МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ, ЧИСЛЕННЫЕ МЕТОДЫ И КОМПЛЕКСЫ ПРОГРАММ/MATHEMATICAL MODELING, NUMERICAL METHODS AND PROGRAM COMPLEXES

DOI: https://doi.org/10.60797/IRJ.2025.161.56

ОБЗОР АЛГОРИТМОВ ПЛАНИРОВЩИКОВ ПАРАЛЛЕЛЬНЫХ ЗАДАЧ: ЭВОЛЮЦИЯ, КЛАССИФИКАЦИЯ И СОВРЕМЕННЫЕ ПОДХОДЫ

Обзор

Матвеев Г.А.[1,] *, Осипов В.И.[2], Роганов В.А.[3]
[1] ORCID : 0000-0002-2988-1791;
[2] ORCID : 0000-0002-8983-3791;
[3] ORCID : 0000-0002-5500-0804;
[1, 2, 3] Институт программных систем им. А.К. Айламазяна РАН, Переславль-Залесский, Российская Федерация
[3] Научно-исследовательский институт механики Московского государственного университета им. М.В. Ломоносова, Москва, Российская Федерация

* Корреспондирующий автор (gera[at]prime.botik.ru)

Аннотация

Планирование параллельных задач является одной из фундаментальных проблем в области высокопроизводительных вычислений, распределённых систем и облачных технологий. Данная задача относится к классу NP-трудных, что исключает возможность получения точного оптимального решения для больших входных данных за полиномиальное время. Поэтому особое внимание исследователей привлекают приближённые, эвристические и метаэвристические методы, которые обеспечивают приемлемое качество решений в разумные сроки. В статье представлен обзор существующих подходов к планированию: от классических эвристических алгоритмов (HEFT, CPOP, PEFT, DSC) и метаэвристик (генетические алгоритмы, муравьиные колонии, роевой интеллект) до динамических методов (work-stealing, gang scheduling), промышленных планировщиков (Borg, Omega, Mesos, Kubernetes) и новейших решений на основе машинного обучения и обучения с подкреплением (Decima, RL-подходы). Рассматриваются вопросы оркестрации — управления жизненным циклом приложений и контейнеров в облачных средах. Подчёркивается связь теоретических моделей с реальными промышленными системами, такими как Google Borg, Apache Mesos и Kubernetes, которые стали индустриальными стандартами. Уделено внимание тестовым бенчмаркам, следам из реальных дата-центров (Google Borg, Microsoft Azure), а также открытым проблемам и перспективам дальнейших исследований.

Ключевые слова: планирование параллельных задач, распределённые системы, классификация алгоритмов планирования, математическая модель, ориентированный ациклический граф.

OVERVIEW OF PARALLEL TASK SCHEDULER ALGORITHMS: EVOLUTION, CLASSIFICATION, AND MODERN APPROACHES

Review article

Matveev G.A.[1,] *, Osipov V.I.[2], Roganov V.A.[3]
[1] ORCID : 0000-0002-2988-1791;
[2] ORCID : 0000-0002-8983-3791;
[3] ORCID : 0000-0002-5500-0804;
[1, 2, 3] Ailamazyan Program Systems Institute of the Russian Academy of Sciences, Pereslavl-Zalessky, Russian Federation
[3] Research Institute of Mechanics, Lomonosov Moscow State University, Moscow, Russian Federation

* Corresponding author (gera[at]prime.botik.ru)

Abstract

Scheduling parallel tasks is one of the fundamental challenges in the field of high-performance computing, distributed systems, and cloud technologies. This problem belongs to the NP-hard class, which excludes the possibility of obtaining an accurate optimal solution for large input data in polynomial time. Therefore, researchers pay special attention to approximate, heuristic and metaheuristic methods that provide acceptable quality solutions in a reasonable time. The article provides an overview of existing approaches to planning: from classical heuristic algorithms (HEFT, CPOP, PEFT, DSC) and metaheuristics (genetic algorithms, ant colonies, swarm intelligence) to dynamic methods (work-stealing, gang scheduling), industrial planners (Borg, Omega, Mesos, Kubernetes) and the latest solutions based on machine learning and reinforcement learning (Decima, RL approaches). The issues of orchestration — lifecycle management of applications and containers in cloud environments are considered. The connection of theoretical models with real industrial systems such as Google Borg, Apache Mesos and Kubernetes, which have become industry standards, is emphasized. Attention is paid to test benchmarks, traces from real data centers (Google Borg, Microsoft Azure), as well as open problems and prospects for further research.

Keywords: parallel task planning, distributed systems, classification of planning algorithms, mathematical model, directed acyclic graph.

Введение

1.1. Исторический контекст

Проблема планирования вычислений возникла ещё в 1960–1970-х годах вместе с первыми многопроцессорными системами. В то время исследователи столкнулись с необходимостью распределения множества задач между несколькими процессорами. Постановка оказалась схожей с задачами теории расписаний, где оптимизация критерия (например, минимизация общего времени завершения работ) приводила к NP-полным задачам [1]. Уже в 1972 г. Кофман и Грэм [2] показали сложность оптимального планирования даже для двух процессоров.

1.2. Современное значение

Сегодня планировщики являются ключевыми компонентами не только суперкомпьютеров, но и облачных дата-центров, распределённых вычислительных платформ и систем обработки больших данных. Например, дата-центр Google Borg ежедневно обслуживает миллионы запросов и управляет сотнями тысяч серверов. Эффективность работы планировщика напрямую влияет на:

– Производительность системы. Примеры параметров:

а) «Продолжительность расписания» или «суммарное (общее) время выполнения всех задач» (от англ. Makespan).

б) «Пропускная способность» (число задач за единицу времени) (от англ. Throughput).

– Соблюдение SLA (от англ. Service Level Agreement) — «соглашение об уровне сервиса». Это контракт между клиентом и поставщиком услуг, где описано, какое качество сервиса гарантируется (например, задержка системы с ответом на запрос клиента не выше 200 мс).

Сюда относятся такие параметры, как:

а) «Задержка в хвосте распределения» (от англ. Tail latency). Это не среднее время ответа, а время ответа для самых «медленных» запросов (например, 99-й процентиль задержек). В системах, использующих облачные вычисления, даже если среднее время ответа быстрое, пользователи с редкими долгими задержками будут недовольны.

б) «Качество обслуживания» QoS (от англ. Quality of Service). Это метрики, характеризующие надёжность, скорость, задержку, пропускную способность и т.д.

– Энергоэффективность вычислительной системы (снижение потребления энергии дата-центрами) [3].

– Стоимость эксплуатации вычислительной системы.

1.3. Объективные трудности

Несмотря на десятилетия исследований, задача оптимального планирования вычислений остаётся открытой. Причины заключаются в следующем:

NP-трудность классических постановок или задач. NP-задачи (от англ. Nondeterministic Polynomial Tasks) — это класс задач, для которых проверить правильность готового решения можно за недетерминированное полиномиальное время. NP-трудные задачи — это задачи, которые не проще самых сложных задач из класса NP. Обычно они не имеют быстрого точного решения, и приходится использовать приближённые/эвристические методы.

Разнообразие архитектур — от гомогенных CPU-систем с центральным процессором (от англ. CPU, Central Processing Unit) до гетерогенных кластеров с графическими процессорами (от англ. GPU, Graphics Processing Unit), тензорными процессорами (от англ, TPU, Tensor Processing Unit) и специализированными ускорителями.

Динамичность нагрузки в облачных сервисах, где пользователи создают и завершают контейнеры в реальном времени.

Многокритериальность — оптимизация не только времени выполнения, но и энергопотребления, справедливости распределения ресурсов (от англ. Fairness), уровня сервиса SLA (от англ. SLA, Service Level Agreement).

1.4. Цель статьи

Актуальность исследуемой темы основана на следующих аспектах:

– Повышенный интерес компаний и рост популярности систем планирования, использующих такие инструменты, как искусственный интеллект (ИИ), машинное обучение и нейронные сети для оптимизации процессов обработки данных, бизнес-процессов и повышения эффективности деятельности компаний.

– Единых стандартов для систем планирования не существует, поэтому каждая крупная компания адаптирует такие системы под свои нужды и запросы. Существуют разные подходы к методам внедрения инструментов планирования в большие системы.

– Технологии в планировании выполнения задач постоянно развиваются и совершенствуются. Необходимо иметь максимально полное представление о текущей ситуации с уже существующими системами планирования, работающими технологиями и влиянии таких систем на производственные и бизнес-процессы.

Данная работа имеет целью систематизировать современные исследования, выделить основные подходы и проследить эволюцию алгоритмов планирования. Обзор, представленный в работе, основан на анализе материалов и литературы о существующих системах планирования и призван актуализировать знания в этой области. Для подготовки обзора просматривались следующие базы данных в области компьютерных наук: ACM Digital Library, IEEE Xplore Digital Library, Springer Lecture Notes in Computer Science (LNCS), материалы из журналов международной издательской компании Springer-Verlag. Кроме того, просматривались материалы по различным конференциям, симпозиумам на тему управления и планирования в параллельных и распределённых системах. Период поиска начинали с 1970-х годов, когда стали появляться первые многопроцессорные системы, и вместе с ними появилась проблема планирования вычислений между несколькими процессорами, по настоящее время. Ключевые слова, которые использовали при поиске: parallel task planning, distributed systems, classification of planning algorithms, task scheduling, Google Borg, Omega, Sparrow, Apache Mesos, Kubernetes, scheduling algorithms for heterogeneous computing systems, task scheduling algorithms, scheduling problems, metaheuristics for scheduling, dynamic task scheduling. Обзор может быть полезен в компаниях для принятия решений по внедрению технологий оптимального планирования выполнения задач.

Математическая модель задачи

2.1. Представление задач

Параллельная программа моделируется ориентированным ациклическим графом DAG (от англ. Directed Acyclic Graph), в котором вершины соответствуют задачам, а рёбра — взаимосвязям с другими задачами. Время выполнения задачи зависит от технических параметров вычислительной системы (например, от выбранного процессора, от объёма памяти).

В ориентированном ациклическом графе узлы соединяются направленными рёбрами, которые указывают направление от одной вершины к другой. Такой граф не содержит петель или циклов.

Ориентированные ациклические графы используются в различных областях науки.

– В распределённых вычислениях, когда необходимо управлять задачами, которые выполняются параллельно и зависят друг от друга, например, по входным и выходным данным.

– При планировании задач, когда каждая задача представляется в виде узла графа с направленными рёбрами.

– При проектировании компиляторов, анализаторов данных.

2.2. Формулировка оптимизационных критериев

Можно выделить следующие оптимизационные критерии работы вычислительной системы или дата-центра:

– Минимизация времени завершения всех задач (от англ. Makespan).

– Максимизация числа задач, выполненных за единицу времени (от англ. Throughput).

– Минимизация задержки выполнения отдельных запросов (от англ. Latency).

– Минимизация энергопотребления вычислительной системы (от англ. Energy).

– Обеспечение заданного уровня качества обслуживания (от англ. QoS и SLA).

2.3. Сложность задачи

По результатам исследований Гэри и Джонсона, даже для простых случаев задача минимизации времени завершения всех задач (от англ. Makespan) является NP-полной. Это означает невозможность нахождения точного решения для больших входных данных за полиномиальное время. Поэтому, на практике используются приближённые алгоритмы.

2.4. Связь с другими задачами

Задачи планирования связаны с другими задачами:

– Задачей укладки в контейнеры (от англ. Bin Packing Problem). Задача укладки в контейнеры состоит в следующем: есть несколько одинаковых контейнеров (ящиков) фиксированного размера и есть предметы разных размеров. Нужно разместить все предметы по контейнерам так, чтобы: они поместились в контейнеры и, при этом, использовалось минимальное количество контейнеров. Главный акцент: минимизация числа контейнеров для всех предметов. Можно привести аналогию с планированием выполнения задач, где процессоры — это «ящики», а задачи с разным временем выполнения — «предметы». Задача состоит в том, чтобы задействовать минимальное количество процессоров для выполнения всех задач.

– Задачей рюкзака (от англ. Knapsack Problem). Задача о рюкзаке состоит в следующем: есть один рюкзак ограниченной вместимости и есть набор предметов. Каждый предмет имеет: вес (стоимость ресурса), ценность (польза). Нужно выбрать подмножество предметов так, чтобы: их суммарный вес не превышал вместимости рюкзака, и суммарная ценность предметов была максимальной. Главный акцент: максимизация ценности при ограниченном ресурсе. Например: нужно выбрать, какие задачи запускать на сервере, если ресурсов на все задачи не хватает.

– Классическими задачами теории расписаний (от англ. Scheduling Theory).

Классификация алгоритмов планирования

Существует множество классификаций алгоритмов планирования. Наиболее распространённые (Sinnen [4], Casavant & Kuhl [5]) из них следующие:

3.1. По времени принятия решений

Алгоритмы планирования по времени принятия решений бывают:

– Статические — расписание строится заранее, до начала выполнения. Применимы для научных workflow-систем, где известны все зависимости и времена выполнения.

– Динамические — решения принимаются в процессе работы, учитывают реальную загрузку. Используются в облачных и многопользовательских системах.

3.2. По архитектуре системы

Алгоритмы планирования по архитектуре вычислительной системы бывают:

– Централизованные — один планировщик для всего кластера.

– Децентрализованные — множество локальных планировщиков, которые синхронизируются между собой.

3.3. По учёту ресурсов

Алгоритмы планирования по учёту вычислительных ресурсов учитывают аппаратные ресурсы вычислительной системы. Например:

– Объём оперативной памяти.

– Тип процессоров. Процессоры бывают:

а) Гомогенные — все процессоры в вычислительной системе одинаковые.

б) Гетерогенные — узлы вычислительной системы различаются по производительности, наличию GPU/TPU и т.д.

– Объём хранилища данных (памяти жёстких дисков).

– Устройства ввода-вывода.

– Используемые системные платы.

– Сетевые устройства и устройства, обеспечивающие высокоскоростную передачу данных между узлами вычислительной системы (например, Gigabit Ethernet, InfiniBand)

3.4. По оптимизируемым метрикам

Можно выделить следующие метрики:

– «Продолжительность расписания» или «суммарное время выполнения всех задач» (от англ. Makespan).

– Справедливость распределения ресурсов (от англ. Fairness). Означает, что ресурсы распределяются честно, без сильного перекоса в пользу одних задач и ограничения в ресурсах других.

– «Задержка в хвосте распределения» — это время ответа системы для самых «медленных» запросов (от англ. Tail latency).

– «Планирование с учётом энергопотребления» (от англ. Energy-aware scheduling).

Классические эвристики

4.1. Список методов

Классические эвристические алгоритмы включают [6], [7], [8]:

– HEFT (от англ. Heterogeneous Earliest Finish Time) — «гетерогенное самое раннее время завершения». Эвристика, которая назначает задачи так, чтобы минимизировать их время завершения в системах с разными процессорами.

– CPOP (от англ. Critical Path on a Processor) — «критический путь на процессоре». Алгоритм, который сначала планирует критический путь задач и, затем, закрепляет его за одним процессором.

– DSC (от англ. Dominant Sequence Clustering) — «кластеризация доминирующей последовательности». Алгоритм, группирующий связанные задачи в кластеры.

– PEFT (от англ. Predict Earliest Finish Time) — «прогнозируемое самое раннее время завершения». Улучшение алгоритма HEFT с прогнозом времени выполнения.

Каждый из них строит расписание на основе анализа критических путей, приоритетов задач и времени передачи данных.

4.2. Алгоритм HEFT

Алгоритм HEFT является одним из наиболее популярных алгоритмов, благодаря простоте и высокой эффективности. Его шаги:

Вычисление приоритетов задач (от англ. Upward Rank) — ранжирование задач по «важности». Это эвристическая величина, которая определяет оценку «важности» задачи в ориентированном ациклическом графе DAG (от англ. Directed Acyclic Graph).

Последовательное назначение задач на процессоры с минимальным временем завершения.

Эксперименты показали, что алгоритм HEFT даёт близкие к оптимальным решения при умеренных накладных расходах.

4.3. Алгоритм CPOP

Алгоритм CPOP выделяет критический путь в графе задач и назначает его на один процессор, чтобы минимизировать задержки передачи данных. Оставшиеся задачи распределяются аналогично алгоритму HEFT.

4.4. Алгоритм PEFT

Алгоритм PEFT улучшает алгоритм HEFT, добавляя прогнозирование времени завершения задач на основе матрицы предпочтений. Это даёт более качественные расписания при небольшом увеличении вычислительной сложности.

Метаэвристики

Метаэвристики — это высокоуровневые стратегии поиска, которые используют и направляют другие эвристики для эффективного поиска решения задачи оптимизации или задачи машинного обучения. Метаэвристические алгоритмы, как правило, являются приближёнными. Источником информации, принципов и подходов для разработки вычислительных систем, предназначенных для решения сложных вычислительных задач, является природа. Такие метаэвристики включают в себя генетические алгоритмы, оптимизацию колоний муравьёв, роевой интеллект, методы имитации отжига.

5.1. Генетические алгоритмы

Генетические алгоритмы GA (от англ. Genetic Algorithm) широко применяются для планирования [9]. Они моделируют эволюционный процесс с операциями скрещивания и мутации. Преимущества: способность находить хорошие решения в больших пространствах поиска. Недостатки: значительные вычислительные расходы.

5.2. Муравьиные колонии

Алгоритмы на основе поведения муравьёв [10] строят расписания, моделируя ферромонные пути. Эти методы особенно эффективны для графов DAG среднего размера.

5.3. Роевой интеллект

Алгоритмы роя частиц PSO (от англ. Particle Swarm Optimization) применяются для оптимизации расписаний [11]. Они показывают высокую сходимость, но могут застревать в локальных минимумах. Данные алгоритмы созданы на основе опыта и обобщения результатов наблюдения за поведением стай птиц/рыб.

5.4. Симуляция отжига

Методы имитации отжига [12] позволяют избегать локальных минимумов за счёт вероятностных переходов.

Динамические алгоритмы

6.1. Алгоритм Work-stealing

Алгоритм Work-stealing [13] означает «забрать работу». Он используется в современных языках программирования (например, Cilk, Java Fork/Join) в качестве стратегии планирования для компьютерных программ, использующих потоки. Идея алгоритма состоит в том, что простаивающий поток, у которого нет задач для выполнения, забирает их из очередей других потоков.

6.2. Алгоритм Gang scheduling

Алгоритм Gang scheduling переводится как «групповое планирование». При этом подходе задачи, которые назначаются алгоритмом планирования в группу (gang) связанных процессов, запускаются одновременно на всех выбранных процессорах или на выбранном подмножестве процессоров. Целью такого планирования является одновременный запуск и выполнение всех процессов (потоков) одной задачи, чтобы избежать простоя и блокировок процессов, когда они ждут завершения коммуникаций с другими процессами. Алгоритм Gang scheduling реализован в таких системах, как Kubernetes, Slurm Workload Manager.

6.3. Программирование мультипроцессоров

В работах [14], [15], [16] изучается поведение динамических планировщиков с миграцией задач. В них рассматривается планировщик потоков на уровне пользователя для мультипроцессорных систем с общей памятью и делается анализ его производительности. Цель данных исследований состоит в моделировании организации работы планировщика таким образом, чтобы эффективно использовать ресурсы процессоров, которые предоставляет ядро операционной системы.

Системные планировщики

7.1. Система планирования Google Borg

Система Google Borg — это внутренняя система планирования задач, разработанная в Google в начале 2000-х годов [17]. Google Borg — это менеджер кластеров, который планирует и контролирует выполнение задач, чтобы сделать управление ресурсами для разработчиков простым и удобным, и максимизировать эффективность использования ресурсов в центрах обработки данных. Он обеспечивает управление сотнями тысяч серверов и миллионами задач ежедневно. Характеризуется высокой масштабируемостью, оптимизацией под уровень сервиса SLA.

Архитектура системы Google Borg представлена на рисунке 1. Она включает в себя компоненты, которые обеспечивают её масштабируемость и надёжность. Рассмотрим основные компоненты системы и взаимодействие между ними:

– Ячейка Borg (от англ. Cell) — состоит из: 1) набора машин, управляемых как единица, 2) логически централизованного контроллера, называемого Borgmaster, и 3) процесса-агента, называемого Borglet, который выполняется на каждой машине в ячейке. Каждая ячейка содержит несколько тысяч серверов. Ячейки гетерогенны по процессорным элементам, памяти, ёмкости диска и другим параметрам.

– Борглет (от англ. Borglet) — это локальный процесс-агент Borg, который присутствует на каждой машине в ячейке. Он отвечает за запуск и остановку задач, управление локальными ресурсами.

– Borgmaster — это логически централизованный контроллер. Borgmaster каждой ячейки Cell состоит из двух процессов: основного процесса Borgmaster и отдельного планировщика Scheduler. Основной процесс Borgmaster обрабатывает удалённые вызовы процедур RPC клиентов (от англ. Remote Procedure Call), которые либо изменяют состояние процесса (например, создают задание), либо предоставляют доступ к данным только для чтения (например, задание на поиск). Он также управляет конечными автоматами для всех объектов в системе (для машин, задач и т.д.), взаимодействует с Борглетами (от англ. Borglets) и предлагает веб-интерфейс для взаимодействия с клиентами. Borgmaster может настраиваться через конфигурационный файл borgcfg (от англ. Config File) и управляться через инструменты командной строки (от англ. Command-Line Tools). Логически Borgmaster представляет собой единый процесс, но на самом деле он повторяется несколько раз, т.е. имеет несколько реплик. Каждая реплика сохраняет в памяти копию большей части состояния ячейки, и это состояние также записывается в распределённое хранилище с быстрым доступом на базе Paxos на локальных дисках реплик.

– Планировщик (от англ. Scheduler) назначает задачи машинам на основе их доступности и ограничений по ресурсам. Он находит набор машин, которые удовлетворяют ограничениям задачи, а также имеют достаточно «доступных» ресурсов, включая ресурсы, назначенные для задач с более низким приоритетом, которые могут быть удалены. Планировщик использует отдельный процесс, чтобы работать параллельно с другими репликами Borgmaster.

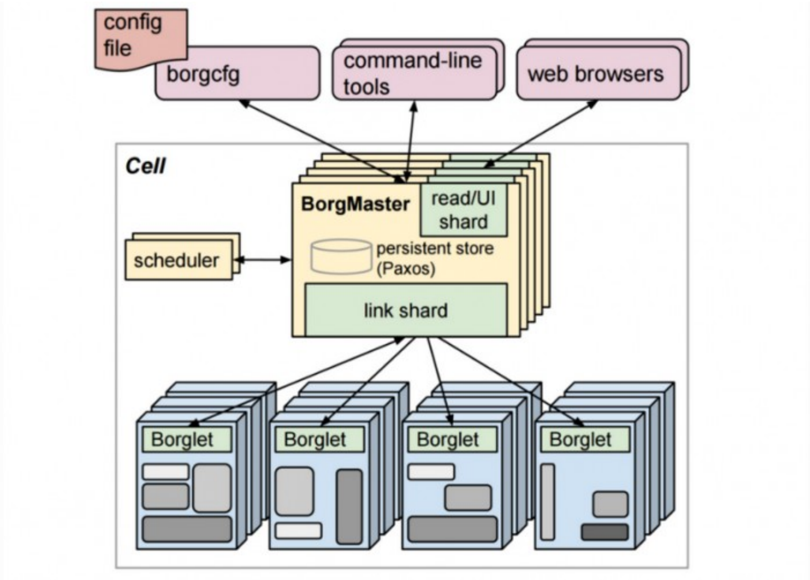

Рисунок 1 - Высокоуровневая архитектура Borg
DOI: https://doi.org/10.60797/IRJ.2025.161.56.1

Основные принципы работы системы Google Borg:

– централизованный планировщик, работающий на кластере;

– SLA (от англ. Service Level Agreement) — «соглашение об уровне сервиса» и приоритеты: критические сервисы с наивысшим приоритетом получают ресурсы первыми;

– изоляция ресурсов между задачами, чтобы одна программа не могла «захватить» весь сервер;

– учёт отказоустойчивости: задачи автоматически перезапускаются на других узлах.

Система Google Borg оказала огромное влияние на индустрию и стала основой для системы Kubernetes.

7.2. Система планирования Omega

Система Omega [18] — это следующая система планирования Google, построенная на децентрализованной архитектуре. В отличие от системы Borg, здесь используется оптимистическое параллельное планирование: несколько планировщиков одновременно составляют расписания, а система разрешает конфликты. Это повышает масштабируемость, снижает узкие места и позволяет обрабатывать тысячи запросов на планирование параллельно.

7.3. Система планирования Sparrow

Система Sparrow [19] использует рандомизацию и выборку для масштабируемого планирования.

Система Sparrow — лёгкий распределённый планировщик для кластеров с короткими задачами (от англ. interactive workloads). Его особенность — использование рандомизированного выбора: запросы ресурсов рассылаются на несколько узлов, и задача назначается туда, где ресурсы освободились быстрее. Такой подход минимизирует задержки (от англ. tail latency) и подходит для сервисов, чувствительных к времени ответа.

7.4. Система Apache Mesos

Система Apache Mesos [20] — фреймворк для кластеров с разделением ресурсов между приложениями.

Система Apache Mesos — это универсальная распределённая система управления ресурсами. Её ключевая идея — «кластер как один большой компьютер». Mesos абстрагирует ресурсы и предоставляет их различным фреймворкам: Hadoop, Spark, MPI, Kubernetes. Планировщики в Mesos используют двухуровневую архитектуру: «мастер» предлагает ресурсы, а отдельные фреймворки решают, как их использовать.

7.5. Система Kubernetes

Система Kubernetes [21] — это открытая система оркестрации контейнеров, ставшая индустриальным стандартом. Оркестрация (от англ. orchestration) — это управление множеством сервисов/контейнеров/задач как единой системой. В отличие от «планирования», которое отвечает за назначение задач на процессоры, оркестрация управляет всем жизненным циклом приложения: запуск, масштабирование, отказоустойчивость, обновления. Система оркестрации контейнеров Kubernetes решает, где запустить контейнеры, следит за их состоянием, перезапускает при сбоях и масштабирует под нагрузку. Она обеспечивает:

– автоматическое размещение контейнеров (от англ. Pods) на узлах кластера;

– масштабирование (увеличение или уменьшение числа контейнеров в зависимости от нагрузки);

– отказоустойчивость (перезапуