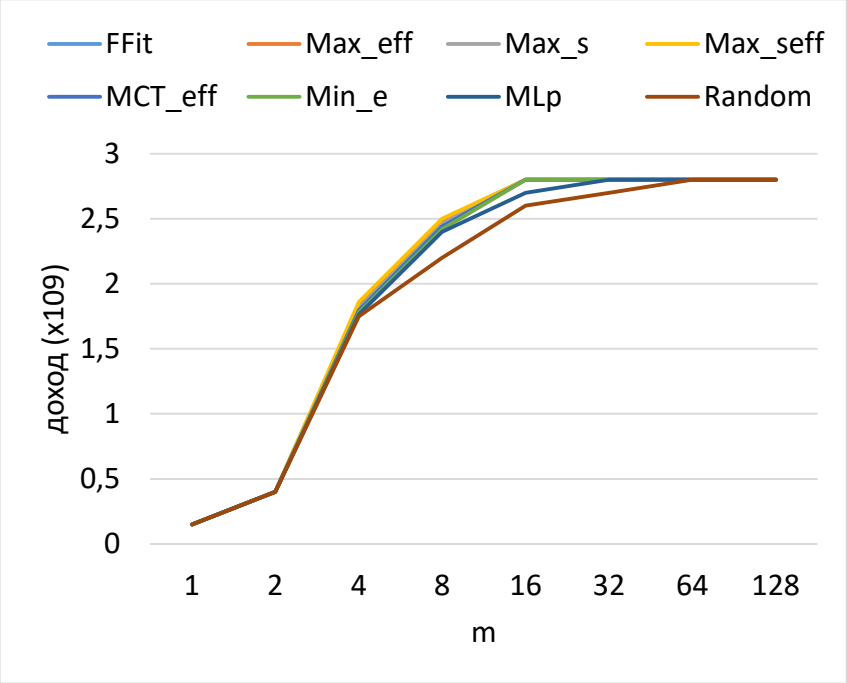


Рисунок 6.19 – Общий доход при использовании SLA с уровнями обслуживания

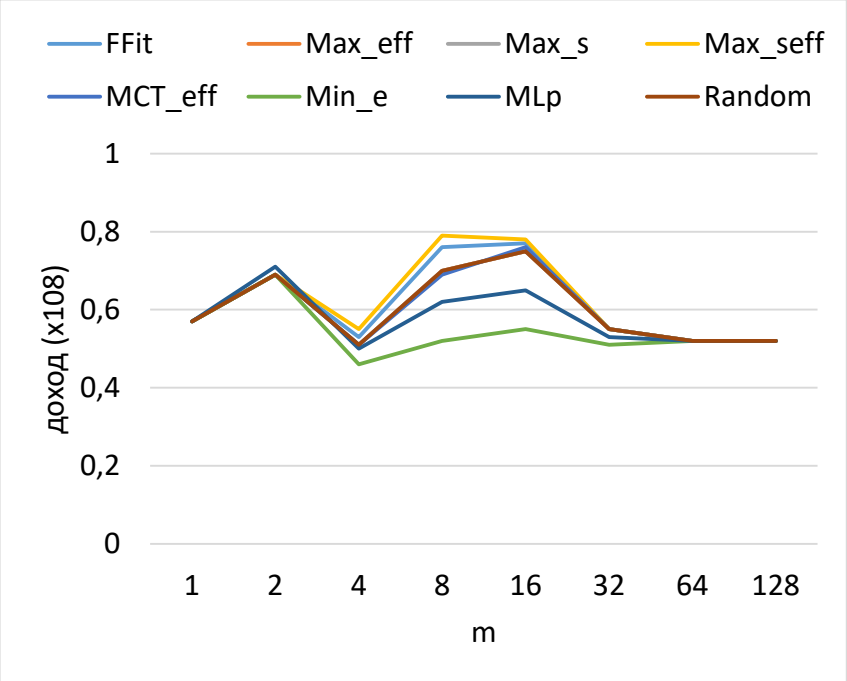


Рисунок 6.20 – Деградация доходов при использовании SLA с 4 уровнями обслуживания

На рисунке 6.20 показано, что стратегия Min_e, распределяющая работу на машину с минимальным общим энергопотреблением, планирует с меньшей деградацией доходов для 8, 16 и 32 машин.

В сценариях с m =1, 2, 64 и 128 исследуемые стратегии распределения

работ имеют незначительную разницу, поскольку в случае $m = 1, 2$ существует лишь небольшое разнообразие машин для распределения. В случае $m = 64$ и $m = 128$ почти все работы принимаются независимо от используемой стратегии, поскольку всегда есть доступные ресурсы. В то же время, в сценарии с 16 машинами видно четкое различие между исследуемыми стратегиями.

Распределение на машину с минимальным общим энергопотреблением Min_e имеет наилучшее поведение во всех сценариях. Второй лучшей стратегией является MLp, имеющая деградацию на 17% выше. Max_eff, Max_s и Max_seff имеют около 75% деградации по сравнению с Min_e. Разница между ними составляет менее 1%.

6.9.3 Деградация энергопотребления

Приведем анализ энергопотребления. На рис. 6.21 и 6.22 представлены результаты, полученные на тех же сценариях, которые были представлены в предыдущем подразделе. На рисунке 6.21 показано общее энергопотребление, а на рисунке 6.22 – деградация энергопотребления.

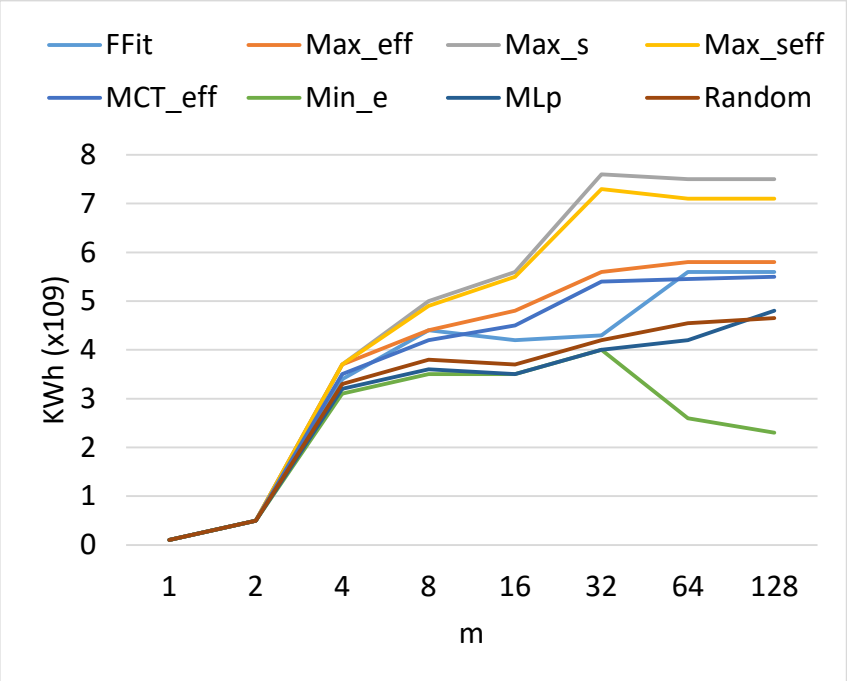


Рисунок 6.21 – Общее энергопотребление при использовании SLA с 4 SL

На рисунке 6.21 видно, что энергопотребление является возрастающей функцией при увеличении числа машин. Однако в диапазоне от 32 до 128 машин потребление практически одинаково для всех исследованных эвристик. Это означает, что все работы приняты, и энергопотребление не увеличивается. Машины без нагрузки выключены. Стратегия, которая распределяет работы на машину с меньшим общим энергопотреблением, примерно в четыре раза лучше, чем стратегия, которая распределяет работы на самую быструю машину.

Из результатов видно, что при увеличении числа уровней обслуживания в SLA энергопотребление немного увеличивается, а деградация немного выше, чем в сценариях с более низкими SL.

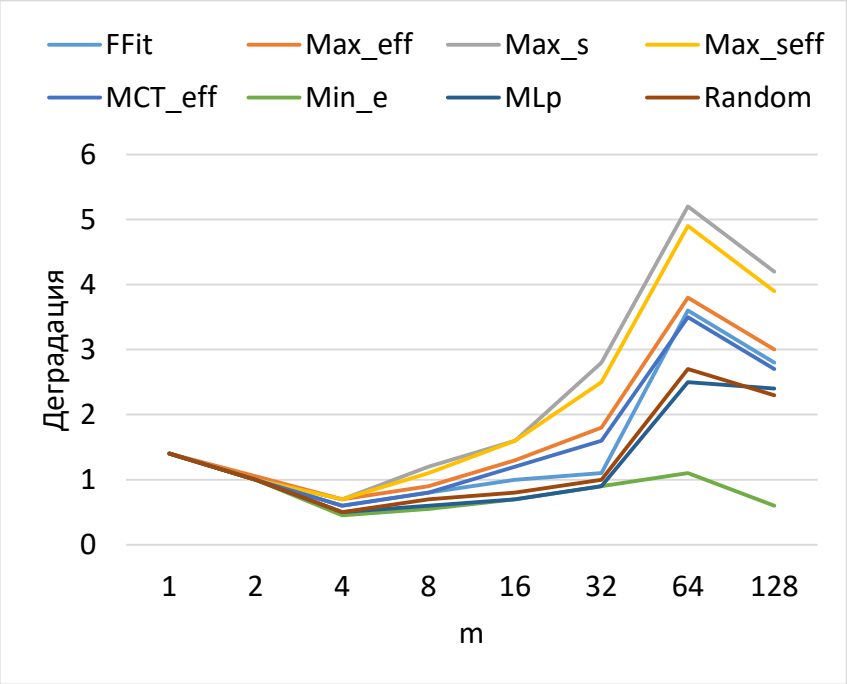


Рисунок 6.22 – Деградация энергопотребления при использовании SLA с 4 SL

Результаты на рисунке 6.22 показывают, что деградация в сценарии со 128 машинами уменьшается по сравнению с 64 машинами. При большем числе машин стратегии имеют больше возможностей для распределения ресурсов, и в целом все стратегии используют преимущества этого разнообразия. Стратегия Min_e имеет наилучшее поведение для минимизации потребления энергии.

6.9.4 Средняя деградация производительности

В предыдущих подразделах представлен анализ дохода и энергопотребления по отдельности. Представляет интерес поиск стратегии, которая обеспечивает наилучший компромисс между доходом и энергопотреблением. Для проведения этого анализа используется техника деградации и ранжирования, а также профиль производительности, описанные в разделах 6.1 и 6.2.

Таблица 6.4 – Деградация и ранжирование, SLA 4

Стратегия	Доход	Энергия	Среднее	Ранг I	Ранг E	Ранг
FFit	0.59	1.52	1.06	5	4	4
Max_s	0.60	2.32	1.46	7	6	6
Max_eff	0.60	1.75	1.18	8	8	8
Max_seff	0.60	2.20	1.40	6	7	7
MCT_eff	0.59	1.61	1.10	3	5	5
Random	0.59	1.30	0.94	4	3	3
Min_e	0.54	0.84	0.69	1	1	1
MLp	0.57	1.24	0.91	2	2	2

В таблице 6.4 приведены средние показатели деградации с четырьмя SL для каждого набора экспериментов. Последние три столбца таблицы содержат ранжирование каждой стратегии относительно дохода, мощности и их среднего значения. Рейтинг-I основан на деградации дохода. Ранжирование-E относится к позиции по отношению к деградации энергопотребления. Видно, что лучшей стратегией распределения ресурсов является назначение работ на машину, которая потребляет меньше энергии до момента распределения (Min_e). Это

приводит к лучшему среднему доходу и меньшему потреблению энергии.

Хорошая производительность этой стратегии обусловлена балансировкой нагрузки между машинами с учетом общего энергопотребления. Машина может иметь меньшее энергопотребление по разным причинам: машина может получать меньшую нагрузку, чем другие машины; она может иметь лучшую энергоэффективность, или и то и другое. Все ситуации приводят к балансировке нагрузки и приносят больший доход при меньшем энергопотреблении.

Вторая лучшая стратегия, MLp, распределяет работу на машину с меньшим числом работ. Она также стремится сбалансировать нагрузку, но этот баланс находится в зависимости от назначенной работы.

Анализ показывает, что если у нас нет информации о скорости машин или их энергоэффективности, то лучше распределять работу на ту машину, у которой меньше работ. Если у нас есть информация о скорости и энергоэффективности, то лучшим вариантом будет назначение работы на машину, которая на момент принятия решения потребляла меньше энергии.

6.9.5 Профиль производительности

Как упоминалось в разделе 6.1, выводы, основанные на средних значениях, могут иметь некоторые негативные аспекты. Чтобы проанализировать последствия того, что небольшая часть данных с большим отклонением будет доминировать в выводах, основанных на средних значениях. Приведем профили производительности стратегий.

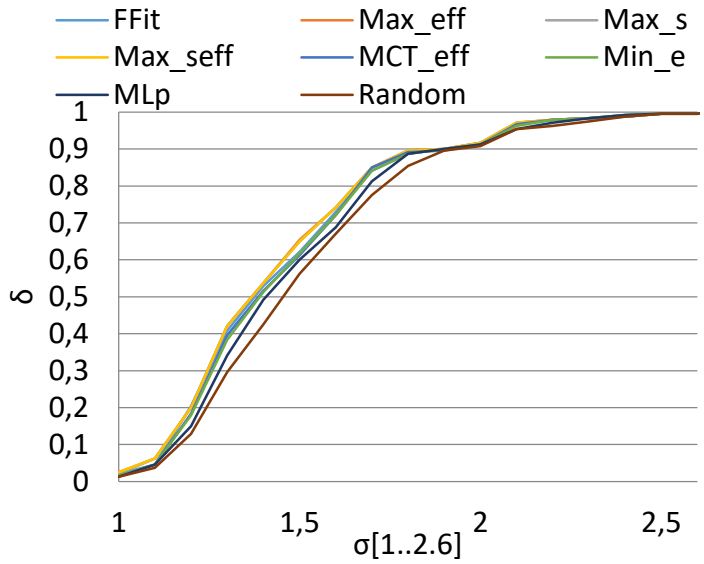


Рисунок 6.23 – Профиль производительности дохода, 8 стратегий

На рисунке 6.23 показаны профили производительности в зависимости от дохода в интервале $\sigma = [1,\ldots,2.7]$ для предоставления объективной информации для анализа тестового набора. Этот рисунок демонстрирует небольшие расхождения в доходах на значительном проценте задач. Min_e имеет самый высокий рейтинг и самую высокую вероятность того, что это лучшая стратегия. Вероятность того, что она является победителем на данной задаче в пределах фактора 1.6 от лучшего решения, близка к 0.7. Если выбирается в качестве границы нашего интереса нахождение в пределах фактора 2, то все стратегии будут подходящими с вероятностью 0.87.

На рисунке 6.24 показаны профили производительности в зависимости от энергопотребления в интервале $\tau = [1,\ldots,9]$. Он показывает большие расхождения в деградации энергопотребления на значительном проценте задач.

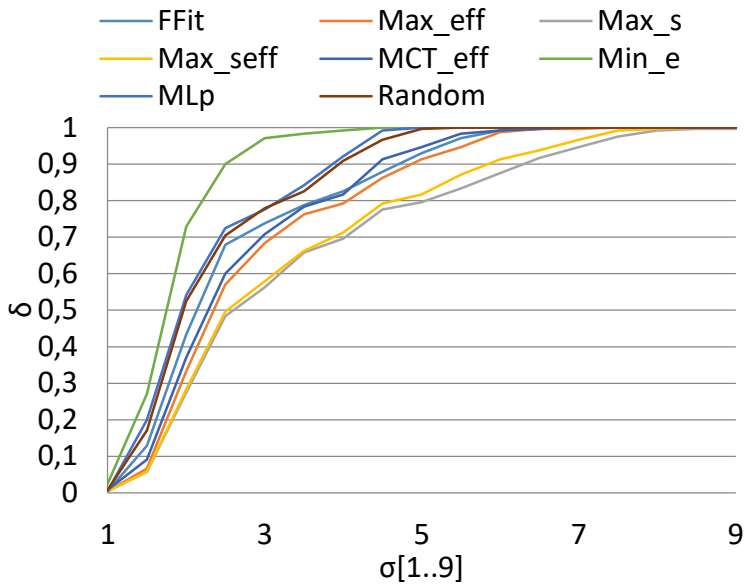


Рисунок 6.24 – Профиль производительности потребления энергии, 8 стратегий

Min_e имеет самый высокий рейтинг и самую высокую вероятность того, что это лучшая стратегия. Если выбирается в качестве области нашего интереса нахождение в пределах фактора 3, то вероятность того, что она окажется победителем на данной задаче, близка к 1. В пределах фактора 2 от сферы наших интересов она побеждает с вероятностью 0,7.

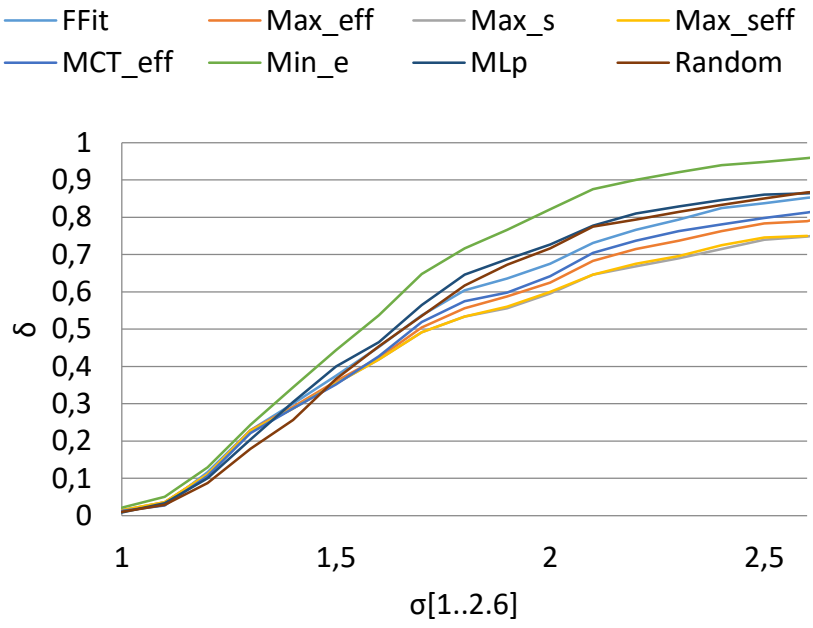


Рисунок 6.25 – Профиль производительности среднего энергопотребления и дохода, 8 стратегий распределения

Наиболее существенным аспектом на рисунке 6.24 является то, что на этом тестовом наборе Min_e доминирует над другими стратегиями: ее профиль эффективности никогда не ниже любой другой при всех значениях коэффициентов эффективности. MLp является второй лучшей стратегией.

На рисунке 6.25 показаны профили производительности 8 стратегий путем усреднения двух метрик: энергопотребления и дохода.

Сценарий 3: пять SLA. В предыдущем подразделе представлен подробный анализ сценария, в котором пользователям доступны 4 различные уровня обслуживания. В этом подразделе сравниваются пять случаев, различающихся типом SLA от SLA 1 с одним уровнем обслуживания до SLA 5 с пятью уровнями обслуживания.

На рисунках 6.26, 6.27, 6.28 показаны доход, мощность и их средние деградации для 8 стратегий, соответственно, при изменении SLA от 1 до 5.

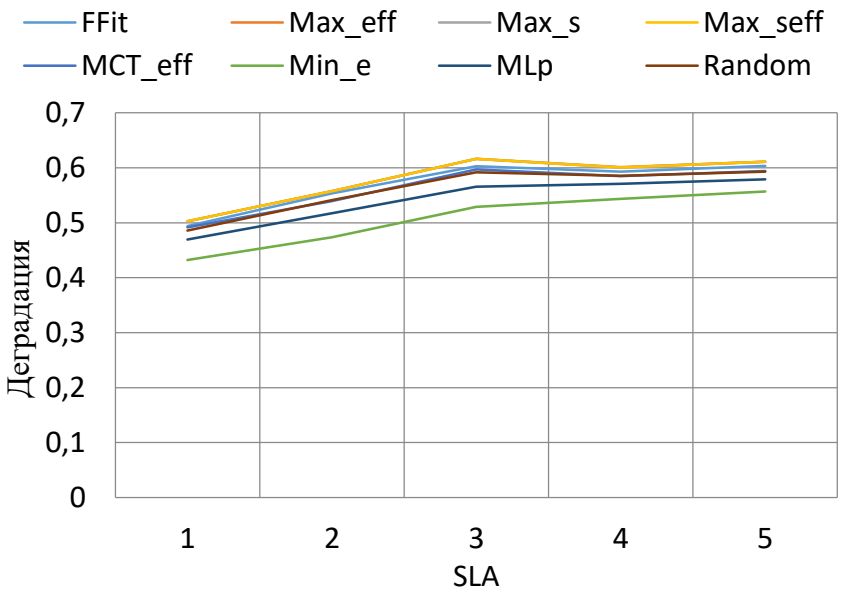


Рисунок 6.26 – Деградация среднего дохода при изменении SLA, 8 стратегий распределения

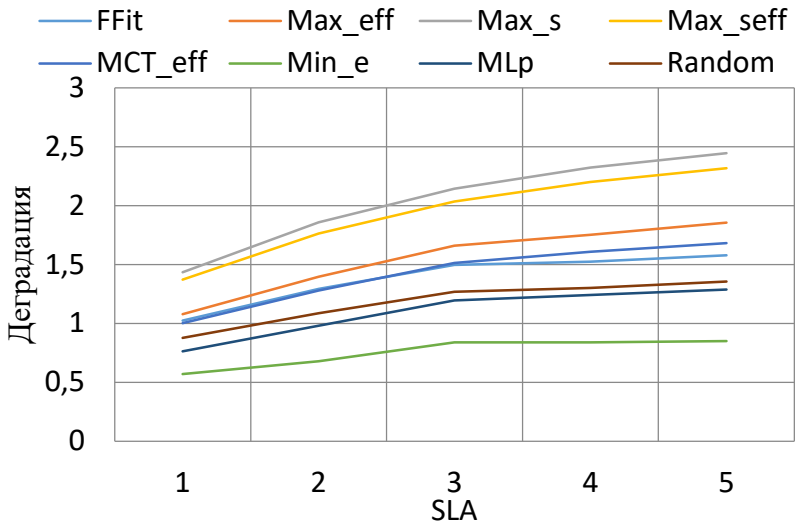


Рисунок 6.27 – Средняя деградация энергопотребления при изменении SLA, 8 стратегий распределения

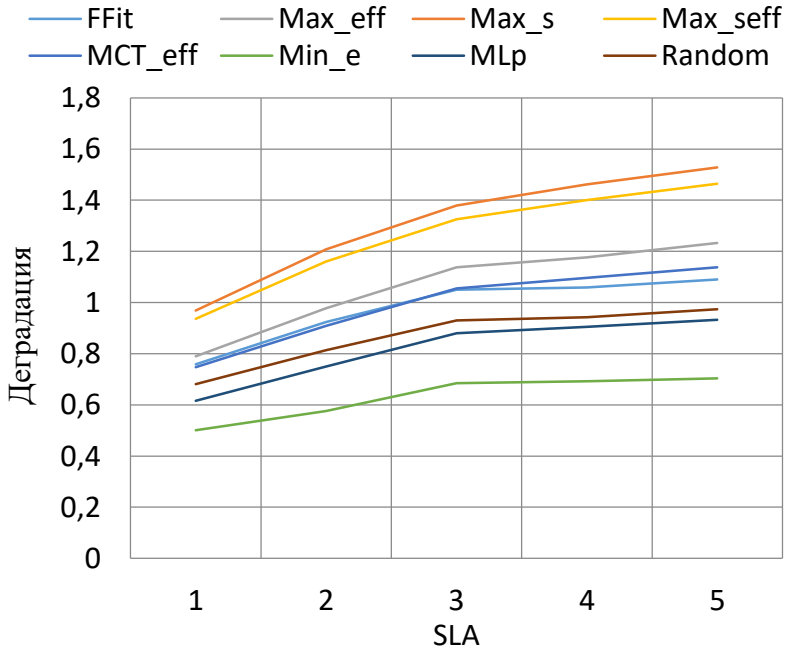


Рисунок 6.28 – Средняя деградация при изменении SLA, 8 стратегий распределения

Наиболее существенным аспектом этих результатов является то, что Min_e доминирует над другими стратегиями: ее деградация ниже для всех SLA. Она показывает лучший доход и меньшее энергопотребление.

В таблице 6.5 приведены значения деградации для SLA с одним, двумя, тремя четырьмя и пятью SL, а также их средние значения.

Таблица 6.5 – Деградация и ранжирование, MSL-PM

SLA	Стратегия	Доход	Энергия	Среднее	Ранг-I	Ранг-E	Ранг
SLA 1	FFit	0.49	1.02	0.76	5	5	5
	Max_s	0.50	1.43	0.97	7	8	8
	Max_eff	0.50	1.08	0.79	6	6	6
	Max_seff	0.50	1.37	0.94	8	7	7
	MCT_eff	0.49	1.00	0.75	4	4	4
	Random	0.49	0.88	0.68	3	3	3
	Min_e	0.43	0.57	0.50	1	1	1
	MLp	0.47	0.76	0.62	2	2	2
SLA 2	FFit	0.55	1.29	0.92	5	5	5
	Max_s	0.56	1.86	1.21	6	8	8
	Max_eff	0.56	1.40	0.98	8	6	6
	Max_seff	0.56	1.76	1.16	7	7	7
	MCT_eff	0.54	1.28	0.91	4	4	4
	Random	0.54	1.09	0.81	3	3	3
	Min_e	0.47	0.68	0.58	1	1	1
	MLp	0.52	0.98	0.75	2	2	2
SLA 3	FFit	0.60	1.50	1.05	5	4	4
	Max_s	0.62	2.14	1.38	7	8	8
	Max_eff	0.62	1.66	1.14	8	6	6
	Max_seff	0.62	2.03	1.33	6	7	7
	MCT_eff	0.60	1.51	1.06	4	5	5
	Random	0.59	1.27	0.93	3	3	3
	Min_e	0.53	0.84	0.68	1	1	1
	MLp	0.57	1.19	0.88	2	2	2

	Стратегия	Доход	Энергия	Среднее	Ранг-I	Ранг-E	Ранг
SLA 4	FFit	0.59	1.52	1.06	5	4	4
	Max_s	0.60	2.32	1.46	7	6	6
	Max_eff	0.60	1.75	1.18	8	8	8
	Max_seff	0.60	2.20	1.40	6	7	7
	MCT_eff	0.59	1.61	1.10	3	5	5
	Random	0.59	1.30	0.94	4	3	3
	Min_e	0.54	0.84	0.69	1	1	1
	MLp	0.57	1.24	0.91	2	2	2
SLA 5	FFit	0.60	1.58	1.09	5	4	4
	Max_s	0.61	2.44	1.53	7	6	8
	Max_eff	0.61	1.85	1.23	8	8	6
	Max_seff	0.61	2.32	1.46	6	7	7
	MCT_eff	0.59	1.68	1.14	4	5	5
	Random	0.59	1.36	0.97	3	3	3
	Min_e	0.56	0.85	0.70	1	1	1
	MLp	0.58	1.29	0.93	2	2	2
Mean	FFit	0.57	1.38	0.98	5	4	4
	Max_s	0.58	2.04	1.31	7	8	8
	Max_eff	0.58	1.55	1.06	8	6	6
	Max_seff	0.58	1.94	1.26	6	7	7
	MCT_eff	0.56	1.42	0.99	4	5	5
	Random	0.56	1.18	0.87	3	3	3
	Min_e	0.51	0.76	0.63	1	1	1
	MLp	0.54	1.09	0.82	2	2	2

Колонки Ранг-I, Ранг-E и Ранг в таблице содержат ранжирование каждой стратегии относительно дохода, энергии и их среднего значения. Ранг-I основан на деградации дохода. Ранг-E основан на деградации потребления энергии. Ранг основан на усреднении двух деградаций.

Видно, что средняя деградация дохода варьируется от 0.43 до 0.62, в то время как деградация энергопотребления имеет большую вариацию от 0.57 до 2.44. И снова Min_e имеет ранг 1 во всех тестовых случаях.

6.9.5 Двухкритериальный анализ

Основная цель – получить набор компромиссных решений, которые представляют собой хорошее приближение к фронту Парето. Формально это не фронт Парето, поскольку исчерпывающий поиск всех возможных решений не проводится, а скорее служит практическим приближением к фронту Парето.

6.9.5.1 Пространство решений и Парето-фронт

Вычисляем набор решений, аппроксимирующих Парето-фронт, для каждой из 8 стратегий: FFit, Max-s, Max-eff, Max-seff, MCT-eff, Min-e, Rand и MLp, 5 SLA, 8 конфигураций машин и 30 рабочих нагрузок. Таким образом, получим аппроксимацию фронтов Парето, рассматривая 8×5×8×30=9600 решений. Это двумерное пространство решений представляет собой допустимое множество решений, удовлетворяющих ограничениям задачи.

На рисунках 6.29–6.30 показаны множества решений и фронты Парето. Обратите внимание, что рассматривается проблема минимизации энергопотребления и максимизации дохода при соблюдении гарантий QoS. Для лучшего представления она преобразуется к минимизации двух критериев: ухудшения дохода и энергопотребления.

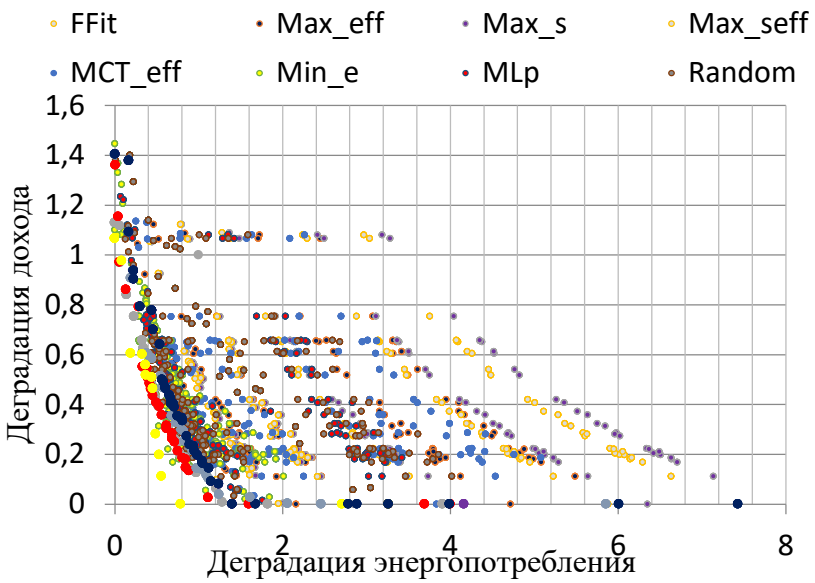


Рисунок 6.29 – Множество решений

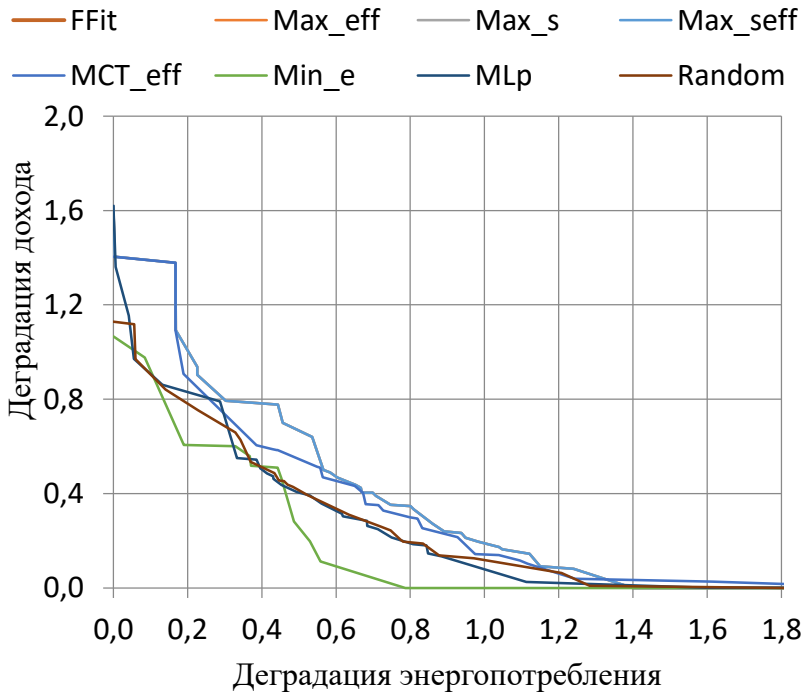


Рисунок 6.30 – Парето фронт

На рисунке 6.29 показано пространство решений (два критерия), включающее 1200 решений для каждой стратегии. Каждое решение представлено деградацией дохода и деградацией энергопотребления. Они охватывают широкий диапазон значений деградации E^{op} от 0 до 8, в то время как значения деградации

V находятся в диапазоне от 0 до 1,8.

На рисунке 6.30 показаны восемь приближений фронтов Парето, созданных исследуемыми стратегиями. Они охватывают деградацию энергопотребления от 0 до 1,8 и деградацию дохода от 0 до 1,6.

Видно, что Min-e, MLp и FFit расположены в левом нижнем углу, являясь одними из лучших решений с точки зрения обоих критериев. Они заметно превосходят MCT-eff.

Однако должны рассматриваться не только фронты Парето. Когда многие решения находятся за пределами фронта Парето, производительность алгоритма меняется. Это как раз случай FFit: хотя фронт Парето имеет высокое качество, многие из генерируемых решений находятся довольно далеко от него, и, следовательно, один запуск алгоритма может дать значительно худшие результаты. Решения FFit охватывают деградации E^{op} от 0 до 6, тогда как решения Min-e находятся в диапазоне от 0 до 3.3 деградации E^{op}.

Результаты анализа фронта Парето указывают на более стабильное поведение Min-e. Как и ожидалось, потери энергии растут с увеличением дохода. Для решений с наилучшим доходом (нулевая деградация) деградация энергопотребления превышает 1.4 для большинства стратегий. Min-e показывает лучший результат, со значением деградации 0.8.

6.9.5 2 Анализ покрытия множеств

В этом разделе используется метрика покрытия множеств, описанная в разделе 6.3, для расчета производительности исследуемых многокритериальных стратегий планирования. Используя эту метрику, можно сравнить два набора не доминирующих решений друг с другом.

В таблице 6.6 представлены результаты SCresults для каждого из восьми фронтов Парето. Строки таблицы показывают значения $SC(A,B)$ для доминирования стратегии A над стратегией B. Столбцы показывают $SC(B,A)$т. е.

доминирование B над A. Последние два столбца показывают среднее значение $SC(A,B)$ для доминирования строки A над столбцом B и ранжирование на основе среднего доминирования.

Аналогично, последние две строки показывают среднее доминирование B над A, и ранг стратегии в каждом столбце. Видно, что $SC(\text{Min}-\text{e}, B)$ доминирует над фронтами других стратегий в диапазоне 68%–93%, а в среднем 85%. $SC(A, \text{Min}-\text{e})$ показывает, что Min-e доминирует над фронтами других стратегий в среднем на 23%.

Ранжирование стратегий основано на проценте доминирования. Более высокий ранг в строках означает, что фронт лучше. Ранг в столбцах показывает, что чем меньше среднее доминирование, тем лучше стратегия. Согласно таблице метрики покрытия множества, стратегией, имеющей наилучший компромисс между максимизацией дохода и минимизацией энергопотребления, является Min-e, за ней следуют MLp и FFit на второй позиции.

Таблица 6.6 – Покрытие множеств и ранжирование

A \ B	FFit	Max-s	Max-eff	Max-seff	MCT-eff	Min-e	Random	MLp	Mean	Ранг
FFit	1.00	0.92	0.92	0.92	0.93	0.40	0.39	0.26	0.68	2
seMax-s	0.19	1.00	1.00	1.00	0.20	0.13	0.11	0.16	0.40	4
Max-eff	0.19	1.00	1.00	1.00	0.20	0.13	0.11	0.16	0.40	4
Max-seff	0.19	1.00	1.00	1.00	0.20	0.13	0.11	0.16	0.40	4
MCT-eff	0.11	0.70	0.68	0.71	1.00	0.13	0.07	.13	0.36	5
Min-e	0.70	0.97	0.95	0.97	0.93	1.00	0.68	0.74	**0.85**	**1**
Rand	0.26	0.95	0.92	0.95	0.90	0.33	1.00	0.26	**0.65**	**3**
MLp	0.33	0.95	0.89	0.92	0.90	0.33	0.39	1.00	**0.68**	**2**
Mean	0.28	0.93	0.91	0.92	0.61	0.23	0.27	0.27		
Ранг	3	7	5	6	4	**1**	**2**	**2**		

6.10 Выводы по шестой главе

В отличие от главы 5, где рассматривается сценарий с одним и двумя уровнями обслуживания, здесь для клиентов предоставляются различные уровни обслуживания.

Хотя большое число уровней приводит к высокой гибкости для клиентов, оно также приводит к значительным накладным расходам на планирование. Следовательно, необходимо найти подходящий компромисс и при необходимости динамически корректировать его.

В данной главе получены следующие основные результаты:

- Сформулирована проблема распределения ресурсов с несколькими уровнями обслуживания и качеством обслуживания для принятия решений по планированию работ и двухкритериальной оптимизации, учитывающей доход провайдера и энергопотребление.
- Проанализированы сценарии с однородными и разнородными машинами разных конфигураций и с различными рабочими нагрузками.
- Проведено всестороннее экспериментальное исследование жадных алгоритмов приемки с известной границей производительности в наихудшем случае и восемь стратегий планирования, учитывающих неоднородность среды (без использования знаний, с учетом энергопотребления и с учетом информации о скорости машин).
- Проведен совместный анализ двух конфликтующих критериев сначала на основе ухудшения производительности каждой стратегии по каждой метрике, затем на основе фронта Парето и метрики покрытия множества.
- Результаты показывают, что с точки зрения максимизации дохода провайдера и минимизации энергопотребления стратегия Min_e превосходит другие алгоритмы. Она доминирует почти во всех тестах. Данная стратегия стабильна даже в значительно отличающихся условиях. Она обеспечивает незначительное снижение производительности и справляется с различными

требованиями. Обнаружено, что информация о скорости машин не помогает существенно улучшить планирование. Стратегия Min_e требует минимальной информации, что важно в нестационарной среде, и небольшой вычислительной сложности; тем не менее, она достигает хорошего улучшения критериев и обеспечивает гарантии качества обслуживания.

Глава 7. ПЛАНИРОВАНИЕ НЕСТАЦИОНАРНОГО ОБЛАЧНОГО VOIP СЕРВИСА

7.1 Введение

Большие и малые корпорации внедряют в своих компаниях технологию Voice over IP (VoIP), чтобы снизить операционные расходы на уровне инфраструктуры. Она имеет более простое подключение и требует меньше аппаратного оборудования. Кроме того, VoIP считается долгосрочной перспективой развития услуг, предлагая более высокую гибкость и больше возможностей, чем традиционная телефонная – коммутируемая телефонная сеть общего пользования (англ., Public Switched Telephone Network – PSTN). Она способна справляться с различными рабочими нагрузками, динамически адаптировать ресурсы в зависимости от спроса.

Планирование предоставления сервисов зависит не только от свойств инфраструктуры, обрабатывающей вызовы и видеоконференции, а также осуществляющей передачу данных и т. д., но и от других пользователей, которые совместно используют ресурсы. Эта проблема осложняется непредсказуемым влиянием временных задержек при запуске VMs, изменением скорости обработки вызовов в зависимости от используемых кодеков и неопределенным временем разговоров.

Традиционные решения недостаточно точно учитывают такую неопределенность, неоднородность инфраструктуры и динамические изменения производительности, присущие масштабируемым VoIP сервисам.

Чтобы лучше понять последствия такой нестационарности, в этой главе исследуется свойства VoIP планирования, предложенного в [3, 28, 31].

Asterisk [85] обеспечивает работу систем IP Private Branch Exchange (PBX) в решениях CVoIP. Asterisk работает в экземплярах VMs для обеспечения вызовов, голосовой почты, видео/аудио конференций, интерактивных телефонных меню,

распределения вызовов и т. д. Помимо передачи изображений и текста, пользователи могут создавать новые функциональные возможности, открывая совершенно новый опыт телефонного общения.

Успех CVoIP в значительной степени зависит от QoS и цены. QoS – это первый аспект, который необходимо рассмотреть в модели, поскольку он имеет более строгие ограничения и чувствительные факторы. Доставка и обработка вызовов определяют QoS в CVoIP.

Наиболее важным аспектом доставки вызовов является адекватная передача голоса по IP-сети. Она зависит от времени прохождения пакетов через Интернет, задержек в очередях на маршрутизаторах, времени прохождения пакетов от источника к месту назначения, изменения задержки пакетов (джиттер), потери пакетов и т. д. Адекватное качество голоса - самый важный аспект обработки вызовов. Он учитывает время на установку и обрыв вызова, а также на преобразование голосовой части вызова в пакеты, передаваемые по сети.

Качество голоса субъективно воспринимается слушателем. Общим критерием, используемым для определения качества голоса, является средняя оценка мнений (англ., Mean Opinion Score – MOS). Она оценивает качество речи, обеспечиваемое кодеком. Каждый кодек предоставляет определенное качество речи только при достаточно низкой загрузке процессора. Теоретически, загрузка процессора на 100% обеспечивает наилучшую ожидаемую производительность. Однако при загрузке уже до 85% процессор не справляется с нагрузкой, появляются дрожание и разрывы звука [104].

Более того, дополнительные элементы, присущие облачным средам, могут влиять на QoS вызовов (например, время развертывания виртуальной инфраструктуры системы CVoIP, шифрование VoIP-трафика и т. д.). Все эти факторы необходимо изучить, чтобы минимизировать их влияние на качество голоса и гарантировать корректную обработку вызовов.

Стоимость – это второй аспект, который необходимо учитывать для успеха модели CVoIP. Провайдеры CVoIP должны оптимизировать аренду и использование ресурсов, чтобы предложить потенциальным клиентам

конкурентоспособные цены. Более того, неэффективное управление ресурсами оказывает прямое негативное влияние на производительность и стоимость. Аренда VMs в облаке используется в качестве параметра для оценки стоимости услуги CVoIP. Она позволяет провайдерам измерить стоимость системы с точки зрения спроса на ресурсы и времени их использования.

Провайдеры CVoIP могут предложить пользователю экономичные и адекватные услуги, если:

1. Аренда ресурсов для предоставления услуги максимально низкая.

2. Использование ресурсов достаточно низкое, чтобы гарантировать качество голоса, а значит, и качество услуги.

Обе цели находятся в конфликте. Провайдер должен максимизировать использование ресурсов для снижения стоимости инфраструктуры, но это ухудшает QoS. С другой стороны, снижение использования ресурсов гарантирует QoS, но увеличивает стоимость аренды инфраструктуры.

Провайдеры VoIP используют методы балансировки нагрузки для достижения высокого качества обслуживания и снижения расходов. К сожалению, существующие методы неэффективны в крупномасштабных системах, таких как облачные вычисления. В последние годы были представлены различные алгоритмы для преодоления проблем CVoIP. Однако распределение вызовов в CVoIP с учетом качества обслуживания и оптимизации затрат провайдера все еще недостаточно изучено.

В этой главе сформулирована проблема планирования VoIP-услуг в облачной среде и предложены две модели для моно-критериальной и би-критериальной задачи оптимизации. Рассмотрен частный случай динамической задачи упаковки в контейнеры и обсуждены решения для оптимизации стоимости провайдера, качества обслуживания и устойчивости.

Предложены и оценены несколько стратегий распределения вызовов, которые используют информацию об использовании VMs, времени предоставления VMs и задержках при запуске VMs и т. д. Эти факторы влияют на недостаточное или избыточное выделение ресурсов, качество вызовов и

стоимость. Проведен анализ стратегий, направленных на оценку числа VMs, необходимых для предоставления услуг VoIP без ухудшения качества, а также на сокращение числа активных VMs.

7.2 Интернет-телефония

VoIP относится к предоставлению услуг связи через публичный Интернет, а не через традиционную телефонную сеть. VoIP – это эволюционный шаг в передаче голоса на большие расстояния, он заключается в передаче закодированного голоса в цифровой форме через Интернет. Кодированный сигнал должен быть декодирован на принимающей стороне, чтобы получатель смог услышать голос отправителя.

Некоторые элементы для установления соединения включают протокол инициирования сеанса (SIP), протоколы передачи голоса и видео, а также услуги, используемые сегодня в Интернете. Услуги VoIP значительно снижают тарифы на вызовы по сравнению с обычными телефонными услугами. Тем не менее, ведущие поставщики VoIP по-прежнему ведут ожесточенную конкурентную борьбу, предлагая необычные услуги с использованием современных облачных технологий.

Преимуществами VoIP перед традиционными телефонными сетями являются:

Снижение затрат. Стоимость использования телефона VoIP дешевле, чем обычного телефона, например, тарифы междугородних вызовов и соединений с традиционными телефонами меньше, вызовы между телефонами VoIP бесплатны, и пользователи могут иметь более одного телефонного номера в разных городских кодах и т. д.

Портативность. Пользователи могут совершать и принимать вызовы, войдя в свою учетную запись VoIP. Как и системы электронной почты, телефонная система VoIP доступна по всему миру.

Гибкость. Пользователи могут по-прежнему использовать свой обычный телефон с помощью VoIP-конвертера или телефонного адаптера VoIP. Они могут принимать телефонные вызовы на тот же номер, чтобы получить собственный VoIP-номер.

Пополняемость. VoIP добавляет расширенные функции. Например, переадресация вызова, ожидание вызова, голосовая почта, определитель номера и трехсторонняя связь – это некоторые услуги, включенные в телефонную услугу VoIP без дополнительной оплаты.

Для предоставления услуги VoIP компаниям необходимо развернуть инфраструктуру VoIP в нескольких районах.

Общая архитектура VoIP включает элементы коммуникационной инфраструктуры, обеспечивающие удаленное подключение телефонов через Интернет. Серверы обеспечивают шлюз, межсетевые коммутаторы, контроллеры сеансов, файервол и т. д. Они используют программное обеспечение Asterisk для эмуляции телефонной станции.

На рисунке 7.1 показана общая архитектура VoIP. Элементы коммуникационной инфраструктуры подключены к Интернету и соединяют телефоны удаленно через Интернет.

Рисунок 7.1. Архитектура VoIP.

Недостаток этой архитектуры проявляется, когда аппаратное обеспечение достигает своей максимальной мощности. Традиционные решения VoIP трудно

наращивать, поэтому для масштабирования системы необходимо увеличить инфраструктуру или заменить существующее оборудование. Избыточное оборудование и связанные с ним расходы не являются эффективным решением, даже с учетом растущего числа клиентов и потенциальной безопасности, связанной с возможностью предоставления услуг в часы пик или при аномальном поведении системы.

Более того, VoIP требует постоянной доступности для любого числа пользователей. Провайдеры могут инвестировать в большую инфраструктуру, чтобы избежать потери вызовов (следовательно, пользователей) и справиться с растущим числом клиентов. В этом случае инфраструктура обычно используется недостаточно эффективно. Более того, серверы необходимо заменять из-за деградации ресурсов.

Компания MiXvoip [171] разработала концепции голосового суперузла (англ., Super Node – SN) и кластера суперузлов (англ., Super Node Cluster – SNC) для обогащения функций телефонных станций.

SNC – это набор SN, развернутых в облаке и логически связанных между собой на локальном уровне. SNC позволяют минимизировать путь между двумя локальными пользователями и повысить качество голоса. Такое развертывание обеспечивает избыточность в данной географической области и гарантирует высокое качество передачи голоса между SNC через публичный Интернет. SNC развертываются в широко распределенных географических точках.

Составляющие элементы (слои) SN показаны на рисунке 7.2. Основными компонентами слоев являются: операционная система (ОС), система безопасности, система мониторинга и голосовой узел.

ОС выполняет программное обеспечение VoIP (Asterisk). В качестве основного варианта используются дистрибутивы LINUX, но может быть использована и другая ОС. Уровень безопасности определяет доступ к серверу, например, пользователь может получить доступ к телефону через свой логин и пароль, используя SSH и FTP сессии. Мониторинг проверяет использование ресурсов и сообщает о неполадках голосовых узлов. Голосовой узел (англ., Voice

Node – VN) содержит функции телефонной системы: голосовую почту, перевод вызова, музыку на удержании, функцию конференции, и прочее. Такая распределенная архитектура распределяет вычисления на несколько узлов, снижает риск сбоев и ограничивает расходы на приобретение оборудования.

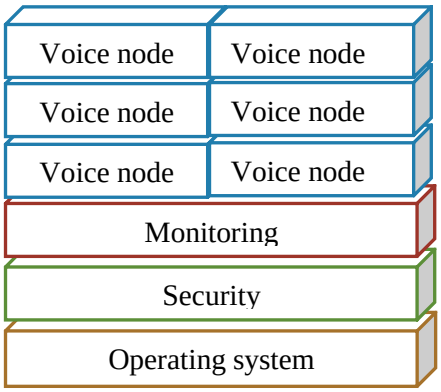


Рисунок 7.2 Голосовая архитектура суперузла**.**

MIXvoip использует концепцию SN и SNC для увеличения пропускной способности и предоставления услуг. В этой модели SNs развертываются несколькими облачными провайдерами с использованием модели IaaS.

Модель Hybrid VoIP (HVoIP) является усовершенствованием традиционной модели VoIP на базе SNs. HVoIP использует традиционную инфраструктура VoIP и VMs для развертывания услуг VoIP, когда установленные серверы достигают своей максимальной мощности.

HVoIP обеспечивает несколько преимуществ по сравнению с традиционным VoIP: улучшает масштабируемость системы и снижает избыточное предоставление ресурсов, гранулярность и стоимость. К сожалению, у HVoIP есть и недостатки: Это дорогостоящая модель, поскольку провайдерам по-прежнему необходимо арендовать офисы и помещения и поддерживать ресурсы для предоставления услуги. Кроме того, это может привести к увеличению неиспользуемых ресурсов; провайдер должен максимально

использовать собственные ресурсы и сокращать арендуемые ресурсы. Модель, полностью основанная на облачных вычислениях, является возможным усовершенствованием HVoIP.

В облачном VoIP (CVoIP) инфраструктура арендуется в локальных облаках. Это позволяет провайдерам VoIP сократить расходы на инфраструктуру и незадействованные ресурсы. Кроме того, CVoIP позволяет увеличить избыточность на географической территории и обеспечивает высокое качество передачи голоса между SN. Подобно HVoIP, SN работают в VMs в некоторой облачной среде. Кроме того, CVoIP добавляет новые функции и возможности, обеспечивает более простую реализацию и интегрирует динамически масштабируемые услуги. Другие преимущества включают доступность, целостность и безопасность передачи данных.

На рисунке 7.3 показано облачное решение этой проблемы. Голосовые узлы предоставляют различные услуги, например, перевод вызова, голосовую почту, функцию конференции, музыку на удержании и т. д. Преимуществом этой архитектуры является повышенная масштабируемость, т. к. VMs могут быть распределены по нескольким процессорам.

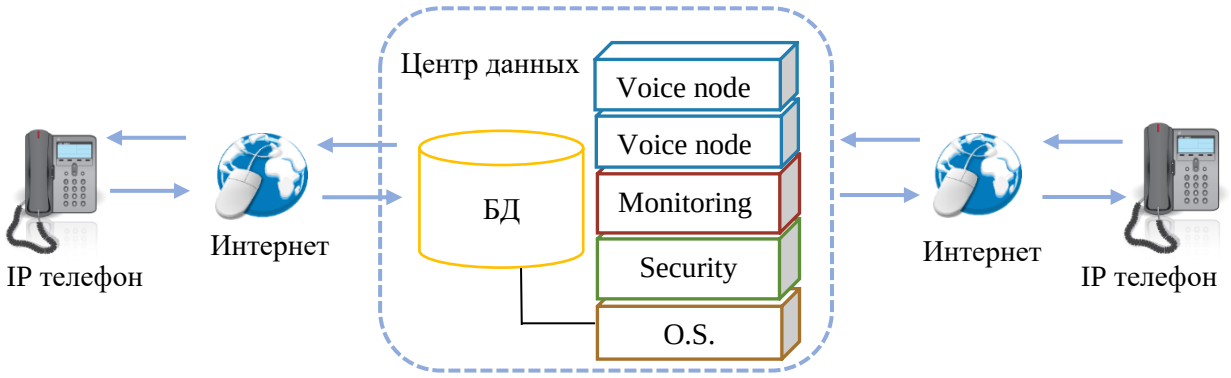


Рисунок 7.3. Облачная архитектура VoIP.

Можно выделить следующие преимущества суперузлов в CVoIP:

Гранулярность. Суперузлы уменьшают конфликты между рабочими станциями пользователей и центральной инфраструктурой при обновлении

производительности центральной. Они обладают полной совместимостью между производителями телефонов и телефонными станциями (управляют несколькими версиями программного обеспечения телефонов).

Масштабируемость. В случае насыщения инфраструктура может увеличить число узлов без изменения существующей архитектуры или влияния на обслуживание потребителей.

Распространяемость. Суперузлы могут быть развернуты в других географических районах, при этом провайдерам не нужно физически устанавливать серверы и оборудование, им достаточно арендовать инфраструктуру в локальных облаках.

Отказоустойчивость. Распределение узлов позволяет достичь большей отказоустойчивости. CVoIP может обнаруживать нерабочие узлы и автоматически перенаправлять трафик на здоровые экземпляры до тех пор, пока не будут восстановлены нерабочие экземпляры.

Снижение затрат. Использование облачных систем снижает стоимость инфраструктуры и инвестиции в оборудование. Провайдеры платят только за используемые ресурсы.

В системе CVoIP распределенная архитектура предполагает, что VN распределены географически, следовательно, они группируются в разных местах (центрах обработки данных). Для развертывания и эффективного управления телефонами через облака необходимо улучшить различные характеристики системы, наиболее важной из которых является использование инфраструктуры.

Для оптимизации общей производительности системы необходимо сбалансировать загрузку процессора обработки голосового сигнала по IP. Перегрузка процессора снижает качество голоса в вызове. Аналогичная проблема может возникнуть и с пропускной способностью сети. Кроме того, время простоя процессора увеличивает бесполезные расходы провайдера.

Распределение вызовов максимизирует производительность CVoIP, уменьшая число активных ресурсов и сохраняя время их простоя как можно меньше. Чтобы минимизировать снижение качества, все ресурсы должны

содержать одинаковый объем вычислительной работы без превышения порога, гарантирующего QoS. Таким образом, преимущества технологии CVoIP будут использованы только при применении эффективных методов планирования.

7.3 Обзор литературы

Миграция VoIP-бизнеса в облачные вычисления вызывает интенсивные исследования в нескольких областях планирования: распределение вызовов, балансировка нагрузки, качество обслуживания, прогнозирование нагрузки, оценка текущей нагрузки и т. д. В следующем разделе рассматриваются последние работы по распределению вызовов, балансировке нагрузки и оценке нагрузки, связанные с управлением VoIP.

7.3.1 Упаковка в контейнеры

Проблема упаковки в контейнеры (англ., bin packing problem) – одна из самых старых и наиболее изученных проблем в компьютерных науках. Набор предметов, каждый из которых имеет размер в диапазоне (0, 1], должен быть упакован в минимальное число контейнеров. Исследование классической задачи одномерной упаковки в контейнеры началось в 1970-х годах с работ Джонсона и Ульмана [139, 212].

Проблема динамической упаковки в контейнеры (DBP) была представлена в 1983 году как обобщение классической проблемы упаковки в контейнеры [94]. Задача распределения вызовов формулируется как DBP, где элементы (вызовы) прибывают и убывают (завершают выполнение вызова в системе CVoIP) в произвольное время. Основная цель – упаковать последовательность элементов с единичными долями (т.е. элементы с размерами 1/w для некоторого целого числа $w \geq 1$) в контейнеры с единичными размерами так, чтобы максимальное число контейнеров, используемых за все время, было минимальным.

В [205] предложен алгоритм онлайн-упаковки в контейнеры под названием Variable Item Size Bin Packing (VIS-BP). Он распределяет ресурсы центра обработки данных, используя живую миграцию VMs. Основная цель – ограничить

комбинации задач в контейнере, чтобы минимизировать количество пустого пространства. Авторы оценивают эффективность алгоритма и приводят теоретическое доказательство для числа используемых серверов и миграций VM. В многомерной версии рассматривается сочетание процессора и сети.

В [216] анализируется широкий спектр контроллеров для размещения и перераспределения VMs с использованием доступности ресурсов (CPU, память и т.д.). Поступающие VMs размещаются на серверах до тех пор, пока позволяет их остаточная емкость. На этом этапе используются алгоритмы упаковки в контейнеры. Контроллер перераспределения запускает миграцию VMs, когда недогруженные серверы опустошаются, а перегруженные освобождаются. Авторы обнаружили, что комбинации контроллеров размещения и периодического перераспределения достигают наивысшей энергоэффективности при условии заданного уровня обслуживания.

В [158] рассматривается вариант задачи динамической упаковки в контейнеры для решения проблемы диспетчеризации запросов, возникающей в облачных играх. В облачной игровой системе компьютерные игры запускаются на облачных серверах, где каждый игровой экземпляр требует определенного числа ресурсов. Для того чтобы обеспечить хороший пользовательский опыт, запросы должны отправляться с достаточным числом ресурсов CPU и GPU. Основной задачей является минимизация числа серверов (контейнеров) для обработки игровых запросов (элементов). Авторы анализируют конкурентный фактор для оригинальной и модифицированной версий алгоритмов Best Fit и First Fit.

7.3.2 Балансировка нагрузки

Основной целью балансировки вызовов является снижение затрат на инфраструктуру и гарантия того, что услуги будут предоставляться наилучшим образом. Для улучшения производительности системы CVoIP было предложено несколько алгоритмов.

В [122] представлена система планирования в реальном времени в виртуализированной среде, которая учитывает ограничения реального времени приложений. Многоядерное динамическое разделение – это механизм динамического разделения физических процессоров на два пула в соответствии с параметрами планирования VMs. Подход использует стратегию глобального планирования вызовов на VMs с экземпляром Asterisk, чтобы улучшить использование ЦП и гарантировать качество вызовов.

В [166] изучается влияние шифрования IPSec на загрузку процессора, пропускную способность и качество голоса. Период голосовой нагрузки оказывает значительное влияние на загрузку процессора и пропускную способность. Результаты показывают, что стратегии могут сэкономить до 40-60 % пропускной способности при правильном выборе периода. Качество вызова, выраженное по шкале MOS, остается практически неизменным вплоть до момента, когда загрузка процессора приближается к 80 %.

В [95] показывается, что PBX сервер Asterisk способен обеспечить возможности VoIP-связи с приемлемым качеством. Авторы используют метрику вероятности блокировки для измерения пропускной способности VoIP-сервера и MOS для оценки качества голосовых вызовов. Результаты эксперимента показывают, что PBX Asterisk, использующая SIP, эффективно обрабатывает более 160 одновременных голосовых вызовов с вероятностью блокировки менее 5%.

В [172] анализируется стратегия Virtual Load-Balanced Call Admission Controller (VLBCAC) для обеспечения контроля допуска для SIP-серверов и балансировки поступающих вызовов. VLBCAC имеет механизмы для прогнозирования числа вызовов, требуемых ресурсов и выбора наиболее подходящих экземпляров VMs с учетом процессора, памяти и пропускной способности. Предлагаемая модель учитывает максимизацию использования ресурсов и пропускной способности системы.

В [144] исследуется масштабируемость SIP-серверов на облачных вычислениях. Автор создает облачную телекоммуникационную службу, которая

растет и сокращается по требованию. Система реагирует на изменения задержки VMs. Она устойчива к сбоям отдельных компонентов в облаке. Менеджер масштабирования нагрузки (LSM) связывает различные технологии в единую целостную систему. LSM принимает решение об увеличении или уменьшении масштаба в зависимости от нагрузки, проверяет состояние работоспособности всех узлов, следит за очередью вызовов, контролирует очередь на частоту вызовов и распределяет нагрузку.

7.3.3 Оценка нагрузки

В настоящее время проводится несколько исследований с целью прогнозирования нагрузки для эффективного ее распределения. Основная идея заключается в прогнозировании входящего трафика (вызовов для VoIP) для минимизации затрат на инфраструктуру и улучшения QoS для конечного пользователя.

В [138] исследуется динамическое масштабирование телекоммуникационных услуг, фокусируясь на SIP-сервисах с учетом состояния вызовов. Во-первых, авторы представляют и оценивают два протокола для прозрачной миграции текущих сессий между SIP-серверами. Во-вторых, в исследовании рассматривается потенциальная ценность проактивного выделения ресурсов на основе прогнозирования нагрузки. Для реализации краткосрочного прогнозирования нагрузки используется самоадаптивный фильтр Калмана. Он сочетается с прогнозами на основе истории, чтобы предвидеть будущие изменения вызовов. Основной целью является снижение эксплуатационных расходов без ущерба для доступности услуг.

В [199] сравниваются две модели прогнозирования для реальной среды VoIP. Интерактивная система частиц (IPS) и нейронная сеть (HC) использовались для прогнозирования входящего голосового трафика во временных интервалах.

В [198], расширяя предыдущую работу, используют IPS, Gaussian Mixture Model (GMM) и Gaussian Process (GP) для оценки входящего голосового трафика. Были предложены гибкие подходы к моделированию, методы оценки трафика, и масштабируемые решения с хорошей точностью предсказания. Все алгоритмы были обучены и протестированы в различных сценариях.

В [88] предложен модуль прогнозирования рабочей нагрузки в облаке для SaaS-провайдеров на основе модели авторегрессионного интегрированного скользящего среднего (ARIMA). В исследовании анализируется влияние «компонента анализатора рабочей нагрузки» с точки зрения эффективности использования ресурсов и QoS. Он оценивает точность прогнозирования будущей рабочей нагрузки, используя реальные трассировки запросов к веб-серверам.

В [162] исследуются динамические характеристики облачных нагрузок и анализируют три алгоритма в качестве их предикторов для повышения энергоэффективности облачных систем. Авторы оценивают эффективность нейронной сети Rand Variable Learning Rate Backpropagation Neural Network (RVLBPNN), Hidden Markov Modelling (HMM) и Naive Bayes Classifier (NBC) для прогнозирования будущих рабочих нагрузок в облачных центрах обработки данных. Все модели оцениваются на предмет их эффективности в прогнозировании рабочих нагрузок с интенсивным использованием памяти и процессора. Основной целью является поддержание высокого уровня QoS и снижение использования ресурсов центра обработки данных.

Rate of Change (RoC) [90] – это стратегия балансировки нагрузки в распределенной системе. Она позволяет запустить динамический, распределенный и неявный механизм балансировки. Балансировщик (Bal) принимает решения о распределении заданий во время выполнения, локально и асинхронно. Каждый Bal учитывает свою собственную нагрузку. Миграция не зависит от нагрузки других Bal. Решение о миграции зависит от текущей нагрузки, изменения нагрузки за временной интервал (скорость изменения нагрузки) и текущих параметров балансировки нагрузки. Разница в нагрузке используется в качестве оценки нагрузки, ее прогнозирования для следующего

временного интервала и определения необходимости процесса балансировки нагрузки.

7.3.4 Балансировка нагрузки в распределенных вычислительных средах

В дополнение к анализу bin-packing и оценки нагрузки рассмотрим достижения в алгоритмах балансировки нагрузки, которые могут дать идеи для того, чтобы предложить адекватную технику распределения вызовов.

VectorDot (VD) [200] — это алгоритм балансировки нагрузки для работы с ресурсами в иерархических системах распределенных вычислений. Алгоритм вдохновлен методом Тойоды для многомерных ранцев. Он позволяет перемещать VMs, виртуальные хранилища и трафик коммутаторов с одного узла на другой без простоя работающего в нем приложения. Функция «точечного продукта» описывает перегруженные серверы из-за чрезмерного использования ЦП, памяти, сети или дискового ввода-вывода. Она используется для поиска узлов (серверов или коммутаторов) для миграции VMs или трафика. Алгоритм проверен в ходе экспериментов как в реальном центре обработки данных, так и в смоделированной среде крупномасштабного центра обработки данных.

Virtual Web Resource (VWR) [176] – это алгоритм балансировки нагрузки для распределения клиентских запросов между реплицированными серверами. Он использует протокол для ограничения скорости перенаправления и предотвращения перегрузки удаленных серверов. Когда сервер перегружен, он получает разрешение на снижение нагрузки и перенаправляет превышающую нагрузку на ближайший веб-сервер с меньшей нагрузкой. Серверы могут принять или отклонить запрос, если запрос отклонен, то для его выполнения выбирается другой сервер.

Стратегия планирования при балансировке нагрузки на VM-ресурсах – SVB [133] – это алгоритм балансировки нагрузки для миграции VMs в среде облачных вычислений. Алгоритм, основанный на генетических алгоритмах (GA),

учитывает вариации системы, исторические данные и стоимость миграции. Он пытается уменьшить или избежать динамической миграции. Хромосома представлена в виде древовидной структуры. Корневой узел – это планирование, промежуточные узлы – реальные машины, а листовые узлы – VMs. Предлагаются соответствующие стратегии выбора, гибридизации и вариации. Этот метод способен реализовать балансировку нагрузки и разумное использование ресурсов, когда нагрузка на систему стабильна и вариативна.

Honeybee Foraging Behavior (HF) [188] — это децентрализованный алгоритм балансировки нагрузки для крупномасштабных облачных систем. Он динамически распределяет веб-сервисы на серверах для регулирования системы в зависимости от спроса. Этот алгоритм используется для самоорганизации и достижения глобальной балансировки нагрузки посредством локальных действий сервера. Каждый физический сервер группирует виртуальные серверы, каждый со своей очередью заданий. Понятие «доска объявлений» используется для сообщения о глобальной прибыли колонии. Незанятые серверы читают доску объявлений, выбирают объявление и обслуживают запрос или случайным образом обслуживают запрос очереди серверов.

Particle Swarm Optimization (TBSLB-PSO) [187] – это механизм балансировки нагрузки для облачных вычислений. Этот алгоритм использует центральный планировщик задач (CTS) для передачи дополнительных задач с перегруженной VM на новую аналогичную VM путем применения информации на доске. Доска содержит всю информацию планировщиков о характеристиках VMs, выполняемых задачах и качестве обслуживания (QoS). Процесс миграции учитывает объем данных, памяти, пропускную способность и число VMs. Неработающие физические машины (ФМ) не будут выбраны в качестве новых хостов для VMs; это позволяет снизить энергопотребление.

Теория муравьиных колоний и сложных сетей (ACCLB) [220] – это механизм балансировки нагрузки для открытой федерации облачных вычислений. Муравей периодически отправляется недогруженным узлом для поиска нагрузки. Во время путешествия он запоминает узел, который имеет

максимальную/минимальную нагрузку, и останавливается в узле, превышающем порог нагрузки (разница между max и min) или общее число шагов перемещения. Затем муравей сообщает двум узлам Nmax и Nmin, чтобы сбалансировать нагрузку между ними. Эта система направлена на решение сложной и динамической задачи балансировки, при этом учитываются характеристики сложной сети (малый мир и свобода от масштабируемости).

Event Driven (ED) [175] – это алгоритм балансировки нагрузки для многопользовательских онлайн-игр (MMOG) в реальном времени на облачных ресурсах. Он представляет собой экономически эффективный хостинг для сессий MMOG. Алгоритм использует событийно-ориентированное решение, которое получает информацию о пропускной способности, анализирует ее в контексте и принимает решение о выделении услуг. События пропускной способности состоят из сессии и ресурсов, статуса события, мониторинга и прогнозирования.

Fuzzy Based Round Robin (FBRR) [196] – это алгоритм балансировки нагрузки на основе нечеткой логики и RR в центре обработки данных в облаке. Он хранит информацию о каждой виртуальной машине и их запросах, которые в настоящее время распределены. Наименее загруженная машина определяется при поступлении запроса на выделение. Фаззификатор учитывает скорость процессора и нагрузку VM для балансировки системы. Fuzzy Inference System (FIS) имитирует принятие решений человеком на основе нечетких правил управления и соответствующих входных лингвистических параметров.

В таблице 7.1 приведен обзор алгоритмов балансировки нагрузки. В таблице 7.2 представлены основные характеристики алгоритмов, а в таблице 7.3 обобщены метрики, использованные для изучения качества алгоритмов.

Таблица 7.1. Среда алгоритмов балансировки нагрузки

Аббревиатура	Алгоритм	Центры обработки	Многоядерные вычисления	Грид-вычисления	Распределенные вычисления	Облачные вычисления	Облачная федерация	WEB-сервис	хранение данных	Ссылка
VD	VectorDot	•								[200]
WCAP	Decentralized Workload and Client Aware Policy				•					[170]
VWR	Virtual Web Resource							•		[176]
JIQ	Job-Idle-Queue	•								[163]
LFM	Lock-Free Multiprocessing		•							[160]
SVB	Scheduling Strategy on LB of VM Resources					•				[133]
CLBVM	Central LB policy for VMs							•		[73]
LBVS	Load Balance Strategy for Virtual Storage								•	[159]
TSLB	Task Scheduling Based on LB					•				[106]
HF-1	Honeybee Foraging Behavior							•		[188]
HF-2	Honeybee Foraging Behavior					•				[149]
BRS	Biased Random Sampling							•		[188]
AC	Active Clustering							•		[188]
TBSLB-PSO	Task-Based System Load Balancing Using Particle Swarm Optimization					•				[187]
LBS-BF	Load Balance Scheduling Based on Firefly					•				[115]
ACCLB	Ant Colony and Complex Network Theory						•			[220]
OLB +LBMM	Two-phase scheduling					•				[215]
ED	Event-Driven					•				[175]
Carton	(LB + DRL)					•				[207]
CB	Compare and Balance					•				[221]
TSPBRR	Time Slice Priority Based Round Robin.					•				[194]
ESCE	Equally Spread Current Execution					•				[142]
CLBDM	Central Load Balancing Decision Model.					•				[185]
MR	MapReduce				•					[127]
VMMP	Virtual Machine to Physical Machine Mapping					•				[180]
FBRR	Fuzzy Based Round Robin	•								[196]
PALB	Power-Aware Load Balancing					•				[119]
LBMM	Load Balancing Min Min			•						[145]
Min-Min	Min-Min			•						[145]
Max-Min	Max-Min			•						[145]
SOAS	Self-organized agent system for dynamical				•					[148]
RoC-LB	Rate of change load balancing algorithm				•					[90]

Таблица 7.2. Основные характеристики алгоритмов балансировки нагрузки.

	Централизованная	Децентрализованная	Иерархическая	Онлайн	Офлайн	Non-clairvoyant	QoS	Стоимость миграции	Сетевая задержка	историческая информация	Статическая	Динамическая	Репликация	Миграция
VD			•		•	•						•		VM VS, and traffic
WCAP			•	•		•						•	Server	-
VWR	•			•		•			•			•	Server	Request
JIQ		•		•		•						•		-
LFM	•			•		•						•		-
SVB	•			•		•		•		•		•		VM
CLBVM	•			•		•						•		VM
LBVS	•									•		•	Data/Files	-
TSLB	•			•		•				•		•		VM
HF-1		•		•		•						•		Job
HF-2		•			•							•		Task
BRS		•		•		•						•		Job
AC		•		•		•						•		Job
TBSLB-PSO	•				•			•				•		Task
LBS-BF	•				•							•		Task
ACCLB		•		•								•		Job
OLB+LBMM	•					•				•	•			-
ED	•			•		•	•					•	Server	Session
Carton		•			•	•						•		-
CB		•			•	•		•				•		VM
TSPBRR	•				•						•			-
ESCE	•			•		•				•		•		-
CLBDM	•			•		•						•		Session and VM
MR		•												-
VMMP	•			•		•						•		-
FBRR	•			•		•						•		-
PALB	•			•		•						•		-
LBMM	•				•	•				•		•		Task
Min-Min	•				•	•				•	•			-
Max-Min	•				•	•				•	•			-
SOAS		•		•		•						•		Task
RoC-LB		•		•	•							•		Job

Таблица 7.3. Метрики, используемые в алгоритмах балансировки нагрузки

	Утилизация	Производительность	Накладные расходы	Время отклика	Масштабируемость	Отказоустойчивость	пропускная способность	Энергопотребление
VD	•							
WCAP	•	•			•	•		
VWR	•	•						
JIQ		•	•	•				
LFM		•					•	
SVB	•		•					
CLBVM	•	•		•			•	
LBVS		•		•	•	•		
TSLB	•	•		•				
HF-1		•			•		•	
HF-2				•		•		
BRS		•			•		•	
AC		•			•		•	
TBSLB-PSO		•						
LBS-BF		•		•				
ACCLB	•	•			•	•		
OLB +LBMM	•	•						
ED	•				•			
Carton	•	•	•					
CB	•		•					
TSPBRR	•			•				
ESCE	•			•				
CLBDM	•			•				
MR		•			•			
VMMP	•							
FBRR	•			•				
PALB	•							•
LBMM	•		•	•			•	
Min-Min	•		•	•			•	
Max-Min	•		•	•			•	
SOAS	•						•	
RoC-LB		•						

Чтобы понять основные характеристики алгоритмов балансировки нагрузки, проведен обзор последних достижений в области балансировки

нагрузки в распределенных вычислительных средах. Показано, что ни одна из этих работ не затрагивает непосредственно проблемное пространство рассматриваемой задачи; однако они составляют ценную основу для нашей работы.

7.4 Модель

Обработка вызова и доставка вызова – это два ключевых вопроса, которые определяют качество вызовов. Обработка вызовов сосредоточена на времени установления и разрыва связи, а также на преобразовании голосовой части вызовов в пакеты, передаваемые по сети. Адекватное качество звука является наиболее важным аспектом при обработке вызова.

7.4.1 Качество обслуживания

Во время обработки вызова кодек увеличивает используемую пропускную способность, но это менее существенно, чем то, что тот же кодек увеличивает загрузку процессора. В [172] показано, что потребляемая пропускная способность при 6 500 звонках в секунду не превышает 100 Мбит/с, а при 10 000 звонках не достигает 400 Мбит/с. В таблице 7.4 показана пропускная способность, используемая различными кодеками, учитывая, что VoIP-звонки используют аудио потоки для конечных точек (звонок между двумя сторонами будет использовать вдвое большую пропускную способность). Число звонков, поддерживаемых соединением 100 Мбит/с, составляет от 6 000 до 25 000 в зависимости от кодека звонков.

Однако число вызовов, обрабатываемых процессором, меньше, чем поддерживается соединением 100 Мбит/с., следовательно, обработка вызовов является ключевой характеристикой для обеспечения QoS. В данной работе предлагается ограничить загрузку процессора для обеспечения QoS.

Таблица 7.4. Пропускная способность кодека.

Кодек	**Bit Rate (kbps)**	**MOS**	**Пропускная способность (kbps)**
G.711	64	4.1	87.2
G.729	8	3.92	31.2
G.723.1	6.3	3.9	21.9
G.723.1	5.3	3.8	20.8
G.726	32	3.85	55.2
G.726	24	-	47.2
G.728	16	3.61	31.5
G722_64k	64	4.13	87.2
ilbc_mode_20	15.2	NA	38.4
ilbc_mode_30	13.33	NA	28.8

Вызовы оказывают различное влияние на загрузку процессора в зависимости от операций, выполняемых Asterisk. Если выполняются операции транскодирования, то загрузка процессора выше, чем, когда транскодирование не используется. В последнем случае Asterisk отвечает только за маршрутизацию вызова. Однако, в зависимости от кодека, нагрузка на процессор также меняется. В таблице 7.5 показана загрузка процессора для звонка без транскодирования [173].

Таблица 7.5. Загрузка процессора для звонков без транскодирования.

Протокол	**Кодек**	**10 Вызовов (%)**	**1 Вызов (%)**
SIP/RTP	G.711	2.36	0.236
SIP/RTP	G.726	2.13	0.213
SIP/RTP	GSM	2.58	0.258
SIP/RTP	LPC10	1.92	0.192

Производительность процессоров ATOM анализируется для VoIP с учетом числа звонков, использования, энергопотребления, обмена сообщениями с базой данных, регистрации и производительности звонков [104]. Видно, что процессор может обрабатывать от 70 до 500 звонков при 100% использовании.

7.4.2 Задержка времени запуска

Эластичность – это способность динамически получать или освобождать вычислительные ресурсы по требованию. Некоторые преимущества эластичности облака: избежание расточительной практики избыточного предоставления ресурсов, адаптация мощности к непредсказуемой рабочей нагрузке с течением времени и влияние на потребление энергии. Однако эластичность имеет смысл, если VMs могут быть запущены вовремя и использованы в течение ожидаемого пользователем времени.

Задержка времени запуска VM (StUp) – это время, необходимое для инициализации VM. Неожиданная задержка запуска VM может привести к снижению QoS и неполному предоставлению ресурсов. Изучение влияния задержек VM StUp на управление CVoIP необходимо для уменьшения факторов, влияющих на выполнение этого критичного ко времени приложения.

В [165] изучили задержки VM StUp в облаках и проанализировали взаимосвязь между StUp и различными факторами (размер образа ОС, тип экземпляра, расположение центра обработки данных и число экземпляров, приобретаемых одновременно). VM StUp – это период времени, который требуется облачным провайдерам, чтобы найти место в центрах обработки данных для создания VMs, выделения ресурсов (например, IP-адресов) для виртуальных машин и копирования/загрузки/конфигурирования образа ОС. Оно варьируется от 44 до 96 секунд для экземпляров Linux и от 429 до 810 секунд для экземпляров Windows на EC2 и Rackspace. У провайдера Azure время VM StUp варьируется от 356 до 406 секунд.

В [190] изучили зависимость между VM StUp, размером диска и содержимым. Авторы разработали и оценили подход для консолидации экземпляров VM. Они уменьшают размер диска и содержимое путем выбора необходимого набора сервисов для создания диска VM с минимально необходимым размером. Этот подход может быть применен ко всем образам VM, для которых пользователь может указать, какие пакеты программного обеспечения верхнего уровня необходимы для его желаемой функциональности.

В [132] измеряется скорость предоставления VM и время доступа к SSH среди облачных провайдеров, расположенных в разных регионах мира (Азия, Европа и США). В таблице 7.6 показаны StUp для нескольких ОС и провайдеров. В таблице 7.7 представлено среднее время создания VM и время доступа к ключу SSH среди облачных провайдеров.

Таблица 7.6. Среднее время запуска VM

Облако	**O.S.**	**StUp (sec.)**
EC2	Linux	96.9
	Windows	810.2
Azure	WebRole	374.8
	WorkedRole	406.2
	VMRole	356.6
Rackspace	Linux	44.2
	Windows	429.2

Таблица 7.7. Среднее время запуска VM с доступом по SSH.

Облако	**StUp (sec.)**
Google Cloud Platform	31
Amazon Web Services	47
Vexxhost	47
Linode	57
DigitalOcean	89
Rackspace	128
Azure	138

7.4.3 Определения

CVoIP инфраструктура состоит из m разнородных суперузлов $SNC_1, SNC_2, \dots, SNC_m$ с относительными скоростями $s_1, s_2, \dots, s_m$. Каждый SNC_i, для всех $i = 1, \dots, m$, состоит из m_i SNs. Каждый SN_k^i, для всех $k = 1, \dots, m_i$, запускает $k_i(t)$ VM в момент времени t (сек.). VMs одного SN идентичны и имеют одинаковую вычислительную мощность. VM_k описывается кортежем $\{st_k, size_k, stUp_k\}$, который состоит из периода начала аренды $st_k \geq 0$, задержки запуска $stUp_k$, вычислительной мощности $size_k$ в MIPS.

SNC содержит набор маршрутизаторов и коммутаторов, которые передают трафик между SN и внешним миром. Коммутатор соединяет точку перераспределения или вычислительные узлы. Соединения вычислительных узлов статичны, но их загрузка изменяется. Сеть межсетевого взаимодействия SNC является локальной. Взаимосвязь между SNC обеспечивается через публичный Интернет.

Рассмотрим n независимых вызовов $J_1, J_2, \dots, J_n$, которые должны быть спланированы на множестве SNC. Вызов J_j описывается кортежем $\{r_j, p_j, u_j\}$, который состоит из времени его освобождения $r_j \geq 0$, продолжительности p_j (время жизни) и вклада в загрузку процессора u_j из-за используемого кодека. Время освобождения вызова недоступно до его подачи, а его продолжительность неизвестна до завершения вызова. Загруженность процессора является константой для данного вызова, которая зависит от используемого кодека и вычислительной мощности VM.

Работа может быть назначена только на одну VM, репликация работ не рассматривается. Работа, размещенная на одной VM, не может быть перенесена на другую.

Чтобы оценить стоимость услуги провайдера для облачного решения, используется метрика, которая полезна для систем с VM, где традиционные

метрики становятся неактуальными. Для такого рода систем метрика должна позволять провайдеру измерять стоимость системы в терминах числа востребованных VM и времени их использования.

Определим модель стоимости, рассмотрев функцию, зависящую от числа арендуемых VM и времени их аренды. Обозначим количество расчетных часов в SNC_i через:

$$\bar{b}_i = \sum_{r=0}^{C_{max}/rp} k_i(r * rp) \tag{7.1}$$

где rp определяет минимальный период аренды VM, обычно $rp = 60$ мин. Общее количество расчетных часов во всех SNC определяется:

$$\bar{b} = \sum_{i=1}^{m} \bar{b}_i \tag{7.2}$$

В дополнение к $st_j, stUp_j, size_j$, VM характеризуется $vmu_j(t)$ использованием (нагрузкой) VM_j в момент времени t. Введем снижение качества как функцию использования VM:

$$\bar{q}_i = \sum_{j=1}^{\bar{b}_i} \sum_{t=st_j}^{ft_j} \gamma(vmu_j(t)) \tag{7.3}$$

где st_j и ft_j определяют время аренды VM_j и функцию пенализации $\gamma(\alpha) = ((\alpha - 0.7) * \overline{3.33})^2$, когда $\alpha > 0.7$ и 0 в противном случае.

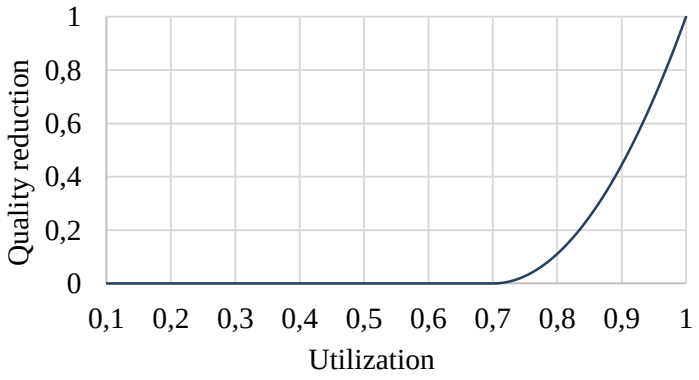


Рисунок 7.4. Снижение качества голоса в зависимости от загрузки процессора (utilization).

На рисунке 7.4 показано поведение снижения качества в зависимости от загрузки VM.

Общее снижение качества определяется следующим образом:

$$\bar{q} = \sum_{i=1}^{m} \bar{q}_i \tag{7.4}$$

Проанализируем число звонков, находящихся в режиме ожидания. Это происходит, когда VMs не могут обработать поступающие вызовы, поэтому вызовы ожидают в очереди свободной VM. Число вызовов, находящихся в режиме ожидания, определяется (5), где s_j – время инициирования J_j, а $\delta(\beta)$ равно 1, если $\beta > 0$, и 0 в противном случае.

$$\bar{c} = \sum_{j=1}^{n} \delta\left(s_j - r_j\right) \tag{7.5}$$

Эта задача рассматривается как частный случай динамической упаковки в контейнеры. Контейнеры представляют VM, а высота элементов определяет вклад вызова в использование VM.

Этот подход позволяет адаптироваться к неопределенностям облака, таким как динамическая эластичность, изменение производительности, виртуализация, свободная связь приложения с инфраструктурой, таким параметрам, как эффективная скорость процессора, число запущенных VMs и задержка времени запуска, а также многим другим.

В работе рассматривается многоцелевая задача, в которой для оптимизации работы алгоритмов используются три критерия: расчетные часы ($\bar{b}$), ухудшение качества ($\bar{q}$) и число задержанных звонков ($\bar{c}$). Хороший алгоритм балансировки нагрузки должен обеспечивать высокую производительность облака, минимизируя при этом ухудшение качества и число задержанных звонков.

Выбор метода для анализа многокритериальных задач не является тривиальным. Для решения такого рода проблем можно использовать несколько подходов.

Чтобы выбрать хорошую стратегию, проведем анализ, основанный на методе деградации. Она показывает, насколько метрика, генерируемая алгоритмами, приближается к наилучшему найденному решению.

Во-первых, оценивается деградация производительности каждой стратегии по сравнению с наилучшей стратегией. Затем эти значения усредняются по всем сценариям, и стратегии ранжируются. Лучшая стратегия с наименьшей средней деградацией имеет ранг 1.

Многокритериальная оптимизация не ограничивается поиском единственного решения данной задачи, как в случае применения подхода средней деградации, а набором решений, известным как Парето-оптимальное множество. Одно решение может представлять собой наилучшее решение в отношении стоимости, в то время как другое решение может быть наилучшим в отношении QoS. Цель состоит в том, чтобы выбрать наиболее подходящее решение и получить набор компромиссных решений, который представляет собой хорошее приближение к Парето-фронту.

Решение является оптимальным по Парето, если никакое другое решение не улучшает его с точки зрения всех критериев. Любое решение, не входящее во фронт, может считаться менее качественным, чем те, которые в него входят.

Один из формально-статистических методов использует метрику покрытия множества $SC(A,B)$, которая вычисляет долю решений в B, над которыми доминируют решения в A:

Кроме того, для оценки моделей прогнозирования используются среднеквадратичное отклонение (RMSD) и симметричная средняя абсолютная процентная ошибка (SMAPE). Они часто используются в статистике для оценки качества моделей прогнозирования [199, 198, 88]. RMSD и SMAPE определяются, соответственно, следующим образом:

$$\text{RMSD} = \sqrt{\frac{\sum_{i=1}^{n}(F(i) - A(i))^2}{n}} \tag{7.6}$$

$$SMAPE = \frac{\sum_{i=1}^{n}|F(i) - A(i)|}{\sum_{i=1}^{n}|A(i) + F(i)|} \tag{7.7}$$

где n – размер набора, $A(i)$ определяет фактическое значение, а $F(i)$ - прогнозируемое значение. Меньшие значения RMSD и SMAPE лучше.

7.5 Распределение вызовов

Задача распределения вызовов похожа на хорошо известную динамическую задачу упаковки в контейнеры, разновидность классической NP-трудной задачи оптимизации, имеющей высокую теоретическую значимость и практическое значение. Классическая задача упаковки в контейнер заключается в эффективном размещении элементов произвольной высоты в минимальное число контейнеров с фиксированной емкостью (в одномерном пространстве). Упаковка в контейнер является очень активной областью исследований в сообществах изучения алгоритмов и исследования операций.

В VoIP планировщик решает, будет ли звонок помещен в одну из доступных в данный момент VMs или должна быть запущена новая машина. Планировщик знает только вклад вызова в загрузку VM u_j. Все решения должны приниматься без знания о продолжительности вызова, скорости поступления вызовов и т. д.

Временное существование элементов является принципиальной новизной данной проблемы. Время жизни вызова и распределение определяют состояние виртуальных машин. В отличие от стандартной формулировки, контейнеры всегда открыты и динамичны, даже если они полностью заполнены. Элементы в контейнерах могут быть завершены (завершение вызова), а утилизация VM может

быть уменьшена в любой момент времени, тогда VM могут использовать свободное пространство для обработки новых вызовов.

Для изучения реалистичных сценариев с разнообразием гетерогенных сред определим три настраиваемых параметра:

Utilization Threshold (UT) — это параметр, который используется в качестве триггера для запроса новой VM, когда текущая утилизация достигает порогового значения. Известно, что процессор не может справиться с нагрузкой, когда загрузка достигает определенного уровня. Могут появиться джиттеры и прерывистый звук. UT зависит от характеристик процессора и позволяет избежать снижения качества голоса.

Порог аренды (RT) — это интервал времени, оставшийся до окончания периода аренды VM. Он ограничивает распределение вызовов на VM. Это позволяет избежать ситуации, когда вызов поступает незадолго до окончания времени аренды и завершается после окончания периода аренды, что приводит к продолжению аренды VM еще на один час, хотя другие VM свободны.

Интервал прогнозирования (англ., Prediction Interval – PI) — это интервал времени, на который выполняется прогнозирование. Предиктор выполняет оценку утилизации на каждом временном шаге (англ., Time Step – TS). Меньший TS обеспечивает лучшее предсказание, но увеличивает накладные расходы системы. Наоборот, больший TS уменьшает накладные расходы системы, но снижает точность предсказания.

Параметры могут быть настроены и динамически адаптированы к различным объективным предпочтениям, рабочим нагрузкам и свойствам облака. Этот подход позволяет адаптироваться к неопределенностям облака.

В первом сценарии $StUp = 0$. Предполагается, что планировщик использует идеальное прогнозирование, поэтому новые VM инициализируются точно в срок, чтобы немедленно начать обработку вызовов. Этот сценарий можно использовать как эталон идеального случая, когда обработка вызова является единственным фактором, влияющим на качество голоса.

Если $UT = 0.7$, система может гарантировать качество обслуживания. Следовательно, проблема сводится к одноцелевой задаче. На рисунке 7.5 показан пример распределения вызовов с гарантией QoS. Когда использование VM_1 превышает $UT = 0.7$, инициализируется VM_2.

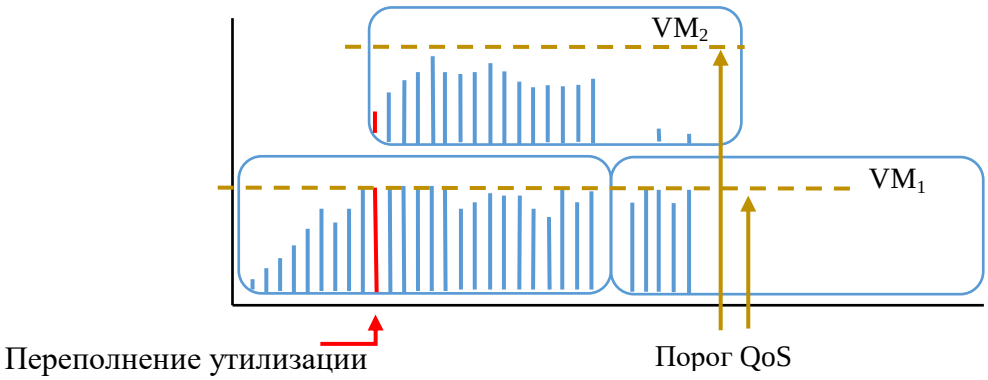


Рисунок 7.5. Распределение вызовов с гарантией QoS.

Если $UT = 1$, система не может гарантировать QoS. На рисунке 7.6 показан пример распределения вызовов. Когда использование VM_1 превышает допустимую загруженность, инициализируется VM_2 и начинает немедленно обрабатывать вызовы.

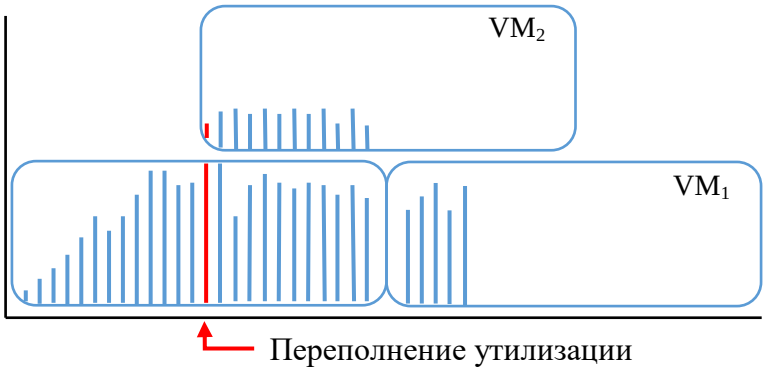


Рисунок 7.6. Распределение вызовов

Во втором сценарии $StUp \neq 0$, а, а $UT = 0.7$. На рисунке 7.7 показан пример распределения вызовов с задержкой времени запуска VM. VM_2запрашивается, когда использование VM_1 превышает UT. Во время VM_2 StUp,

VM_1 продолжает обработку вызовов с загрузкой, превышающей порог, снижая QoS. Наихудший случай возникает, когда VM_1 не имеет достаточно ресурсов для обработки поступающих вызовов. В этом случае система переводит вызовы в режим ожидания, ожидая доступных ресурсов.

В зависимости от типа и количества информации, используемой для распределения, стратегии распределения вызовов можно разделить на следующие категории: (1) Knowledge-Free (KF), без информации о вызовах и необходимых ресурсах; (2) Utilization-Aware (UA), с информацией об использовании VM; (3) Rental-Aware (RA) с известным временем запуска VM; и (4) Load-Aware (LA) с информацией о загрузке VM.

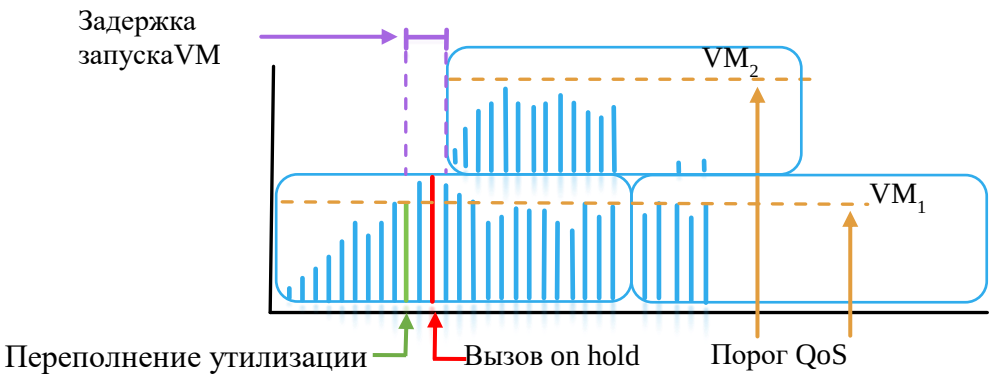


Рисунок 7.7. Распределение вызовов с задержкой времени запуска

Алгоритм 7.1 описывает стратегию BFit_RT, в которой поддерживаются два списка: Список допустимых голосовых узлов (AVNL) и список без AVNL (nAVNL). Список AVNL содержит BH с временем аренды, которое завершается не менее чем за RT минут. Оба списка отсортированы в порядке убывания их использования. Иногда используется термин VN вместо VM для согласованности с терминологией существующего распределителя вызовов.

В таблице 7.8 представлены стратегии распределения вызовов, которые запрашивают VM, когда поступающий вызов превышает UT. Кроме того,

предусмотрен механизм уменьшения числа активных VM. Стратегия Rental-Threshold ограничивает распределение вызовов за RT минут до того, как VM закончит аренду.

Algorithm 7.1. Best Fit with Rental-Threshold (BFit_RT)

Input: Voice node list (VNlist), UT, RT, and call.

Output: Allocation of call in one voice node.

```
1  vnIndex ← -1
2  Create AVNL and nAVNL lists with RT time.
3  Sort AVNL by utilization on decreasing order.
4  Sort nAVNL by utilization on decreasing order.
5  vnIndex ← Best_Fit(AVNL, UT, call)
6     if vnIndex < 0 then
7        vnIndex ←Best_Fit(nAVNL, UT, call)
8        if vnIndex < 0 then
9           vnIndex ←Best_Fit(nAVNL, 1.0, call)
10          if vnIndex < 0 then
11             vnIndex ←Best_Fit(AVNL, 1.0, call)
12 if vnIndex < 0 then
13    Send call to call_queue
14    Start a new node voice
15 else
16       Insert call into VN with index vnIndex
17 Endif
```

Таблица 7.8. Стратегии распределения вызовов.

Тип	Стратегия	Описание
KF	Rand	Распределяет работу *j* на VM случайным образом с использованием равномерного распределения.
	RR	Распределяет работу *j* на VM используя алгоритм Round Robin
RA	MaxFTFit	Распределяет работу *j* на VM со скорейшим окончанием
	MidFTFit	Распределяет работу *j* на VM с кратчайшим временем до половины времени аренды.
	MinFTFit	Распределяет работу *j* на VM с самым близким временем окончания
UA	FFit	Распределяет работу *j* на первую VM, способную выполнить ее.
	BFit	Распределяет работу *j* на VM с наименьшей оставшейся утилизацией.
	WFit	Распределяет работу *j* на VM с наибольшей оставшейся утилизацией
KF + RT	Rand_05, 10, 15 RR_05, 10, 15	Распределяет работу *j* на VM с RT =5, 10 и 15 мин используя Rand и RR
RA + RT	MaxFTFit_05, 10, 15 MidFTFit_05, 10, 15 MinFTFit_05, 10, 15	Распределяет работу *j* на VM с RT = 5, 10 и 15 мин используя MaxFTFit, MidFTFit, и MinFTFit
UA + RT	BFit_05, 10, 15 FFit_05, 10, 15 WFit_05, 10, 15	Распределяет работу *j* на VM с RT = 5, 10 и 15 мин используя BFit, FFit, и WFit

В таблице 7.9 описаны стратегии распределения вызовов с прогнозированием нагрузки. При каждом заданном TS стратегия оценивает загрузку для следующего PI, и, если она превышает UT, запрашивает дополнительную VM.

Таблица 7.9. Стратегии распределения вызовов с прогнозированием.

Тип	Стратегия	Описание
LA	Rand_StUp Rand_s10, 20, 30	Распределяет работу *j* на VM используя Rand, и RR. PI = 10, 20, 30 и StUp сек
	RR_StUp RR_s10, 20, 30	
RA + LA	MaxFTFit_StUp MaxFTFit_s10, 20, 30	Распределяет работу *j* на VM используя MaxFTFit, MidFTFit, и MinFTFit. PI = to 10, 20, 30 и StUp сек.
	MidFTFit_StUp MidFTFit_s10, 20, 30	
	MinFTFit_StUp MinFTFit_s10, 201, 30	
UA + LA	BFit_StUp BFit_s10, 20, 30	Распределяет работу *j* на VM используя BFit, FFit, и WFit. PI = 10, 20, 30 и StUp сек
	FFit_StUp FFit_s10, 20, 30	
	WFit_StUp WFit_s10, 20, 30	

7.6 Стратегии распределения вызовов с прогнозированием нагрузки

Адекватная оценка нагрузки может помочь повысить эффективность распределения вызовов. В лучшем случае система может предсказать необходимость в дополнительных VMs точно за StUp времени, чтобы иметь VM точно в срок, когда это необходимо, чтобы избежать снижения качества обслуживания и увеличения стоимости.

Ранее было описано, как временная задержка StUp влияет на распределение вызовов (см. рисунок 7.7). На рисунке 7.8 показан пример прогнозирования нагрузки, балансировщик оценивает нагрузку для следующего PI (линии с черными точками), и если прогноз превышает UT (линии с красными точками), то балансировщик запускает новые VMs до того, как поступающие вызовы ухудшат

QoS. Все модели прогнозирования используются для запроса новых VMs. Они позволяют балансировщику оценить число VMs на каждом TS.

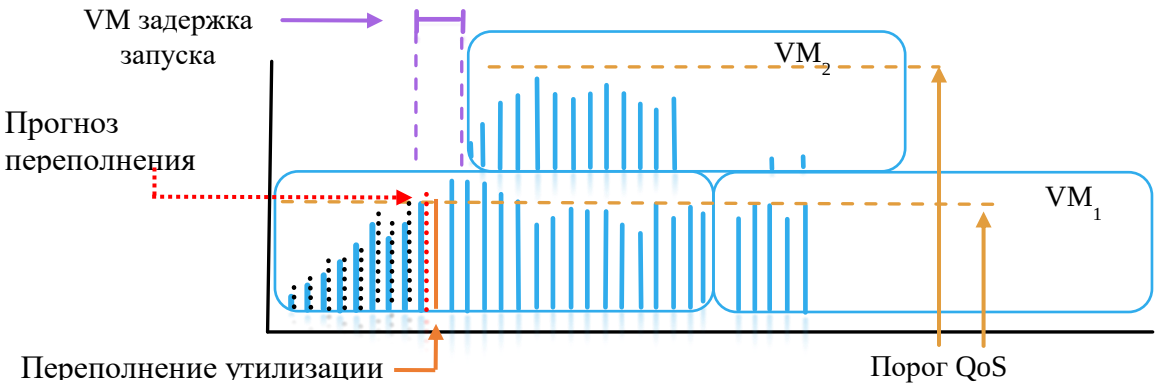


Рисунок 7.8. Распределение вызовов с предсказанием и задержкой времени запуска.

7.6.1 Стратегия Rate of Change

Rate of Change - RoC [90] — это динамический децентрализованный алгоритм распределенной балансировки нагрузки, который достигает цели минимизации времени простоя процессора и недопустимо высоких накладных расходов на перезагруженность. Для балансировки он рассчитывает скорость изменения нагрузки между двумя TS.

TS является адаптивным параметром, и его длина может меняться. RoC вычисляет разницу в нагрузке (Δ) и использует ее в качестве оценки нагрузки для PI. Δ позволяет распределять вызовы более эффективно. Более тонкая выборка позволяет улучшить баланс системы, но увеличивает накладные расходы.

Используется понятие Δ как механизм прогнозирования запросов на новые VM, которые могут быть предоставлены после времени StUp. Δ используется для оценки числа VMs для каждого PI. Это позволяет инициализировать VM до того, как поступающие вызовы ухудшат QoS.

Пусть $u_i(t)$ – использование SNC_i в момент времени t, а $k_i(t)$- число работающих VMs, тогда скорость изменения нагрузки за период TS определяется как:

$$\Delta_i(t) = (u_i(t) - u_i(t - TS))/k_i(t) \quad (7.8)$$

Будущую нагрузку системы оцениваем как:

$$u_i(t + PI) = u_i(t) + \Delta_i(t) \quad (7.9)$$

На рисунке 7.9 показан сценарий прогнозирования RoC, сплошные линии представляют реальное использование, а пунктирные линии – расчетное использование. Момент времени для предсказания определяется как $T_{i+1} = T_i + TS$. Если загрузка превышает UT, то система запрашивает новую VM. UT – это регулируемый параметр, который зависит от порога использования, чтобы гарантировать QoS.

Например, в момент времени T_4 система запускает новую VM на основе прогнозируемого использования для T_5 (прогноз C’). Это пример неправильного избыточного предоставления ресурсов. В момент времени T_5 система прогнозирует использование в T_6 меньше, чем UT (прогноз D’), и не запрашивает новую VM. Однако реальная загрузка превышает UT, что приводит к ухудшению качества обслуживания (under-provisioning). Наконец, в момент времени T_9 система прогнозирует загрузку в T_{10} больше, чем UT (прогноз H’), и запрашивает новую VM. Это пример адекватного предоставления VM.

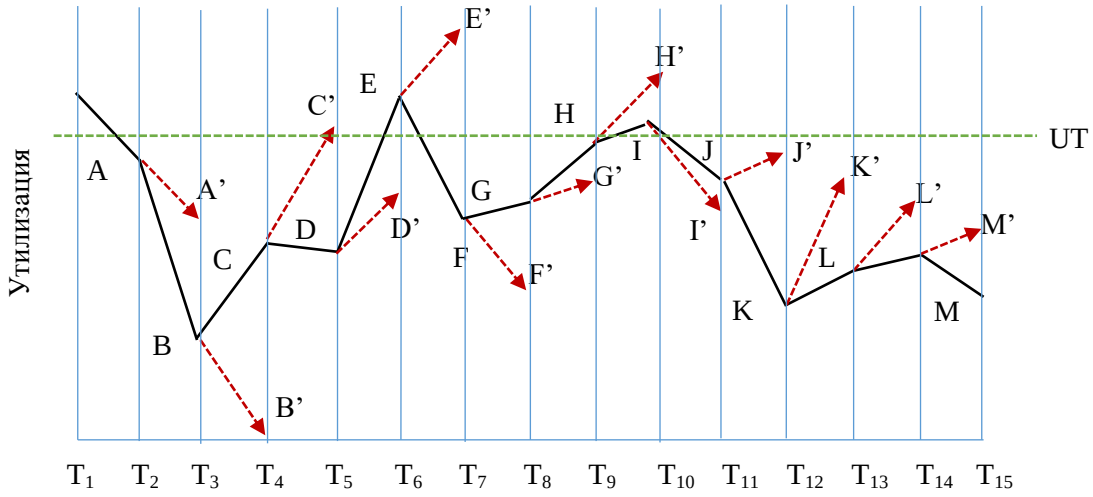


Рисунок 7.9. Прогнозирование RoC.

7.6.2 Нейронные сети

Нейронные сети (НС) (англ. NN “ neural network ") широко используются для прогнозирования, классификации или управления, распознавания объектов и т. д. На рисунке 7.10 показана архитектура НС для прогнозирования вызовов. Узлы представляют собой искусственные нейроны, а стрелки характеризуют связи между ними. Взаимосвязанная группа узлов (выход одного нейрона может быть входом другого) образует сеть.

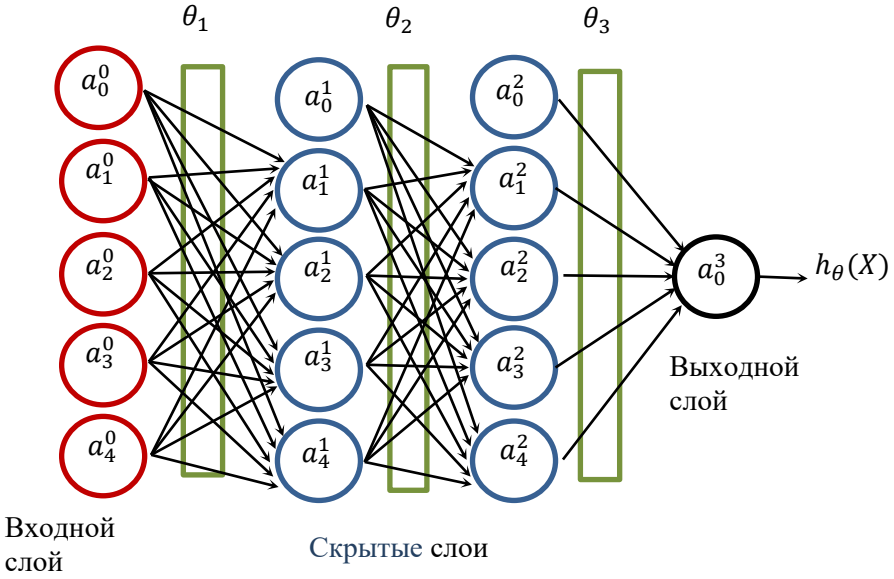


Рисунок 7.10. Архитектура нейронной сети

Входными данными для нейронной сети являются $X = [a_0^0, a_1^0, a_2^0, a_3^0, a_4^0]$ – весовые коэффициенты между слоями 1, 2 и 3 соответственно, а $h_\theta(X) = a_0^3$ – выход. В нашем случае $a_0^3 = u_i\ (t + PI)$. Входы a_1^0, a_2^0, a_3^0 и a_4^0 определяются следующим образом: $a_1^0 = u_i(t)$, $a_2^0 = u_i(t - TS)$, $a_3^0 = u_i(t - 2 * TS)$ и $a_4^0 = t$.

Элементы a_0^0, a_0^1, и a_0^2 определяют смещение на каждом слое. Нейронная сеть рассматривает сигмоидную (логистическую) функцию активации, определяемую (7.12).

Активация блока i (для $i > 0$) в слое j (для $j > 0$) определяется (7.13), где lon(θ_i) – количество весов на θ_i.

$$g(x) = \frac{1}{1 + e^{-z}} \tag{7.10}$$

$$a_j^i = g\left(\sum_{j=0}^{lon(\theta_i)} a_j^{i-1} * \theta_{i,j}\right) \tag{7.11}$$

Оценочная функция определяется в (7.14). Первый член — это сумма логарифмических ошибок. Для обучающего множества $T = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$, где m – количество элементов на T, $y^{(i)}$ – значение результата i-го элемента, а $\mathrm{h}_\theta(x^{(i)})$ – предсказанное значение входов $x^{(i)}$. Второй член — это член регуляризации, где λ управляет относительной важностью двух членов, а L – это номер слоя в нейронной сети, см. (7.15).

$$J(T) = \frac{-1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(\mathrm{h}_\theta(x^{(i)})\right) + \left(1 - y^{(i)}\right) \log\left(1 - \mathrm{h}_\theta(x^{(i)})\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{L-1} \theta_j^2 \tag{7.12}$$

$$\theta_i^2 = \sum_{j=1}^{lon(\theta_i)} (\theta_{i,j})^2 \tag{7.13}$$

Целью является обучение НС и минимизация $J(\theta)$ для получения лучших значений для $\theta_1, \theta_2,$ и θ_3, затем можно использовать эти значения для прогнозирования поступающих вызовов. Алгоритм 7.2 показывает шаги обучения нейронной сети, для обратного распространения используется градиентный спуск.

Algorithm 7.2. Обучение нейронной сети

Input: Training set *T*, number of iteration (iter).

Output: Values of θ_1, θ_2, and θ_3.

```
1   Initialize θ1, θ2, θ3 and J(θ)
2   Normalization of T.
3   For j ← 1 to 30
4      Random initialization of δ1,δ2, and δ3
5      For i ← 1 to iter.
6         For each xi into T
7            Compute hθ(xi)
8         Endfor
9         Compute J(T)
10        Apply backpropagation
11        Update δ1, δ2, and δ3
12     Endfor
13     If J(θ) > J(T)
14        J(θ) ← J(T)
15        θ1 ← δ1, θ2 ← δ2, θ3 ← δ3
16     Endif
17  Endfor
```

7.6.3 Прогнозирование на основе истории

Для сравнения эффективности ранее описанных прогнозирующих методов, был разработан прогнозирующий метод, который учитывает фактическое использование и историческую информацию об использовании ресурсов. В

момент времени t исторический предиктор оценивает $u_i\ (t + PI)$ на основе: $u_i\ (t)$ и $\widehat{u_i}(t + PI)$, где $\widehat{u_i}(t)$ определяет историческое стандартное отклонение использования в момент времени t. Будущая нагрузка системы оценивается как:

$$u_i\ (t + PI) = u_i(t) \pm \left(x * \widehat{u_i}(t + PI)\right), \tag{7.14}$$

где x - одно равномерно распределенное случайное число в интервале [0, 1]. Кроме того, генерируется случайное значение, определяющее, увеличивается или уменьшается утилизация (±).

На рисунке 7.11 показан пример утилизации за 85 дней с интервалом в 315 секунд.

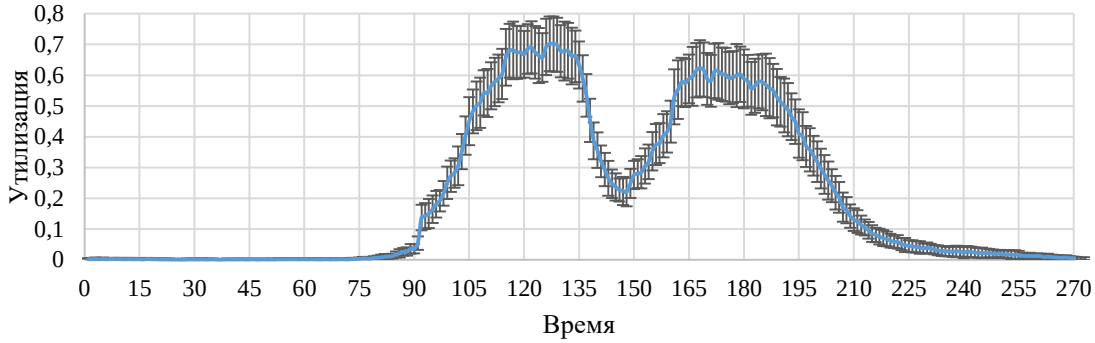


Рисунок 7.11. Прогнозирование на основе истории при 315 секундах TS.

7.7 Экспериментальные параметры

Для проведения экспериментов использовался стандартный трассировочный симулятор CloudSim [89], расширенный нашими алгоритмами, которые поддерживают динамическое поступление вызовов, задержки при запуске VM, статистический анализ и модели прогнозирования.

7.7.1 Рабочая нагрузка

В работе используются трассы реального VoIP-сервиса [198], которые включают телефонные звонки со следующей информацией: индекс звонка, ID

пользователя, совершающего звонок, IP локального телефона, пункт назначения звонка, название страны назначения, поставщик телекоммуникационных услуг, начало звонка (временная метка), продолжительность звонка (в секундах), продолжительность платного звонка, стоимость минуты и т.д. Число звонков в день и продолжительность звонков представлены в Таблице 7.10 и Таблице 7.11.

Таблица 7.10. Число вызовов в день

День	Всего	Среднее
Понедельник	131.443	21.906
Вторник	129.379	21.563
Среда	131.460	21.910
Четверг	130.439	21.739
Пятница	120.999	20.166

Таблица 7.11. Продолжительность звонков

Время (мин)	Число вызовов
0 - 1	310.602
1 - 2	136.211
2 - 3	68.988
3 - 4	39.392
4 - 5	23.397
…	…
19 - 20	721

В первом сценарии используется 24 рабочие нагрузки; каждая рабочая нагрузка включает телефонные звонки, сделанные в течение одной недели. На рисунке 7.12 показано среднее количество звонков в день по часам в течение 24 недель, гистограмма звонков в час в течение дня показывает типичное поведение для бизнес-клиентов, два пика с 10 до 12 и между 14 и 16 часами.

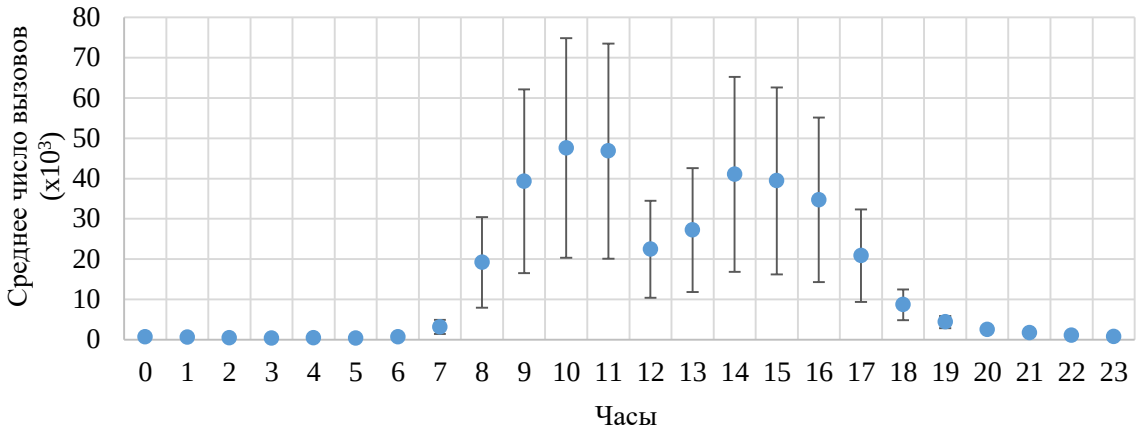


Рисунок 7.12. Среднее число звонков в день по часам в течение 24 недель со стандартным отклонением.

В выходные дни уровень рабочей нагрузки низкий, поэтому CVoIP требуется только одна виртуальная машина для обработки всех звонков. Для данного исследования были исключены выходные дни, т. к. их можно заменить 24 расчетными часами в день. Одна VM всегда работает, даже если в системе нет звонков.

Во втором сценарии используются 30 рабочих нагрузок с телефонными звонками, сделанными в течение одного дня, остальные рабочие нагрузки используются для обучения NN и являются основой для прогнозирования на основе исторических данных. Как и в предыдущем сценарии, рабочая нагрузка учитывает только будние дни.

7.7.2 Параметры прогнозирования

Правило прогнозирования является основополагающим для оценки числа VM, необходимых для предоставления услуги в CVoIP. В следующих разделах описывается настройка RoC и правила прогнозирования нейронной сети.

7.7.2.1 Настройка параметров RoC

RoC использует линейное правило, которое учитывает только разницу нагрузки между двумя временными кадрами, см. формулу 7.9. Целью настройки является выбор адекватного $\Delta_i(t)$ для получения лучшего прогноза. Было установлено, что система CVoIP более уязвима, когда число виртуальных машин невелико, поэтому предлагается правило прогнозирования, которое учитывает число виртуальных машин, работающих в системе, оно определяется следующим образом:

$$\Delta_i(t) = (u_i(t) - u_i(t - Si))/k_i(t), \tag{7.15}$$

где $k_i(t)$ описывает число работающих виртуальных машин. Когда $k_i(t)$ велико, система может лучше реагировать на резкие изменения нагрузки, поскольку у нее больше ресурсов для поддержки большого числа поступающих вызовов. Оценим ее производительность с помощью моделирования.

Сравним два правила прогнозирования для стратегий RoC. Чтобы оценить их эффективность, используем семь StUp: 45, 90, 135, 180, 225, 270 и 315 секунд, и четыре TS: 10, 20, 30 и StUp секунд в качестве тестовых случаев.

Таблица 7.12 и Таблица 7.13 показывают значения RMSD для обоих правил предсказания, nRoC превосходит RoC во всех тестовых случаях. Наихудшее значение наблюдается при прогнозировании с PS и StUp, равными 315. Аналогично значениям RMSD, nRoC превосходит RoC во всех тестовых случаях для значений SMAPE, см. таблицу 7.14 и таблицу 7.15.

Таблица 7.12. Значения RMSD для nRoC.

StUp \ PS	10	20	30	StUp
45	14.067	19.278	22.892	26.637
90	**13.991**	19.084	22.907	33.773
135	14.047	19.180	22.883	38.417
180	14.006	19.155	22.754	41.358
225	14.145	19.227	22.869	44.672
270	14.405	19.462	23.010	46.337
315	14.552	19.553	22.849	**49.615**

Таблица 7.13. Значения RMSD для RoC.

StUp \ PS	10	20	30	StUp
45	41.028	56.491	66.746	78.607
90	**41.007**	56.469	66.746	98.481
135	41.015	56.416	66.759	111.995
180	41.022	56.460	66.759	118.451
225	41.139	56.423	66.764	127.266
270	41.051	56.431	66.764	131.595
315	41.216	56.555	66.799	**138.387**

Таблица 7.14. Значения SMAPE для nRoC ($x10^{-2}$).

StUp \ PS	10	20	30	StUp
45	0.693	0.997	1.195	1.421
90	**0.690**	0.988	1.195	1.817
135	0.692	0.994	1.193	2.060
180	0.691	0.992	1.187	2.202
225	0.698	0.997	1.192	2.369
270	0.694	0.999	1.192	2.451
315	0.693	1.000	1.180	**2.602**

Таблица 7.15. Значения SMAPE для RoC (x10^{-2}).

StUp \ PS	10	20	30	StUp
45	1.949	2.732	3.243	3.843
90	**1.948**	2.732	3.244	4.844
135	1.949	2.731	3.244	5.484
180	1.949	2.732	3.244	5.821
225	1.950	2.732	3.244	6.255
270	1.950	2.733	3.244	6.467
315	1.952	2.734	3.245	**6.810**

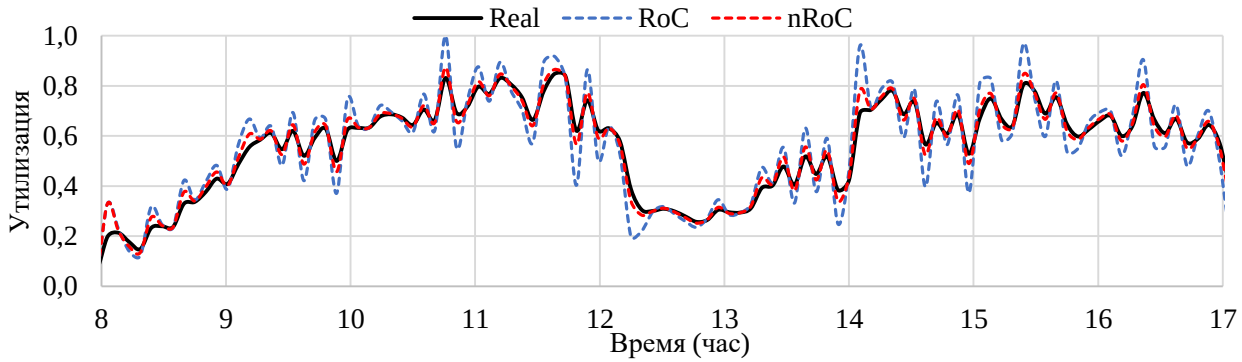


Рисунок 7.13. Оценка нагрузки с использованием 315 сек. периода предсказания

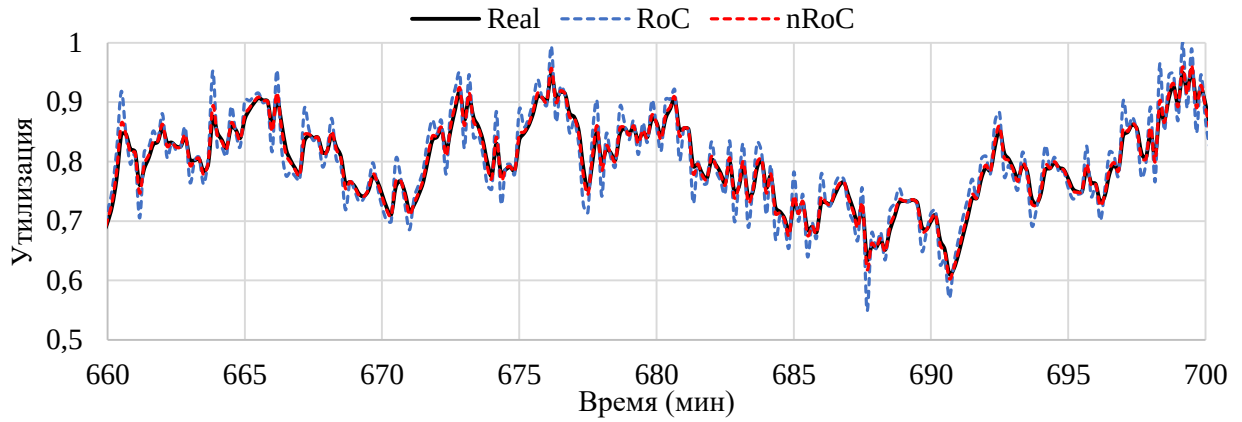


Рисунок 7.14. Оценка нагрузки с использованием 10 сек. периода предсказания.

На рисунке 7.13 показан пример реальных и прогнозируемых значений нагрузки за 9 часов с периодом прогнозирования 315 сек. Аналогично, на рисунке 7.14 представлены те же значения для 40 минут с периодом прогнозирования 10

сек. В обоих примерах предсказание nRoC больше соответствует реальным значениям нагрузки.

Теперь проанализируем поведение стратегий назначения с обоими правилами предсказания.

В таблице 7.16 показаны значения SC(A,B) для доминирования стратегии RoC над nRoC в четырех временных интервалах предсказания. Видно, что RoC доминирует над решениями nRoC в диапазоне 8.3 - 37.5%, в среднем 22.2%.

В таблице 7.17 приведены значения SC(B,A) для доминирования стратегии nRoC над RoC. Видно, что nRoC доминирует над решениями RoC в диапазоне 67.5 - 99.2%, в среднем 84.3%.

Доминирование SC(B,A) подтверждает, что nRoC может генерировать решение в среднем на 84.3% лучше или, по крайней мере, с тем же качеством, что и RoC, для различных времен запуска и интервалов предсказания.

Таблица 7.16. Покрытие множеств nRoC.

StUp / Стратегия	45	90	135	180	225	270	315
BFit	0.908	0.958	0.933	0.933	0.933	0.883	0.858
FFit	0.967	0.983	**0.992**	0.950	0.917	0.900	0.908
MaxFTFit	0.958	0.875	0.942	0.917	0.883	0.833	0.858
MidFTFit	0.867	0.817	0.900	0.792	0.850	0.825	0.842
MinFTFit	0.792	0.858	0.775	0.733	**0.675**	0.700	0.725
Rand	0.775	0.858	0.800	0.833	0.775	0.742	0.742
RR	0.792	0.783	0.842	0.792	0.833	0.800	0.700
WFit	0.775	0.833	0.883	0.833	0.825	0.808	0.750

Таблица 7.17. Покрытие множеств RoC

Стратегия \ StUp	45	90	135	180	225	270	315
BFit	0.225	0.142	0.117	0.125	0.125	0.192	0.167
FFit	0.133	**0.083**	0.117	0.117	0.158	0.175	0.117
MaxFTFit	0.142	0.217	0.192	0.108	0.158	0.183	0.100
MidFTFit	0.192	0.242	0.158	0.258	0.217	0.175	0.167
MinFTFit	0.325	0.225	0.308	0.350	**0.375**	0.333	0.167
Rand	0.333	0.208	0.292	0.258	0.300	0.300	0.300
RR	0.250	0.283	0.233	0.333	0.225	0.283	0.358
WFit	0.308	0.267	0.208	0.283	0.250	0.233	0.317

7.7.2.2 Настройка нейронной сети

Для того чтобы найти наилучшую версию нейронной сети (НС), необходимо настроить несколько параметров. Во-первых, следует найти адекватное значение λ. НС использует утилизацию трех TS для прогнозирования. Следовательно, рассмотрим $u_i(t)$, $u_i(t - TS)$ и $u_i(t - 2 * TS)$ для оценки $u_i(t + PI)$.

Оценим алгоритмы при $PI = TS = stUp = 315$ секунд. Как было сказано ранее, в лучшем случае дополнительные VM принимают вызовы именно тогда, когда это необходимо. Значения θ_1, θ_2 и θ_3 были вычислены для каждой модели со следующей конфигурацией: λ ={10.0,1.0,0.5,0.1,0.01,0.001,0.0001, 0.00001}. НС для всех конфигураций тренировалась с 15 днями нагрузки и проверялась с 15 днями нагрузки (10 новых дней нагрузки, и остальные 5 дней являются частью 15 дней тренировки). Для получения наилучших значений θ_1, θ_2 и θ_3 процесс обучения был выполнен 30 раз.

В таблице 7.18 показаны вычисленные значения RMSD и SMAPE для различных значений λ, лучшими решениями являются алгоритмы с малыми значениями λ для обеих метрик.

Таблица 7.18. RMSD и SMAPE для значений λ.

λ	RMSD	SMAPE
0.00001	**106.739660**	**0.055020**
0.0001	107.239957	0.055468
0.001	108.844112	0.056072
0.01	118.981582	0.060927
0.1	120.277697	0.062772
0.5	130.424748	0.072682
1.0	138.546018	0.079169
10.0	167.208843	0.10274

НС использует утилизацию трех ТС для прогнозирования PI. Для обучения НС рассматриваются три различных значения: $u_i(t)$ – утилизация в момент времени t, $u_i(t) = \sum_{k=t-TS}^{t} \frac{u_i(k)}{TS}$ – средняя утилизация в интервале $[t - TS,\ t]$, и $u_i(t) = \max[u_i(t - TS), u_i(t)]$ – максимальная утилизация в интервале $[t - TS, t]$.

Оценим три версии НС с предыдущими конфигурациями и наименьшим значением λ ($\lambda = 0{,}00001$). В конце выберем лучшую модель для использования в качестве предиктора вызовов.

В таблице 7.19 показаны значения RMSD и SMAPE, при использовании различных значений загрузки VM для предсказания: текущей, средней и максимальной. Видно, что лучшей версией является НС, которая учитывает текущую утилизацию в момент времени t.

Таблица 7.19. RMSD и SMAPE для значений утилизации.

Утилизация	RMSD	SMAPE
Реальная	**106.739660**	**0.055020**
Средняя	111.681136	0.056846
Максимальная	110.478730	0.057947

Наконец, проанализируем различные тренировочные наборы. Для этого варьируем количество дней в тренировочном наборе и регулируем результирующие значения θ_1, θ_2 и θ_3 в предикторе.

Рассматриваются четыре набора с 30, 45, 60 и 75 рабочими нагрузками для создания обучающего набора. Обучающие наборы содержат 15 рабочих нагрузок из тестового набора, 5 рабочих нагрузок из проверочного набора и остальные – новые рабочие нагрузки. Проверочный набор содержит 5 рабочих нагрузок из тренировочного набора и 10 новых рабочих нагрузок. Тестовое множество содержит 30 рабочих нагрузок, 15 рабочих нагрузок из обучающего множества и 15 новых рабочих нагрузок.

В таблице 7.20 показаны значения RMSD и SMAPE для различных версий обучающего набора. Видно, что лучшей версией НС является обучение с 75 днями рабочей нагрузки и $\lambda = 0.00001$.

Таблица 7.20. RMSD и SMAPE для обучающих наборов различной длины.

Число обучающих наборов	RMSD	SMAPE
30	115.679943	0.054781
45	116.057055	0.055013
60	115.563836	0.054721
75	**115.026791**	**0.054598**

7.8 Экспериментальный анализ

VoIP-провайдеры предоставляют VM на почасовой основе. Когда время аренды VM заканчивается, VM может быть выключена только в том случае, если VM не обрабатывает вызовы. В любом другом случае VM продолжает работать еще в течение часа. В первом сценарии анализируются стратегии с нулевым StUp, что похоже на идеальное предсказание нагрузки, и изучается поведение стратегий распределения при двух UT, с 0.7 и 1.

7.8.1 Первый сценарий

В таблице 7.21 представлены параметры моделирования для первого сценария.

Таблица 7.21. Настройка моделирования для первого сценария.

Имя	Значение
Super Node Cluster	1
Число стратегий:	20
Число рабочих нагрузок	24
Нагрузка	168h
Число кодеков	4
Число вызовов	2,526,595

В таблице 7.22 представлены средние значения $\bar{b}$ в течение 24 недель для обоих UT. Для UT = 0.7 лучшими стратегиями являются BFit/FFit, а худшими – WFit/MinFTFit. RR_05, RR_10, RR_15 имеют лучшую производительность, чем RR. Аналогично, WFit_05, WFit_10 и WFit_15 лучше, чем WFit. Разница между лучшей стратегией BFit и худшей (MinFTFit) составляет в среднем около 111 расчетных часов в неделю. Кроме того, 17 стратегий улучшают производительность Round Robin, используемого в настоящее время для VoIP-

сервиса. Снижение $\bar{b}$ составляет от 25.96% до 1.22%, и только две стратегии имеют худшую производительность.

Таблица 7.22. Средние недельные затраты на оплату счетов для различных UTs.

Стратегия \ UT	0.7	1.0
BFit	252.08	200.35
FFit	252.42	200.79
MaxFTFit	254.71	204.12
FFit_05	259.46	206.79
FFit_15	261.75	209.28
FFit_10	261.79	208.95
BFit_05	266.13	213.69
BFit_15	269.21	216.22
BFit_10	273.25	217.16
MidFTFit	276.08	221.23
RR_15	279.79	223.97
WFit_15	283.79	227.25
RR_10	289.00	231.59
WFit_10	290.42	232.93
RR_05	306.88	245.84
WFit_05	311.29	248.98
Rand	336.29	263.96
RR	340.46	266.13
WFit	351.08	272.12
MinFTFit	363.96	279.83
Среднее	288.99	229.56

Проведем совместный анализ двух метрик в соответствии с методикой средней деградации и получим набор компромиссных решений, которые представляют собой хорошее приближение к Парето-фронту.

В таблице 7.23 представлены $\bar{b}$ деградация, $\bar{q}$ деградация и их средние значения. Последние четыре столбца таблицы содержат ранжирование каждой

стратегии по отношению к стоимости, качеству и их средним значениям. Ранг $\bar{b}$ основан на деградации $\bar{b}$. Ранг $\bar{q}$ основан на деградации $\bar{q}$. Средний ранг – основан на усреднении двух рангов. Ранг – это относительная позиция по отношению ко всем стратегиям.

Таблица 7.23. Деградация и ранжирование.

Стратегия	$\bar{b}$	$\bar{q}$	Среднее	Ранг $\bar{b}$	Ранг $\bar{q}$	Средний Ранг	Ранг
BFit	0.263	21.099	10.681	**1**	20	6	4
BFit_05	6.911	17.930	12.420	7	16	12	6
BFit_10	8.405	17.342	12.874	9	13	15	5
BFit_15	8.091	17.240	12.665	8	12	14	**3**
FFit	0.535	21.031	10.783	**2**	19	7	4
FFit_05	3.483	18.758	11.120	4	18	8	5
FFit_10	4.638	18.069	11.354	5	17	10	5
FFit_15	4.812	17.819	11.315	6	14	9	**3**
MaxFTFit	2.096	17.835	9.965	**3**	15	4	**1**
MidFTFit	10.536	16.442	13.489	10	11	16	4
MinFTFit	39.147	12.800	25.973	20	10	20	7
Rand	31.263	1.094	16.178	17	4	17	4
RR	32.681	0.732	16.707	18	**2**	18	**3**
RR_05	22.542	2.144	12.343	15	5	11	**3**
RR_10	15.388	4.666	10.027	13	7	5	**3**
RR_15	11.772	6.980	9.376	11	9	**2**	**3**
WFit	35.435	0.000	17.718	19	**1**	19	**3**
WFit_05	23.904	1.085	12.494	16	**3**	13	**2**
WFit_10	16.606	3.301	9.954	14	6	**3**	**3**
WFit_15	13.292	5.428	9.360	12	8	**1**	**3**

BFit, распределяющий вызовы на основе стратегии Best Fit, является лучшей стратегией для $\bar{b}$. Однако это наихудшая стратегия для $\bar{q}$. Она имеет тенденцию к увеличению утилизации и снижению качества. Лучшей стратегией для $\bar{q}$ является WFit, она имеет тенденцию к недоиспользованию виртуальных машин, сохраняя качество, но увеличивая число виртуальных машин и стоимость аренды.

Стратегия с наилучшим компромиссом между двумя метриками – WFit_15. Она распределяет вызовы с использованием стратегии Worst Fit и ограничивает

распределение вызовов за 15 минут до того, как VM достигнет своего времени аренды.

Для решения общей двухкритериальной задачи мы хотим получить набор компромиссных решений, которые представляют собой хорошее приближение к Парето-фронту. Формально это не Парето-фронт, поскольку исчерпывающий поиск всех возможных решений не проводится, а скорее служит практическим приближением к Парето-фронту.

На рисунке 7.15 показаны наборы решений для двадцати стратегий, полученных на основе 109 дней рабочей нагрузки. Это двумерное пространство решений представляет собой выполнимое множество решений, удовлетворяющих ограничениям задачи. Пространство решений охватывает диапазон значений $\bar{b}$ от 0 до 0.65, в то время как значения $\bar{q}$ находятся в диапазоне от 0 до 0.26.

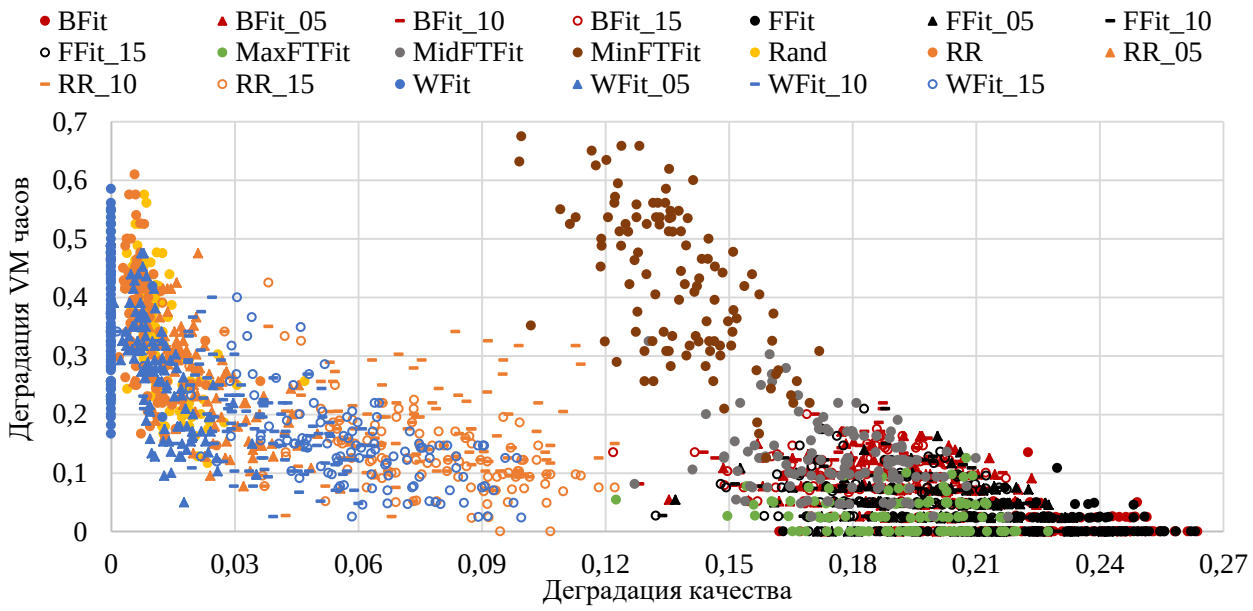


Рисунок 7.15. Пространство решений.

Видно, что пространство решений разделено на три группы, расположенные в правой нижней части, левой верхней части и посередине. BFit, FFit и MaxFTFit расположены в правой нижней части, являясь одними из лучших решений с точки зрения $\bar{b}$. Они превосходят другие стратегии, такие как RR, которые в настоящее время используются для VoIP-сервиса. WFit находится в левой части, являясь

одним из лучших решений с точки зрения $\bar{q}$. Три версии WFit с различными RT (WFit_05, WFit_10, WFit_15) имеют хорошее поведение.

WFit является лучшей для $\bar{q}$ деградации. Диапазон деградации $\bar{b}$ составляет от 0.16 до 0.56. WFit_05 увеличивает деградацию $\bar{q}$ до 0.017, но уменьшает $\bar{b}$ до 0.05. Для WFit_10 деградация $\bar{q}$ повышается с 0.017 до 0.06, но $\bar{b}$ снижается до 0.023.

Наконец, WFit_15 имеет широкий диапазон решений для $\bar{q}$ деградации (от 0.019 до 0.099), но только 20% их решений превышают 20% $\bar{b}$ деградации. Варианты WFit охватывают различные сектора на Парето-фронте, и они показывают наилучший компромисс между обеими критериями для двадцати стратегий. Пространство решений MaxFTFit находится в том же диапазоне стоимости, что и BFit и FFit. Оно перекрывает $\bar{q}$ обеих стратегий.

Стратегии WFit_05, WFit_10, WFit_15 и MaxFTFit лучше охватывают пространство решений и Парето-фронт. Они являются хорошими вариантами для VoIP-провайдеров. На рисунке 7.16 показаны двадцать приближений Парето-фронтов, созданных исследуемыми стратегиями.

Два набора решений без доминирования можно сравнить с помощью метрики покрытия множества.

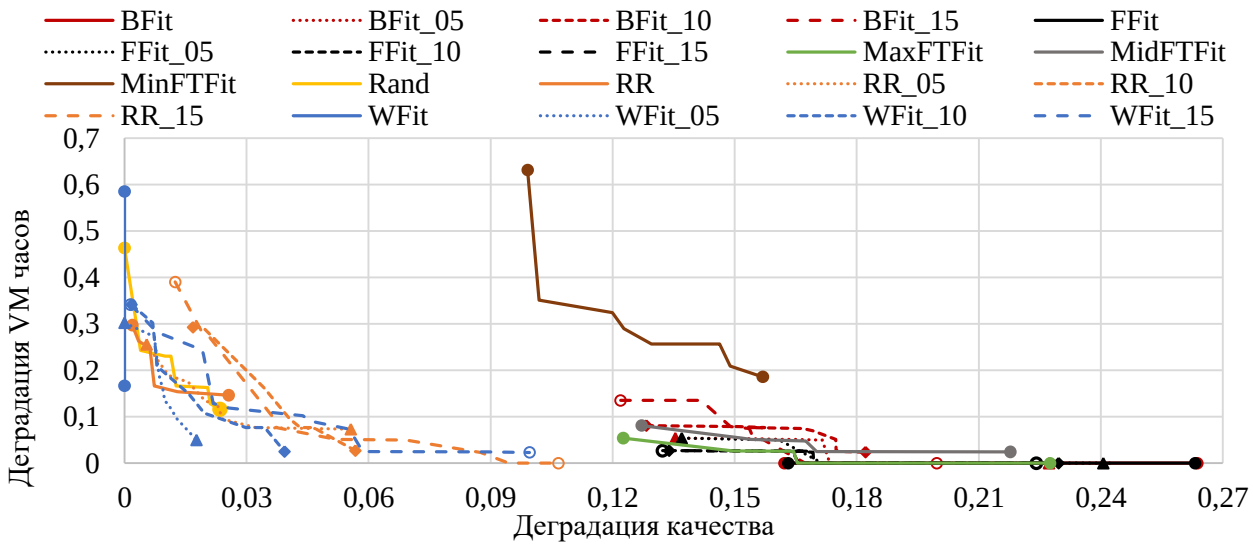


Рисунок 7.16. Парето-фронт.

Рейтинг стратегий основан на проценте покрытия. Более высокий рейтинг означает лучший фронт. В таблице 7.24 представлены результаты SC для каждого из двадцати Парето-фронтов. Согласно метрике покрытия множества, стратегией, которая имеет наилучший компромисс между $\bar{b}$ и $\bar{q}$, является WFit_15, за ней следуют MaxFTFit, RR_15 и WFit_05.

Видно, что MaxFTFit доминирует над фронтами других стратегий в диапазоне 0-60%, занимая второе место в среднем на 21.8%. $SC(A, MaxFTFit)$ показывает, что MaxFTFit доминирует над другими фронтами в среднем на 6.9%. Между тем, WFit_05 и MaxFTFit, занимающие второе и третье места, доминируются другими стратегиями в среднем на 19.5% и 21.9%, соответственно. Они доминируют над другими стратегиями на 6.2% и 6.9%.

Таблица 7.24. Покрытие множеств и ранг (дни).

A \ B	BFit	BFit_05	BFit_10	BFit_15	FFit	FFit_05	FFit_10	FFit_15	MaxFTFit	MidFTFit	MinFTFit
BFit	1	0	0	0	0.30	0	0	0	0	0	0
BFit_05	0.07	1	0.06	0.02	0.08	0.17	0.05	0.01	0.01	0.01	0
BFit_10	0.03	0.17	1	0.07	0.02	0.11	0.10	0.05	0.02	0.02	0
BFit_15	0.03	0.23	0.28	1	0.05	0.10	0.11	0.10	0.03	0.01	0
FFit	0.21	0	0	0	1	0	0	0	0	0	0
FFit_05	0.33	0.06	0.04	0.02	0.30	1	0	0	0.01	0	0
FFit_10	0.18	0.17	0.09	0.06	0.18	0.38	1	0.06	0.04	0.03	0
FFit_15	0.21	0.29	0.19	0.12	0.21	0.33	0.34	1	0.03	0.06	0
MaxFTFit	0.43	0.42	0.28	0.24	0.46	0.60	0.47	0.38	1	0.09	0
MidFTFit	0.02	0.20	0.31	0.18	0	0.14	0.13	0.08	0.03	1	0
MinFTFit	0	0	0	0	0	0	0	0	0	0	1
Rand	0	0	0	0	0	0	0	0	0	0.01	0.94
RR	0	0	0	0	0	0	0	0.01	0	0	0.90
RR_05	0	0.04	0.04	0.04	0	0	0	0.01	0.01	0.08	0.97
RR_10	0.01	0.13	0.20	0.17	0	0.06	0.06	0.07	0.04	0.28	0.99
RR_15	0.02	0.32	0.39	0.39	0.02	0.11	0.17	0.16	0.06	0.52	0.98
WFit	0	0	0	0	0	0	0	0	0	0	0.84
WFit_05	0	0.04	0.02	0.02	0	0.02	0	0.01	0	0.03	0.96
WFit_10	0	0.09	0.15	0.17	0.01	0.05	0.06	0.06	0.03	0.25	0.97
WFit_15	0.02	0.26	0.32	0.32	0.01	0.14	0.16	0.14	0.08	0.45	0.98
Mean	0.128	0.171	0.169	0.140	0.132	0.160	0.13	0.106	0.069	0.142	0.477
Rank	12	19	18	15	14	17	13	9	**3**	16	20

A \ B	Rand	RR	RR_05	RR_10	RR_15	WFit	WFit_05	WFit_10	WFit_15	Mean	Rank
BFit	0	0	0	0	0	0	0	0	0	0.065	18
BFit_05	0	0	0	0	0	0	0	0	0	0.074	17
BFit_10	0	0	0	0	0	0	0	0	0	0.079	16
BFit_15	0	0	0	0	0	0	0	0	0	0.096	14
FFit	0	0	0	0	0	0	0	0	0	0.061	19
FFit_05	0	0	0	0	0	0	0	0	0	0.088	15
FFit_10	0	0	0	0	0	0	0	0	0	0.109	11
FFit_15	0	0	0	0	0	0	0	0	0	0.139	8
MaxFTFit	0	0	0	0	0	0	0	0	0	0.218	**2**
MidFTFit	0	0	0	0	0	0	0	0	0	0.105	13
MinFTFit	0	0	0	0	0	0	0	0	0	0.050	20
Rand	1	0.01	0.09	0.03	0	0	0.05	0.01	0.02	0.107	12
RR	0.52	1	0.10	0.02	0.03	0	0.07	0.00	0.02	0.133	10
RR_05	0.03	0.02	1	0.21	0.11	0.01	0.01	0.19	0.14	0.145	7
RR_10	0.02	0	0.02	1	0.31	0	0	0.01	0.21	0.178	6
RR_15	0	0	0.02	0.01	1	0	0	0	0.02	0.210	**3**
WFit	0.27	0.41	0.08	0.01	0.02	1	0.09	0.03	0.01	0.138	9
WFit_05	0.42	0.15	0.51	0.20	0.15	0.01	1	0.22	0.15	0.195	4
WFit_10	0.01	0.04	0.06	0.38	0.28	0	0.02	1	0.28	0.194	5
WFit_15	0	0.01	0.02	0.06	0.39	0	0	0.02	1	0.219	**1**
Mean	0.113	0.082	0.095	0.096	0.114	0.051	0.062	0.074	0.092		
Rank	10	5	7	8	11	**1**	**2**	4	6		

Однако не следует рассматривать только Парето-фронты, когда многие решения находятся за пределами оптимальных по Парето решений. Это случай BFit_xx, FFit_xx, RR_xx: хотя Парето-фронты имеют хорошее качество, многие из сгенерированных решений находятся довольно далеко от них, и, следовательно, один запуск алгоритма может дать значительно худшие результаты.

7.8.1 Второй сценарий

Во втором сценарии в качестве тестовых примеров используются восемь задержек StUp: 0, 45, 90, 90, 135, 180, 225, 270 и 315 секунд [165, 132] для VM и десять PS: 10, 20, 30, StUp (PS = StUp). StUp = 0 представляет собой идеальный предиктор, поскольку система запускает VM, когда ей это необходимо, предиктор идеально оценивает нагрузку в предыдущем PI, и он используется в качестве эталона для оценки производительности других стратегий при нескольких StUp.

Основной целью является анализ деградации качества обслуживания, присущей StUp, и снижение деградации качества за счет прогнозирования нагрузки. В таблице 7.25 представлена схема моделирования для второго сценария. В нем рассматривается 30 рабочих нагрузок по 24 часа без выходных и праздников.

Таблица 7.25. Настройка моделирования для второго сценария.

Имя	Значение
Super Node Cluster	1
Число стратегий:	128
С прогнозом	32
Без прогнозома	96
Число рабочих нагрузок	30
Нагрузка	24 hours
Число кодеков	4
Число вызовов	643,718
VMs StUp	7

Не представлено снижение качества голоса ($\bar{q}$), поскольку разница между лучшей и худшей стратегией составляет около 9.292×10^{-3}. Проанализируем число звонков, находящихся в режиме ожидания $\bar{c}$ как показатель снижения QoS.

Проведем совместный анализ двух метрик в соответствии с методикой средней деградации. Сначала представим анализ арендованных виртуальных машин ($\bar{b}$) и числа звонков на ожидании ($\bar{c}$) по отдельности. Затем найдем стратегию, обеспечивающую наилучший компромисс между ними.

В таблице 7.26 представлены средние значения деградации $\bar{b}$, $\bar{c}$ и их средние значения для лучших стратегий. Последние четыре столбца таблицы содержат ранжирование каждой стратегии относительно стоимости поставщика, качества, средних значений и ранжирования между стратегиями. Ранг $\bar{b}$ основан на деградации тарификационных часов. Ранг $\bar{c}$ основан на числе звонков на ожидании. Средний ранг – основан на усреднении двух рангов. Rank – это относительная позиция всех стратегий, она рассматривает среднее значение Rank $\bar{b}$ и Rank $\bar{c}$.

Таблица 7.26. Деградация и ранжирование для лучших стратегий распределения.

Стратегия	Deg $\bar{b}$	Deg $\bar{c}$	Mean	Rank $\bar{b}$	Rank $\bar{c}$	Rank mean	Rank
NN_FFit_stUp	0.0000	4.1342	2.0671	**1**	100	127	32
NN_BFit_stUp	0.0084	3.9974	2.0029	**2**	99	126	32
NN_MaxFTFit_stUp	0.0157	4.4158	2.2158	**3**	101	128	34
RoC_WFit_s30	0.4455	0.0000	0.2227	105	**1**	30	36
RoC_MinFTFit_s30	0.5118	0.0000	0.2559	118	**1**	36	49
RoC_Rand_s30	0.3918	0.0000	0.1959	85	**1**	21	18
RoC_RR_s30	0.4056	0.0000	0.2028	91	**1**	22	23
Roc_MinFTFit_s10	0.5253	0.0132	0.2692	121	**2**	41	53
RoC_WFit_s10	0.4543	0.0132	0.2338	106	**2**	33	38
RoC_RR_s10	0.4192	0.0158	0.2175	95	**3**	27	29
Rand_15	0.1731	0.0579	0.1155	64	11	**1**	12
RoC_FFit_s30	0.0464	0.2026	0.1245	15	28	**2**	**2**
RoC_BFit_s30	0.0517	0.2026	0.1272	20	28	**3**	**3**
RoC_FFit_s10	0.0425	0.2500	0.1462	9	33	9	**1**
RoC_FFit_s20	0.0427	0.2474	0.1451	10	32	8	**1**
RR	0.4222	0.0184	0.2203	**97**	**4**	**29**	**32**

Лучшей стратегией для $\bar{b}$ является NN_FFit_stUp, которая распределяет вызовы на основе First Fit с временем предсказания, равным StUp. Однако это вторая худшая стратегия для $\bar{c}$. Она имеет тенденцию к увеличению утилизации и снижению качества. NN_BFit_stUp и NN_MaxFTFit_stUp – вторая и третья лучшие стратегии (в отношении $\bar{b}$). Все стратегии используют нейронные сети для прогнозирования поступления будущих вызовов.

Лучшими стратегиями для $\bar{c}$ являются RoC_MinFTFit_s30 (где вызовы помещаются в VM, время аренды которой ближе всего к завершению), RoC_Rand_s30, RoC_RR_s30 и RoC_WFit_s30. Они используют RoC для прогнозирования рабочей нагрузки, а время прогнозирования составляет 30 секунд. Они имеют тенденцию к недоиспользованию виртуальных машин, уменьшая $\bar{c}$, но увеличивая $\bar{b}$.

Наконец, 28 стратегий улучшают производительность Round Robin, учитывая среднее ухудшение $\bar{b}$ и $\bar{c}$. Ранг Round Robin равен 32, по отношению к относительному положению всех стратегий, это означает, что, по крайней мере 52 стратегии имеют производительность равную или лучшую, чем эта стратегия.

7.9 Выводы

В данной главе формулируются и исследуются проблемы планирования, связанные с облачными системами VoIP, с учетом динамической нагрузки, и с прогнозированием начала использования VM. Бикритериальная модель учитывает стоимость услуг, выраженную в тарификационных часах для используемых виртуальных машин, и качество обслуживания, на которое влияют обработка вызовов и их ожидание на линии.

Проводится комплексное моделирование на реальных данных онлайновых неклаирвоянтных стратегий планирования с фиксированным порогом использования для запроса VM, а также стратегий с динамическим

прогнозированием нагрузки. Результаты экспериментов показывают, что предложенные алгоритмы могут быть эффективно использованы в облачной среде VoIP.

Показано, что наши стратегии с прогнозированием нагрузки превосходят известные стратегии, обеспечивая приемлемое качество обслуживания и более низкую стоимость. Также проанализирована устойчивость этих стратегий при изменении времени задержки запуска виртуальных машин для работы с реалистичными облачными средами VoIP.

ЗАКЛЮЧЕНИЕ

Работа носит теоретический характер. Полученные в ней результаты позволяют изучить различные аспекты планирования работ в распределенных гетерогенных вычислительных системах в условиях нестационарности, и исследовать теоретические оценки границ оптимизации.

Заявленные и представленные выше результаты можно сгруппировать следующим образом:

- Сформулирована проблема планирования работ в гетерогенной распределенной вычислительной системе в условиях ее нестационарности для различных сценариев и предложены подходы к ее решению путем динамической адаптации планировщика к различным рабочим нагрузкам, изменениям конфигурации системы и ее параметров.
- Разработаны алгоритмы с гарантированным конкурентным фактором для онлайн случая и аппроксимационным фактором для оффлайн случая с использованием подхода динамической "кражи работ" (англ. “job stealing").
- Разработана двухуровневая модель гетерогенной распределенной вычислительной системы, которая интегрирует две задачи планировщика: распределение параллельных работ по машинам и локального планирования. Решения по планированию принимаются без точной информации о производительности машин и времени выполнения работ.
- Для снижения влияния неполноты информации и эффективного решения соответствующей проблемы оптимизации, введен и исследован коэффициент допустимости (англ. “admissible factor”), включенного в политику планирования. Показано как он может быть динамически скорректирован для того, чтобы справиться с непредсказуемой рабочей нагрузкой улучшая конкурентный фактор. Получены теоретических оценки конкурентных и

аппроксимационных факторов алгоритма улучшая и расширяя известные результаты.

- Разработана модель планирования с регулированием периодов простоя машин. Получены новые и обобщены известные предельные границы производительности планирования в наихудшем случае.
- Разработаны модели планирования облачных вычислений на основе уровней обслуживания. Проведен конкурентный анализ алгоритмов для различных сценариев с одним или несколькими уровнями обслуживания, с одной или несколькими машинами, а также с жадным и ограниченным принятием работ.
- Сформулирован, разработан и исследован облачный VoIP планировщик, который учитывает динамическую рабочую нагрузку, вариативность времени запуска VM, свойства вызовов, инфраструктуры, и т. д. Он адаптируется к поведению VM из-за числа и типа вызовов (голосовой почты, видео/аудио конференций, передачи изображений и текста, и т. п.) для предотвращения снижения качества сервиса.
- Задача планирования рассмотрена как частный случай динамической упаковки в контейнеры. Временное существование элементов (вызовов) при упаковке является принципиальной новизной данной проблемы. В отличие от стандартной формулировки, контейнеры всегда открыты, даже если они полностью заполнены, так как элементы в контейнерах могут быть удалены (завершение вызова). Экспериментальный анализ на реальных данных компании MiXvoip (телекоммуникационный и Интернет-провайдер, Люксембург) показал, что предложенные алгоритмы с методами прогнозирования нагрузки и разработанными настраиваемыми параметрами, превосходят известные стратегии, обеспечивая высокое качество обслуживания и более низкую стоимость и могут быть эффективно использованы в облачной среде VoIP.

СПИСОК ЛИТЕРАТУРЫ

Статьи автора в журналах, рекомендованных ВАК РФ, Scopus, Web of Science

1. Armenta-Cano, F. A., Tchernykh, A., Cortés-Mendoza, J. M., Yahyapour, R., Drozdov, A. Y., Bouvry, P., Kliazovichd, D., Avetisyane, A., Nesmachnow, S. Min_c: Heterogeneous concentration policy for energy-aware scheduling of jobs with resource contention // Programming and Computer Software. – 2017. – Т. 43. – №. 3. – С. 204-215.
2. Cortés-Mendoza, J. M., Tchernykh, A., Simionovici, A. M., Bouvry, P., Nesmachnow, S., Dorronsoro, B., & Didelot, L. VoIP service model for multi-objective scheduling in cloud infrastructure // International Journal of Metaheuristics. – 2015. – Т. 4. – №. 2. – С. 185-203.
3. Cortés-Mendoza, J.M.; Tchernykh, A.; Armenta-Cano, F.A.; Bouvry, P.; Drozdov, A.Y.; Didelot, L. Biobjective VoIP service management in cloud infrastructure // Scientific Programming. – 2016. – Т. 2016.
4. Cristobal-Salas A, Tchernykh, A., Gaudiot, J. L., & Lin, W. Y. Non-strict execution in parallel and distributed computing // International Journal of Parallel Programming. – 2003. – Т. 31. – №. 2. – С. 77-105.
5. Díaz A. R., Tchernykh A., Ecker K. H. Algorithms for dynamic scheduling of unit execution time tasks //European Journal of Operational Research. – 2003. – Т. 146. – №. 2. – С. 403-416.
6. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., González-García, J. L., Röblitz, T., & Ramírez-Alcaraz, J. M. Multiple workflow scheduling strategies with user run time estimates on a grid // Journal of Grid Computing. – 2012. – Т. 10. – №. 2. – С. 325-346.
7. Ivashko, E. E., Ivashko, A. A., Safonov, G. R., Tchernykh, A. Cost-Efficient

Strategy in Clouds with Spot Price Uncertainty //Automation & Remote Control. – 2020. – T. 81. – №. 4.

8. Kliazovich, D., Pecero, J. E., Tchernykh, A., Bouvry, P., Khan, S. U., & Zomaya, A. Y. CA-DAG: Modeling communication-aware applications for scheduling in cloud computing //Journal of Grid Computing. – 2016. – T. 14. – №. 1. – C. 23-39.
9. Ramírez-Alcaraz, J. M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., González-García, J. L. Hirales-Carbajal, A. Job allocation strategies with user run time estimates for online scheduling in hierarchical grids //Journal of Grid Computing. – 2011. – T. 9. – №. 1. – C. 95-116.
10. Ramírez-Velarde, R., Tchernykh, A., Barba-Jimenez, C., Hirales-Carbajal, A., Nolazco-Flores, J. Adaptive resource allocation with job runtime uncertainty // Journal of Grid Computing. – 2017. – T. 15. – №. 4. – C. 415-434.
11. Tchernykh A., Ecker K. Worst case behavior of list algorithms for dynamic scheduling of non-unit execution time tasks with arbitrary precedence constrains // IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences. – 2008. – T. 91. – №. 8. – C. 2277-2280.
12. Tchernykh, A.; Babenko, M.; Chervyakov, N.; Miranda-Lopez, V.; Avetisyan, A.; Drozdov, A.Y.; Rivera-Rodriguez, R.; Radchenko, G.; Du, Z. Scalable data storage design for nonstationary IoT environment with adaptive security and reliability //IEEE Internet of Things Journal. – 2020. – T. 7. – №. 10. – C. 10171-10188.
13. Tchernykh, A.; Bychkov, I.; Feoktistov, A.; Gorsky, S.; Sidorov, I.; Kostromin, R.; Edelev, A.; Zorkalzev, V.; Avetisyan, A. Mitigating Uncertainty in Developing and Applying Scientific Applications in an Integrated Computing Environment // Programming and Computer Software. – 2020. – T. 46. – №. 8. – C. 483-502.
14. Tchernykh, A.; Cortés-Mendoza, J.M.; Bychkov, I.; Feoktistov, A.; Didelot, L.; Bouvry, P.; Radchenko, G.; Borodulin, K. Configurable cost-quality optimization of cloud-based VoIP // Journal of Parallel and Distributed Computing. – 2019. –

T. 133. – C. 319-336.

15. Tchernykh, A.; Lozano, L.; Schwiegelshohn, U.; Bouvry, P.; Pecero, J.E.; Nesmachnow, S.; Drozdov, A.Y. Online Bi-Objective Scheduling for IaaS Clouds Ensuring Quality of Service // Grid Computing. – 2015.
16. Tchernykh, A.; Pecero, J.E.; Barrondo, A.; Schaeffer, E. Adaptive energy efficient scheduling in Peer-to-Peer desktop grids //Future Generation Computer Systems. – 2014. – T. 36. – C. 209-220
17. Tchernykh, A.; Schwiegelshohn, U.; On-line hierarchical job scheduling on grids with admissible allocation //Journal of Scheduling. – 2010. – T. 13. – №. 5. – C. 545-552.
18. Tchernykh, A.; Schwiegelsohn, U.; Talbi, E.; Babenko, M. Towards understanding uncertainty in cloud computing with risks of confidentiality, integrity, and availability //Journal of Computational Science. – 2019. – T. 36. – C. 100581.
19. Tchernykh, A.; Stepanov, A.; Lupenko, A.; Tchernykh, N. Abstract Network Machine: parallel computation on associative networks with incompete data structures //Dyna. – 2003. – T. 70. – №. 140. – C. 71-88.
20. Tchernykh, A.; Trystram, D.; Brizuela, C.; Scherson, I. Idle regulation in non-clairvoyant scheduling of parallel jobs //Discrete Applied Mathematics. – 2009. – T. 157. – №. 2. – C. 364-376.
21. Quezada-Pina, A., Tchernykh, A., González-García, J. L., Hirales-Carbajal, A., Ramírez-Alcaraz, J. M., Schwiegelshohn, U., Yahyapour, R., & Miranda-López, V. Adaptive parallel job scheduling with resource admissible allocation on two-level hierarchical grids //Future Generation Computer Systems. – 2012. – T. 28. – №. 7. – C. 965-976.

Другие публикации автора по теме диссертации

22. Armenta-Cano, F.; Tchernykh, A.; Cortés-Mendoza J, M.; A., A.; Yahyapour, D.; Drozdov A, Y.; Bouvry P, K. Heterogeneous job consolidation for power aware scheduling with quality of service. // In Proceedings of the Russian Conference on Supercomputing – 2015. – С. 687-697.
23. Armenta-Cano, F., Tchernykh, A., Yahyapour, R., & Nabrzyski, J. Cost Optimization of Virtual Machine Provisioning in Federated IaaS Clouds // 5th International Conference on Supercomputing in Mexico. – 2015.
24. Babenko, M., Tchernykh, A., Chervyakov, N., Murga, D., Dvoryaninova, I., Miranda-López, V., Kucherov, N. Multi-Clouds Workload Distribution for the Secure and Reliable Storage of Data under Uncertainty //Applications". Irkutsk: ESI SB RAS, 2017, 224 p. – 2017. – С. 30.
25. Barrondo, A., Tchernykh, A., Schaeffer, E., & Pecero, J. Energy efficiency of knowledge-free scheduling in peer-to-peer desktop grids //2012 International Conference on High Performance Computing & Simulation (HPCS). – IEEE, 2012. – С. 105-111.
26. Canosa, R., Tchernykh, A., Cortés-Mendoza, J. M., Rivera-Rodriguez, R., Rizk, J. L., Avetisyan, A., Du Z., Radchenko G., Morales, E. R. C. Energy consumption and quality of service optimization in containerized cloud computing // 2018 Ivannikov Ispras Open Conference (ISPRAS). – IEEE, 2018. – С. 47-55.
27. Combarro, M., Tchernykh, A., Kliazovich, D., Drozdov, A., & Radchenko, G. Energy-aware scheduling with computing and data consolidation balance in 3-tier data center //2016 International Conference on Engineering and Telecommunication (EnT). – IEEE, 2016. – С. 29-33.
28. Cortés-Mendoza, J. M., Tchernykh, A., Bychkov, I., Feoktistov, A., Bouvry, P., Didelot, L. Load-aware strategies for cloud-based VoIP optimization with VM startup prediction // 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). – IEEE, 2017. – С. 472-481.
29. Cortés-Mendoza, J. M., Tchernykh, A., Drozdov, A. Y., Didelot, L. Robust cloud

VoIP scheduling under VMs startup time delay uncertainty // 2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC). – IEEE, 2016. – C. 234-239.

30. Cortés-Mendoza, J. M., Tchernykh, A., Drozdov, A., Bouvry, P., Simionovici, A. M., Kliazovich, D., Avetisyan, A. Distributed adaptive VoIP load balancing in hybrid clouds // Russian Supercomputing Days 2015. – 2015.
31. Cortés-Mendoza, J. M., Tchernykh, A., Radchenko, G., Drozdov, A. Y. RoC Prediction for Bi-Objective Cost-QoS Optimization of Cloud VoIP Call Allocations //2017 IVth International Conference on Engineering and Telecommunication (EnT). – IEEE, 2017. – C. 119-123.
32. Cristóbal-Salas A., Tchernykh A., Gaudiot J. L. Non-strict evaluation of the FFT algorithm in distributed memory systems // European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. – Springer, Berlin, Heidelberg, 2003. – C. 188-195.
33. Cristóbal-Salas, A., Chernykh, A., Rodríguez-Alcantar, E., & Gaudiot, J. L. Exploiting single-assignment properties to optimize message-passing programs by code transformations // Symposium on Implementation and Application of Functional Languages. – Springer, Berlin, Heidelberg, 2004. – C. 1-16.
34. Galaviz-Alejos, L. A., Armenta-Cano, F., Tchernykh, A., Radchenko, G., Drozdov, A. Y., Sergiyenko, O., & Yahyapour, R. Bi-objective heterogeneous consolidation in cloud computing // Latin American High Performance Computing Conference. – Springer, Cham, 2017. – C. 384-398.
35. Garibay Martinez, R.; Tchernykh, A.; Ecker, K.H. Re allocation Cost Minimization in Distributed Dynamic Real Time Systems. // In Proceedings of the 16th International Conference on Real-Time and Network Systems. – 2008. – C. 49–52.
36. Gómez, C. E., Chavarriaga, J., Tchernykh, A., Castro, H. E. Improving reliability for provisioning of virtual machines in desktop clouds //European Conference on Parallel Processing. – Springer, Cham, 2019. – C. 669-680.
37. Hirales-Carbajal, A., Tchernykh, A., Röblitz, T., Yahyapour, R. A Grid

simulation framework to study advance scheduling strategies for complex workflow applications // 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW). – IEEE, 2010. – С. 1-8.

38. Iturriaga, S., Nesmachnow, S., Tchernykh, A., and Dorronsoro, B. The processor working set and its Multiobjective workflow scheduling in a federation of heterogeneous green-powered data centers //2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). – IEEE, 2016. – С. 596-599.
39. Kliazovich, D., Pecero, J. E., Tchernykh, A., Bouvry, P., Khan, S. U., Zomaya, A. Y. CA-DAG: Communication-aware directed acyclic graphs for modeling cloud computing applications // 2013 IEEE sixth international conference on cloud computing. – IEEE, 2013. – С. 277-284.
40. Lopez-Falcon, E. C., Miranda-López, V., Tchernykh, A., Babenko, M., Avetisyan, A. Bi-objective analysis of an adaptive secure data storage in a multi-cloud // Latin American High Performance Computing Conference. – Springer, Cham, 2018. – С. 307-321.
41. Miranda V., Tchernykh A., Kliazovich D. Dynamic communication-aware scheduling with uncertainty of workflow applications in clouds //International Conference on Supercomputing in Mexico. – Springer, Cham, 2015. – С. 169-187.
42. Schwiegelshohn U., Tchernykh A. Online scheduling for cloud computing and different service levels // 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. – IEEE, 2012. – С. 1067-1074.
43. Schwiegelshohn U., Tchernykh A., Yahyapour R. Online scheduling in grids //2008 IEEE International Symposium on Parallel and Distributed Processing. – IEEE, 2008. – С. 1-10.
44. Tchernykh A. From Static Scheduling Towards Understanding Uncertainty // Algorithms and Scheduling Techniques to Manage Resilience and Power

Consumption in Distributed Systems. – 2016. – Т. 5. – № 7. – С. 19. (Reports)

45. Tchernykh A. Idle Regulation in On-Line Scheduling of Multiprocessor Jobs // International Seminar on Scheduling in Computer and Manufacturing Systems. 04231, Dagstuhl, Germany – 2004. (abstracts).
46. Tchernykh A. Multi criteria optimization with Quality of Service in Cloud Computing // CLOUDCOMS - Eastern Europe-Luxembourg Workshop on Cloud Computing Communications, Security and Services. The 3rd IEEE International Conference on Cloud Networking (IEEE CloudNet 2014). – 2014.
47. Tchernykh A. Online Scheduling for Cloud Computing and Different Service Levels and Quality of Service // New Challenges in Scheduling Theory. // Marseille-Lumini-CIRM – 2014. (book of abstracts).
48. Tchernykh A. Online Scheduling in Grids. New Challenges in Scheduling Theory // Marseille-Lumini-CIRM. – 2008. (book of abstracts).
49. Tchernykh A., Lozano Rizk J. Big Data-aware Scheduling with Uncertainty in Cloud Computing // Big Network Workshop, CENIC - Corporation for Education Network Initiatives in California. – 2013.
50. Tchernykh A., Trystram D. Adaptive strategy for on-line scheduling of real world parallel applications //SOCAL. Southern California Workshop on Parallel and Distributed Processing and Architecture (book of abstracts), Santa Barbara, USA, – 2002. – Т. 2. – С. 7-8.
51. Tchernykh A., Trystram D. Online scheduling of multiprocessor jobs with idle regulation //International Conference on Parallel Processing and Applied Mathematics. – Springer, Berlin, Heidelberg, 2003. – С. 131-144.
52. Tchernykh A., Trystram D., Rapine C. Adaptive (a, b, c)-scheme strategy for on-line scheduling //New Trends in Scheduling in Parallel and Distributed Systems, CIRM, Luminy, Marseille, France. – 2001.
53. Tchernykh, A., Babenko, M., Kuchukov, V., Miranda-López, V., Avetisyan, A., Rivera-Rodriguez, R., & Radchenko, G. Data reliability and redundancy optimization of a secure multi-cloud storage under uncertainty of errors and falsifications //2019 IEEE International Parallel and Distributed Processing

Symposium Workshops (IPDPSW). – IEEE, 2019. – С. 565-572.

54. Tchernykh, A., Cortés-Mendoza, J. M., Pecero, J. E., Bouvry, P., & Kliazovich, D. Adaptive energy efficient distributed VoIP load balancing in federated cloud infrastructure //2014 IEEE 3rd International Conference on Cloud Networking (CloudNet). – IEEE, 2014. – С. 27-32.
55. Tchernykh, A., Facio-Medina, A., Pulido-Gaytan, B., Rivera-Rodriguez, R., Cortés-Mendoza, J. M., Radchenko, G., Babenko M., Chernykh I., Kulikov I., Nesmachnow, S. Toward digital twins' workload allocation on clouds with low-cost microservices streaming interaction //2020 Ivannikov Ispras Open Conference (ISPRAS). – IEEE, 2020. – С. 115-121.
56. Tchernykh, A., Lozano, L., Bouvry, P., Pecero, J. E., Schwiegelshohn, U., & Nesmachnow, S. Energy-aware online scheduling: ensuring quality of service for IaaS clouds //2014 International Conference on High Performance Computing & Simulation (HPCS). – IEEE, 2014. – С. 911-918.
57. Tchernykh, A., Lozano, L., Schwiegelshohn, U., Bouvry, P., Pecero, J. E., Nesmachnow, S. Bi-objective online scheduling with quality of service for IaaS clouds // 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet). – IEEE, 2014. – С. 307-312.
58. Tchernykh, A., Ramírez, J. M., Avetisyan, A., Kuzjurin, N., Grushin, D., Zhuk, S. Two level job-scheduling strategies for a computational grid // PPAM International Conference on Parallel Processing and Applied Mathematics. – Springer, Berlin, Heidelberg, 2005. LNCS 3911, Springer-Verlag, 2006. – С. 774-781.
59. Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N. Online hierarchical job scheduling on Grids //From Grids to Service and Pervasive Computing. – Springer, Boston, MA, 2008. – С. 77-91.
60. Tchernykh, A., Schwiegelsohn, U., Alexandrov, V., & Talbi, E. G. Towards understanding uncertainty in cloud computing resource provisioning //Procedia Computer Science. – 2015. – Т. 51. – С. 1772-1781.
61. Tchernykh, A., Schwiegelsohn, U., Alexandrov, V., Talbi, E. G. Uncertainty in

clouds: Challenges of efficient resource provisioning // Russian Conference on Supercomputing – 2015. – С. 698-699.

62. Yaurima-Basaldua, V. H., Tchernykh, A., Castro-Garcia, Y., Villagomez-Ramos, V. M., & Burtseva, L. Genetic algorithm calibration for two objective scheduling parallel jobs on hierarchical Grids //International Conference on Parallel Processing and Applied Mathematics. – Springer, Berlin, Heidelberg, 2011. – С. 61-70.
63. Zhuk, S., Chernykh, A., Avestiyan, A., Gaissaryan, S., Kuzjurin, N., Pospelov, A., & Grushin, D. Comparison of scheduling heuristics for grid resource broker //Proceedings of the Fifth Mexican International Conference in Computer Science, 2004. ENC 2004. – IEEE, 2004. – С. 388-392.
64. Патент № 2744815 Российская Федерация, МПК G06F 7/72. Устройство для перевода чисел из системы остаточных классов и расширения оснований: № 2020120649; заявл. 22.06.2020; опубл. 16.03.2021 / Бабенко М.Г., Кучуков В.А., Черных А.Н., Кучеров Н.Н.; заявитель и патентообладатель Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет». – 13 с.

Цитируемая литература

65. Капустин В. Ф. Неопределенность: виды, интерпретации, учет при моделировании и принятии решений // Вестник Санкт-Петербургского университета. – 1993. – Т. 2. – С. 108-114.
66. Черных, А. Н., Бычков, И. В., Феоктистов, А. Г., Горский, С. А., Сидоров, И. А., Костромин, Р. О., Еделев А.В., Зоркальцев В.И., Аветисян, А. И. Смягчение неопределенности при разработке научных приложений в интегрированной среде // Труды Института системного программирования РАН. – 2021. – Т. 33. – №. 1. – С. 151-172.
67. Aggarwal, A. K., Kent, R. D. An adaptive generalized scheduler for grid applications // 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05). – IEEE, 2005. – С. 188-194.
68. Albers S. Better bounds for online scheduling // SIAM Journal on Computing. – 1999. – Т. 29. – №. 2. – С. 459-473.
69. Ali, S., Maciejewski, A. A., Siegel, H. J., & Kim, J. K. Definition of a robustness metric for resource allocation // Proceedings International Parallel and Distributed Processing Symposium. – IEEE, 2003. – С. 10.
70. Amazon, “Elastic Cloud,” 2012. Pricing. [Online]. Available: http://aws.amazon.com/ec2/pricing/.
71. Bar-Noy, A., Freund, A., Naor, J. On-line load balancing in a hierarchical server topology // SIAM Journal on Computing. – 2001. – Т. 31. – №. 2. – С. 527-549.
72. Baruah, S. K., Haritsa, J. R. Scheduling for overload in real-time systems // IEEE Transactions on computers. – 1997. – Т. 46. – №. 9. – С. 1034-1039.
73. Bhadani, A., Chaudhary, S. Performance evaluation of web servers using central load balancing policy over virtual machines on cloud // Proceedings of the Third Annual ACM Bangalore Conference. – 2010. – С. 1-4.
74. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T. G. Scheduling divisible loads in parallel and distributed systems. – John Wiley & Sons, 1996. – Т. 8.

75. Blayo, E., Debreu, L., Mounié, G., Trystram, D. Dynamic load balancing for adaptive mesh ocean circulation model //Engineering Simulations. – 2000. – Т. 22. – №. 2. – С. 8-24.

76. Błażewicz, J., Drozdowski, M., Ecker, K. Management of resources in parallel systems // Handbook on Parallel and Distributed Processing. – Springer, Berlin, Heidelberg, 2000. – С. 263-341.

77. Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Weglarz, J. Scheduling computer and manufacturing processes. – springer science & Business media, 2013.

78. Bougeret M., Dutot P.F., Jansen K., Otte C., Trystram D. A fast 5/2-approximation algorithm for hierarchical scheduling. // In European Conference on Parallel Processing. Springer, – 2010. – С. 157-167.

79. Bougeret M., Dutot P.F., Jansen K., Otte C., Trystram D. Approximating the non-contiguous multiple organization packing problem. // In IFIP International Conference on Theoretical Computer Science. Springer. – 2010. – С. 316-327.

80. Bougeret, M., Dutot, P. F., Jansen, K., Otte, C., Trystram, D. Approximation algorithms for multiple strip packing // International Workshop on Approximation and Online Algorithms. – Springer, Berlin, Heidelberg, 2009. – С. 37-48.

81. Bougeret, M., Dutot, P. F., Jansen, K., Robenek, C., & Trystram, D. Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms // Discrete Mathematics, Algorithms and Applications. – 2011. – Т. 3. – №. 4. – С. 553-586.

82. Bougeret, M., Dutot, P. F., Trystram, D., Jansen, K., & Robenek, C. Improved approximation algorithms for scheduling parallel jobs on identical clusters // Theoretical Computer Science. – 2015. – Т. 600. – С. 70-85.

83. Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Freund, R. F., Robertson, J. P., Theys, M. D., Yao, B. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous

distributed computing systems // Journal of Parallel and Distributed computing. – 2001. – T. 61. – №. 6. – C. 810-837.

84. Brent, R. P. The parallel evaluation of arithmetic expressions in logarithmic time // Complexity of sequential and parallel numerical algorithms. – 1973. – C. 83-102.
85. Bryant, R., Madsen, L., Van Meggelen, J. Asterisk: The Definitive Guide: The Future of Telephony Is Now. – " O'Reilly Media, Inc.", 2013.
86. Buyya, R., Abramson, D., Giddy, J., Stockinger, H. Economy Models for Resource Management and Grid Scheduling // Concurrency and Computation: Practice and Experience. – 2002. – T. 14. – C. 1507-1542.
87. Cai, X., Wu, X., Zhang, L., Zhou, X. Scheduling with stochastic approaches // Sequencing and Scheduling with Inaccurate Data. – 2014. – C. 3-45.
88. Calheiros, R. N., Masoumi, E., Ranjan, R., Buyya, R. Workload prediction using ARIMA model and its impact on cloud applications' QoS // IEEE Transactions on Cloud Computing. – 2014. – T. 3. – №. 4. – C. 449-458.
89. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., Buyya, R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms // Software: Practice and Experience. – 2011. – T. 41. – №. 1. – C. 23-50.
90. Campos, L. M., Scherson, I. D. Rate of change load balancing in distributed and parallel systems // Parallel Computing. – 2000. – T. 26. – №. 9. – C. 1213-1230.
91. Canon L. C., Jeannot E. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments //IEEE Transactions on Parallel and Distributed Systems. – 2009. – T. 21. – №. 4. – C. 532-546.
92. Cat X., Zhang L. Preemptive stochastic online scheduling on uniform machines with bounded speed ratios // ICSSSM11. – IEEE, 2011. – C. 1-4.
93. Chapin, S. J., Cirne, W., Feitelson, D. G., Jones, J. P., Leutenegger, S. T., Schwiegelshohn, U., Smith, W., Talby, D. Benchmarks and standards for the evaluation of parallel job schedulers // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1999. – C. 67-90.

94. Coffman, Jr E. G., Garey, M. R., Johnson, D. S. Dynamic bin packing //SIAM Journal on Computing. – 1983. – Т. 12. – №. 2. – С. 227-258.

95. Costa, L. R., Nunes, L. S. N., Bordim, J. L., Nakano, K. Asterisk PBX capacity evaluation // 2015 IEEE International Parallel and Distributed Processing Symposium Workshop. – IEEE, 2015. – С. 519-524.

96. Da Silva, D. P., Cirne, W., Brasileiro, F. V. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids // European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2003. – С. 169-180.

97. da Silva, F. A. B., Scherson, I. D. Improving parallel job scheduling using runtime measurements // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 2000. – С. 18-38.

98. Dail, H., Casanova, H., Berman, F. A decoupled scheduling approach for the GrADS program development environment // SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing. – IEEE, 2002. – С. 55-55.

99. Decker, T., Krandick, W. Parallel real root isolation using the Descartes method //International Conference on High-Performance Computing. – Springer, Berlin, Heidelberg, 1999. – С. 261-268.

100. Downey A. B. A parallel workload model and its implications for processor allocation //Cluster Computing. – 1998. – Т. 1. – №. 1. – С. 133-145.

101. Downey A. B. Predicting queue times on space-sharing parallel computers // Proceedings 11th International Parallel Processing Symposium. – IEEE, 1997. – С. 209-218.

102. Dutot, P. F., Jansen, K., Robenek, C., & Trystram, D. A (2+ ε)-approximation for scheduling parallel jobs in platforms //European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2013. – С. 78-89.

103. Dutot, P. F., Pascual, F., Rzadca, K., & Trystram, D. Approximation algorithms for the multiorganization scheduling problem //IEEE Transactions on Parallel and Distributed Systems. – 2011. – Т. 22. – №. 11. – С. 1888-1895.

104. Eleftheriou C. 3CX Phone System and ATOM N270 Processor Benchmarking. 2010. [online] http://www.3cx.com/blog/voip-howto/atom-processor-n270-benchmarking.
105. Epstein, L., Noga, J., Woeginger, G. J. On-line scheduling of unit time jobs with rejection: minimizing the total completion time // Operations Research Letters. – 2002. – Т. 30. – №. 6. – С. 415-420.
106. Fang, Y., Wang, F., Ge, J. A task scheduling algorithm based on load balancing in cloud computing // International conference on web information systems and mining. – Springer, Berlin, Heidelberg, 2010. – С. 271-277.
107. Feitelson, D. G., Jettee, M. A. Improved utilization and responsiveness with gang scheduling // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1997. – С. 238-261.
108. Feitelson, D. G., Rudolph, L. Evaluation of design choices for gang scheduling using distributed hierarchical control // Journal of Parallel and Distributed Computing. – 1996. – Т. 35. – №. 1. – С. 18-34.
109. Feitelson, D. G., Rudolph, L. Metrics and benchmarking for parallel job scheduling // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1998. – С. 1-24.
110. Feitelson, D. G., Rudolph, L. Parallel job scheduling: Issues and approaches // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1995. – С. 1-18.
111. Feitelson, D. G., Rudolph, L. Toward convergence in job schedulers for parallel supercomputers // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1996. – С. 1-26.
112. Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., Wong, P. Theory and practice in parallel job scheduling // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1997. – С. 1-34.

113. Feldmann, A., Kao, M. Y., Sgall, J., Teng, S. H. Optimal on-line scheduling of parallel jobs with dependencies // Journal of Combinatorial Optimization. – 1998. – Т. 1. – №. 4. – С. 393-411.
114. Feldmann, A., Kao, M. Y., Sgall, J., Teng, S. H. Optimal online scheduling of parallel jobs with dependencies // Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing. – 1993. – С. 642-651.
115. Florence, A. P., Shanthi, V. A load balancing model using firefly algorithm in cloud computing // Journal of Computer Science. – 2014. – Т. 10. – №. 7. – С. 1156.
116. Foster I., Kesselman C. (ed.). The Grid 2: Blueprint for a new computing infrastructure. – Elsevier, 2003.
117. Foster, I. Globus toolkit version 4: Software for service-oriented science // Proceedings of IFIP International Conference on Network and Parallel Computing, LNCS. – 2005. – Т. 3779. – С. 213-223.
118. Fujimoto N., Hagihara K. Near-optimal dynamic task scheduling of independent coarse-grained tasks onto a computational grid // 2003 International Conference on Parallel Processing, 2003. Proceedings. – IEEE, 2003. – С. 391-398.
119. Galloway, J. M., Smith, K. L., Vrbsky, S. S. Power aware load balancing for cloud computing // Proceedings of the World Congress on Engineering and Computer Science. – 2011. – Т. 1. – С. 19-21.
120. Garey M. R., Graham R. L. Bounds for multiprocessor scheduling with resource constraints // SIAM Journal on Computing. – 1975. – Т. 4. – №. 2. – С. 187-200.
121. Garey M. R., Johnson D. S. Complexity results for multiprocessor scheduling under resource constraints //SIAM Journal on Computing. – 1975. – Т. 4. – №. 4. – С. 397-411.
122. Ghosal D., Serazzi G., Tripathi S. K. The processor working set and its use in scheduling multiprocessor systems //IEEE Transactions on Software Engineering. – 1991. – Т. 17. – №. 5. – С. 443-453.
123. González García J. L., Yahyapour R., Tchernykh A. Load balancing for parallel

computations with the finite element method // Computación y Sistemas, – 2013. . – Т. 17. – №. 3. – С. 299-316.

124. Gören S., Sabuncuoglu I. A Bi-criteria approach to scheduling in the face of uncertainty: considering robustness and stability simultaneously //Sequencing and Scheduling with Inaccurate Data. Y. Sotskov, F. Werner (eds.). – 2014. – С. 253-280.
125. Graham, R. L. Bounds on multiprocessing timing anomalies // SIAM journal on Applied Mathematics. – 1969. – Т. 17. – №. 2. – С. 416-429.
126. Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. Optimization and approximation in deterministic sequencing and scheduling: a survey //Annals of Discrete Mathematics. – Elsevier, 1979. – Т. 5. – С. 287-326.
127. Gunarathne, T., Wu, T. L., Qiu, J., Fox, G. MapReduce in the Clouds for Science // 2010 IEEE Second International Conference on Cloud Computing Technology and Science. – IEEE, 2010. – С. 565-572.
128. Gupta, B. D., Palis, M. A. Online real-time preemptive scheduling of jobs with deadlines on multiple machines // Journal of Scheduling. – 2001. – Т. 4. – №. 6. – С. 297-312.
129. GWA - Grid Workloads Archive [online] http://gwa.ewi.tudelft.nl/ – 2021.
130. Hamscher, V., Schwiegelshohn, U., Streit, A., & Yahyapour, R. Evaluation of job-scheduling strategies for grid computing //International Workshop on Grid Computing. – Springer, Berlin, Heidelberg, 2000. – С. 191-202.
131. Heymann, E., Senar, M. A., Luque, E., Livny, M. Self-adjusting scheduling of master-worker applications on distributed clusters // European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2001. – С. 742-751.
132. Hoffman, K. Ready. Steady. Go! The Speed of VM Creation and SSH Acces. [online] http://blog.cloud66.com/ready-steady-go-the-speed-of-vm-creation-and-ssh-key-access-on-aws-digitalocean-linode-vexxhost-google-cloud-rackspace-and-microsoft-azure.
133. Hu, J., Gu, J., Sun, G., Zhao, T. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment // 2010 3rd

International Symposium on Parallel Architectures, Algorithms and Programming. – IEEE, 2010. – С. 89-96.

134. Huang, P., Peng, H., Lin, P., Li, X. Static strategy and dynamic adjustment: an effective method for grid task scheduling // Future Generation Computer Systems. – 2009. – Т. 25. – №. 8. – С. 884-892.
135. Iverson M. A., Ozguner F., Follen G. J. Run-time statistical estimation of task execution times for heterogeneous distributed computing //Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing. – IEEE, 1996. – С. 263-270.
136. Iyer, S., Druschel, P. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O // Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles. – 2001. – С. 117-130.
137. Jansen K., Trystram D. Scheduling parallel jobs on heterogeneous platforms //Electronic Notes in Discrete Mathematics. – 2016. – Т. 55. – С. 9-12.
138. Janssens, N., An, X., Daenen, K., Forlivesi, C. Dynamic scaling of call-stateful SIP services in the cloud // International Conference on Research in Networking. – Springer, Berlin, Heidelberg, 2012. – С. 175-189.
139. Johnson, D. S. Near-optimal bin packing algorithms : дис. – Massachusetts Institute of Technology, 1973.
140. Karatza, H. D. A simulation-based performance analysis of gang scheduling in a distributed system // Proceedings 32nd Annual Simulation Symposium. – IEEE, 1999. – С. 26-33.
141. Kasperski A., Zielinski P. Minmax (regret) scheduling problems //Sequencing and Scheduling with Inaccurate Data. – 2014. – С. 159-210.
142. Kaur, J. Comparison of load balancing algorithms in a cloud // International Journal of Engineering Research and Applications. – 2012. – Т. 2. – №. 3. – С. 1169-1173.
143. Kianpisheh S., Jalili S., Charkari N. M. Predicting job wait time in grid environment by applying machine learning methods on historical information // International Journal of Grid and Distributed Computing. – 2012. – Т. 5. – №. 3.

– C. 11-22.

144. Kim, J. Y. On SIP Server Clusters and the Migration to Cloud Computing Platforms. – Columbia University, 2016.
145. Kokilavani, T., Amalarethinam, D. G. Load balanced min-min algorithm for static meta-task scheduling in grid computing // International Journal of Computer Applications. – 2011. – Т. 20. – №. 2. – С. 43-49.
146. Kuijl A., Emmerich M. T. M., Li H. A novel multi-objective optimization scheme for Grid resource allocation //Proceedings of the 6th international workshop on Middleware for grid computing. – 2008. – С. 1-6.
147. Kurowski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A., Pukacki, J. Dynamic grid scheduling with job migration and rescheduling in the GridLab resource management system // Scientific Programming. – 2004. – Т. 12. – №. 4. – С. 263-273.
148. Laredo, J. J., Dorronsoro, B., Pecero, J., Bouvry, P., Durillo, J. J., Fernandes, C. Designing a self-organized approach for scheduling bag-of-tasks // 2012 Seventh International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. – IEEE, 2012. – С. 315-320.
149. LD, D. B., Krishna, P. V. Honey bee behavior inspired load balancing of tasks in cloud computing environments // Applied Soft Computing. – 2013. – Т. 13. – №. 5. – С. 2292-2303.
150. Lee, C. B., Schwartzman, Y., Hardy, J., & Snavely, A. Are user runtime estimates inherently inaccurate? //Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 2004. – С. 253-263.
151. Lee, L. T., Liang, C. H., Chang, H. Y. An adaptive task scheduling system for Grid Computing // The Sixth IEEE International Conference on Computer and Information Technology (CIT'06). – IEEE, 2006. – С. 57-57.
152. Lee, Y. H., Leu, S., Chang, R. S. Improving job scheduling algorithms in a grid environment // Future generation computer systems. – 2011. – Т. 27. – №. 8. – С. 991-998.

153. Lepère, R., Mounié, G., Trystram, D. An approximation algorithm for scheduling trees of malleable tasks // European Journal of Operational Research. – 2002. – Т. 142. – №. 2. – С. 242-249.
154. Lepere, R., Mounie, G., Trystram, D., ROBIČ, B. Malleable tasks: An efficient model for solving actual parallel applications // Parallel Computing: Fundamentals and Applications. – 2000. – С. 598-605.
155. Leung, J. Y. T. (ed.). Handbook of scheduling: algorithms, models, and performance analysis. – CRC press, 2004.
156. Lezama Barquet A., Tchernykh A., Yahyapour R. Performance evaluation of infrastructure as service clouds with SLA constraints // Computación y Sistemas. – 2013. – Т. 17. – №. 3. – С. 401-411.
157. Li, H., Buyya, R. Model-based simulation and performance evaluation of grid scheduling strategies // Future Generation Computer Systems. – 2009. – Т. 25. – №. 4. – С. 460-465.
158. Li, Y., Tang, X., Cai, W. Dynamic bin packing for on-demand cloud resource allocation // IEEE Transactions on Parallel and Distributed Systems. – 2015. – Т. 27. – №. 1. – С. 157-170.
159. Liu, H., Liu, S., Meng, X., Yang, C., Zhang, Y. LBVS: A load balancing strategy for virtual storage // 2010 International Conference on Service Sciences. – IEEE, 2010. – С. 257-262.
160. Liu, X., Pan, L., Wang, C. J., Xie, J. Y. A lock-free solution for load balancing in multi-core environment // 2011 3rd International Workshop on Intelligent Systems and Applications. – IEEE, 2011. – С. 1-4.
161. Lloyd, E. L. Concurrent task systems // Operations Research. – 1981. – Т. 29. – №. 1. – С. 189-201.
162. Lu, Y., Panneerselvam, J., Liu, L., Wu, Y. RVLBPNN: A workload forecasting model for smart cloud computing // Scientific Programming. – 2016. – Т. 2016.
163. Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J. R., Greenberg, A. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services // Performance Evaluation. – 2011. – Т. 68. – №. 11. – С. 1056-1071.

164. Mahjoub A., Sánchez J. E. P., Trystram D. Scheduling with uncertainties on new computing platforms // Computational Optimization and Applications. – 2011. – Т. 48. – №. 2. – С. 369-398.

165. Mao, M., Humphrey, M. A performance study on the vm startup time in the cloud // 2012 IEEE Fifth International Conference on Cloud Computing. – IEEE, 2012. – С. 423-430.

166. Mazalek, A., Vranova, Z., Stankova, E. Analysis of the impact of IPSec on performance characteristics of VoIP networks and voice quality // International Conference on Military Technologies (ICMT) 2015. – IEEE, 2015. – С. 1-5.

167. McCann, C., Vaswani, R., Zahorjan, J. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors // ACM Transactions on Computer Systems (TOCS). – 1993. – Т. 11. – №. 2. – С. 146-178.

168. Megow N., Uetz M., Vredeveld T. Models and algorithms for stochastic online scheduling // Mathematics of Operations Research. – 2006. – Т. 31. – №. 3. – С. 513-525.

169. Megow N., Vredeveld T. Approximation in preemptive stochastic online scheduling // European Symposium on Algorithms. – Springer, Berlin, Heidelberg, 2006. – С. 516-527.

170. Mehta, H., Kanungo, P., Chandwani, M. Decentralized content aware load balancing algorithm for distributed computing environments //Proceedings of the International Conference & Workshop on Emerging Trends in Technology. – 2011. – С. 370-375.

171. MIXvoip 2021. [online] https://www.mixvoip.com/.

172. Montazerolghaem, A., Yaghmaee, M. H., Leon-Garcia, A., Naghibzadeh, M., Tashtarian, F.A load-balanced call admission controller for IMS cloud computing // IEEE Transactions on Network and Service Management. – 2016. – Т. 13. – №. 4. – С. 806-822.

173. Montoro, P., Casilari, E. A comparative study of VoIP standards with asterisk // 2009 Fourth International Conference on Digital Telecommunications. – IEEE, 2009. – С. 1-6.

174. Muraña J. et al. Characterization, modeling and scheduling of power consumption of scientific computing applications in multicores //Cluster Computing. – 2019. – Т. 22. – №. 3. – С. 839-859.

175. Nae, V., Prodan, R., Fahringer, T. Cost-efficient hosting and load balancing of massively multiplayer online games // 2010 11th IEEE/ACM International Conference on Grid Computing. – IEEE, 2010. – С. 9-16.

176. Nakai, A. M., Madeira, E., Buzato, L. E. Load balancing for internet distributed services using limited redirection rates // 2011 5th Latin-American Symposium on Dependable Computing. – IEEE, 2011. – С. 156-165.

177. Naroska E., Schwiegelshohn U. On an on-line scheduling problem for parallel jobs //Information Processing Letters. – 2002. – Т. 81. – №. 6. – С. 297-304.

178. Nguyen, T. D., Vaswani, R., Zahorjan, J. Parallel application characterization for multiprocessor scheduling policy design // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1996. – С. 175-199.

179. Nguyen, T. D., Vaswani, R., Zahorjan, J. Using runtime measured workload characteristics in parallel processor scheduling // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1996. – С. 155-174.

180. Ni, J., Huang, Y., Luan, Z., Zhang, J., Qian, D. Virtual machine mapping policy based on load balancing in private cloud environment // 2011 International Conference on Cloud and Service Computing. – IEEE, 2011. – С. 292-295.

181. Pascual F., Rzadca K., Trystram D. Cooperation in multi-organization scheduling. // In European Conference on Parallel Processing. Springer. – 2007. – С. 224-233.

182. Patel P., Ranabahu A. H., Sheth A. P. Service level agreement in cloud computing // International Conference on Object Oriented Programming, Systems (OOPSLA’09). – 2009.

183. Pinedo M. Scheduling: Theory, Algorithms, and Systems // Prentice-Hall, New Jersey. – 2002.

184. PWA - Parallel Workloads Archive, 2021. [online] http://www.cs.huji.ac.il/labs/parallel/ workload/
185. Radojević, B., Žagar, M. Analysis of issues with load balancing algorithms in hosted (cloud) environments // 2011 Proceedings of the 34th International Convention MIPRO. – IEEE, 2011. – С. 416-420.
186. Rahman, M., Ranjan, R., Buyya, R., Benatallah, B. A taxonomy and survey on autonomic management of applications in grid computing environments // Concurrency and Computation: Practice and Experience. – 2011. – Т. 23. – №. 16. – С. 1990-2019.
187. Ramezani, F., Lu J., Hussain, F. K. Task-based system load balancing in cloud computing using particle swarm optimization // International Journal of Parallel Programming. – 2014. – Т. 42. – №. 5. – С. 739-754.
188. Randles, M., Lamb, D., Taleb-Bendiab, A. A comparative study into distributed load balancing algorithms for cloud computing // 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops. – IEEE, 2010. – С. 551-556.
189. Rapine, C., Scherson, I. D., Trystram, D. On-line scheduling of parallelizable jobs // European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 1998. – С. 322-327.
190. Razavi, K., Razorea, L. M., Kielmann, T. Reducing vm startup time and storage costs by vm image content consolidation // European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2013. – С. 75-84.
191. Robertazzi T. G., Yu D. Multi-source grid scheduling for divisible loads //2006 40th Annual Conference on Information Sciences and Systems. – IEEE, 2006. – С. 188-191.
192. Rodero, I., Guim, F., Corbalan, J., Fong, L., Sadjadi, S. M. Grid broker selection strategies using aggregated resource information // Future Generation Computer Systems. – 2010. – Т. 26. – №. 1. – С. 72-86.
193. Rosti, E., Smirni, E., Serazzi, G., Dowdy, L. W. Analysis of non-work-conserving processor partitioning policies // Workshop on Job Scheduling

Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1995. – C. 165-181.

194. Samal, P., Mishra, P. Analysis of variants in Round Robin Algorithms for load balancing in Cloud Computing // International Journal of Computer Science and Information Technologies. – 2013. – Т. 4. – №. 3. – С. 416-419.
195. Schwiegelshohn U., Yahyapour R. Attributes for communication between grid scheduling instances //Grid Resource Management. – Springer, Boston, MA, 2004. – С. 41-52.
196. Sethi S., Sahu A., Jena S. K. Efficient load balancing in cloud computing using fuzzy logic //IOSR Journal of Engineering. – 2012. – Т. 2. – №. 7. – С. 65-71.
197. Shmoys D. B., Wein J., Williamson D. P. Scheduling parallel machines on-line // SIAM Journal on Computing. – 1995. – Т. 24. – №. 6. – С. 1313-1331.
198. Simionovici, A. M., Tantar, A. A., Bouvry, P., Tchernykh, A., Cortés-Mendoza, J. M., Didelot, L.VoIP traffic modelling using Gaussian mixture models, Gaussian processes and interactive particle algorithms // 2015 IEEE Globecom Workshops (GC Wkshps). – IEEE, 2015. – С. 1-6.
199. Simionovici, A. M., Tantar, A., Bouvry, P., Didelot, L. Predictive modeling in a voip system // Journal of Telecommunications and Information Technology. – 2013. – Т. 4. – С. 32-40.
200. Singh, A., Korupolu, M., Mohapatra, D. Server-storage virtualization: integration and load balancing in data centers // SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. – IEEE, 2008. – С. 1-12.
201. Sinnen, O., Sousa, L. Exploiting unused time slots in list scheduling considering communication contention // European Conference on Parallel Processing. – Springer, Berlin, Heidelberg, 2001. – С. 166-170.
202. Sleator, R. E. Tarjan, Amortized efficiency of list update and paging rules, Comm // Assoc. Comput. Mach. – 1985. – Т. 28. – С. 202-208.
203. Smith W., Foster I., Taylor V. Predicting application run times using historical information //Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1998. – С. 122-142.

204. Sobalvarro, P. G., Weihl, W. E. Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1995. – С. 106-126.
205. Song, H. J., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., Chien, A. The microgrid: a scientific tool for modeling computational grids // SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing. – IEEE, 2000. – С. 53-53.
206. Sotskov Y. N., Werner F. Sequencing and scheduling with inaccurate data // Sequencing and Scheduling with Inaccurate Data. – 2014. – С. 1-432.
207. Stanojevic, R., Shorten, R. Load balancing vs. distributed rate limiting: an unifying framework for cloud control // 2009 IEEE International Conference on Communications. – IEEE, 2009. – С. 1-6.
208. Taylor, I. J., Deelman, E., Gannon, D. B., Shields, M. Workflows for e-Science: scientific workflows for grids. – Springer-Verlag, 2006.
209. Tchernykh, A., Babenko, M., Chervyakov, N., Cortés-Mendoza, J. M., Kucherov, N., Miranda-López, V., Deryabin, M., Dvoryaninova, I., & Radchenko, G. Towards mitigating uncertainty of data security breaches and collusion in cloud computing //2017 28th International Workshop on Database and Expert Systems Applications (DEXA). – IEEE, 2017. – С. 137-141.
210. Trenz M., Huntgeburth J. C., Veit D. J. The role of uncertainty in cloud computing continuance: antecedents, mitigators, and consequences. ECIS 2013: 147 – 2013. – С. 1–12.
211. Tsafrir, D., Etsion, Y., Feitelson, D. G. Backfilling using system-generated predictions rather than user runtime estimates // IEEE Transactions on Parallel and Distributed Systems. – 2007. – Т. 18. – №. 6. – С. 789-803.
212. Ullman, J. D. The performance of a memory allocation algorithm., Princeton University, Department of Electrical Engineering // Computer Science Laboratory. – 1971. – Т. 47.
213. Vredeveld T. Stochastic online scheduling // Computer Science-Research and Development. – 2012. – Т. 27. – №. 3. – С. 181-187.

214. Wang, F., Papaefthymiou, M., Squillante, M. Performance evaluation of gang scheduling for parallel and distributed multiprogramming // Workshop on Job Scheduling Strategies for Parallel Processing. – Springer, Berlin, Heidelberg, 1997. – С. 277-298.

215. Wang, S. C., Yan, K. Q., Liao, W. P., Wang, S. S. Towards a load balancing in a three-level cloud computing network // 2010 3rd international conference on computer science and information technology. – IEEE, 2010. – Т. 1. – С. 108-113.

216. Wolke, A., Tsend-Ayush, B., Pfeiffer, C., Bichler, M. More than bin packing: Dynamic resource allocation strategies in cloud data centers // Information Systems. – 2015. – Т. 52. – С. 83-95.

217. Yarmolenko, V., Sakellariou, R. An evaluation of heuristics for SLA based parallel job scheduling // Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. – IEEE, 2006. – С. 8 pp.

218. Ye D., Mei L. On-Line scheduling of parallel jobs in heterogeneous multiple clusters //Frontiers in Algorithmics and Algorithmic Aspects in Information and Management. – Springer, Berlin, Heidelberg, 2012. – С. 139-148.

219. Yu, J., Buyya R. A taxonomy of workflow management systems for grid computing // Journal of grid computing. – 2005. – Т. 3. – №. 3-4. – С. 171-200.

220. Zhang, Z., Zhang, X. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation // 2010 The 2nd International Conference on Industrial Mechatronics and Automation. – IEEE, 2010. – Т. 2. – С. 240-243.

221. Zhao, Y., Huang, W. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud // 2009 Fifth International Joint Conference on INC, IMS and IDC. – IEEE, 2009. – С. 170-175.

222. Zikos S., Karatza H. D. Resource allocation strategies in a 2-level hierarchical grid system //41st Annual Simulation Symposium (anss-41 2008). – IEEE, 2008. – С. 157-164.

223. Zitzler, E. Evolutionary algorithms for multiobjective optimization: Methods and applications. – Ithaca: Shaker, 1999. – T. 63.

Приложение 1. Регистрация программ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017660880

Программа моделирования работы устройств концепции интернет вещей на базе системы остаточных классов

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет» (RU)***

Авторы: ***Червяков Николай Иванович (RU), Бабенко Михаил Григорьевич (RU), Черных Андрей Николаевич (RU), Кучеров Николай Николаевич (RU), Ващенко Ирина Сергеевна (RU)***

Заявка № **2017617582**
Дата поступления **31 июля 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***28 сентября 2017 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018612694

Модуль оценки рисков безопасности облачных, краевых и туманных вычислений в условиях вычислительной неопределенности

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет» (RU)***

Авторы: ***Червяков Николай Иванович (RU), Бабенко Михаил Григорьевич (RU), Черных Андрей Николаевич (RU), Кучукова Наталья Николаевна (RU), Гудиева Наталья Григорьевна (RU)***

Заявка № **2017663493**
Дата поступления **25 декабря 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***21 февраля 2018 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019610639

Среда моделирования адаптивной цифровой фильтрации в системе остаточных классов

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет» (RU)***

Авторы: ***Бабенко Михаил Григорьевич (RU), Червяков Николай Иванович (RU), Черных Андрей Николаевич (RU), Кучуков Виктор Андреевич (RU), Кучеров Николай Николаевич (RU), Кучукова Екатерина Андреевна (RU), Гудиева Наталья Григорьевна (RU)***

Заявка № **2018665316**
Дата поступления **28 декабря 2018 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***15 января 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2019611375

Распределенная система надежного хранения и обработки данных в мультиоблачной среде

Правообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования «Северо-Кавказский федеральный университет» (RU)***

Авторы: ***Бабенко Михаил Григорьевич (RU), Червяков Николай Иванович (RU), Черных Андрей Николаевич (RU), Кучеров Николай Николаевич (RU), Кучуков Виктор Андреевич (RU), Кучукова Екатерина Андреевна (RU), Аль-Гальда Сафват Чиад (IQ)***

Заявка № **2018665335**
Дата поступления **28 декабря 2018 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***24 января 2019 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

Приложение 2. Патент

РОССИЙСКАЯ ФЕДЕРАЦИЯ

ПАТЕНТ

НА ИЗОБРЕТЕНИЕ

№ 2744815

Устройство для перевода чисел из системы остаточных классов и расширения оснований

Патентообладатель: ***Федеральное государственное автономное образовательное учреждение высшего образования "Северо-Кавказский федеральный университет" (RU)***

Авторы: ***Бабенко Михаил Григорьевич (RU), Кучуков Виктор Андреевич (RU), Черных Андрей Николаевич (RU), Кучеров Николай Николаевич (RU)***

Заявка № **2020120649**
Приоритет изобретения **22 июня 2020 г.**
Дата государственной регистрации
в Государственном реестре изобретений
Российской Федерации **16 марта 2021 г.**
Срок действия исключительного права
на изобретение истекает **22 июня 2040 г.**

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев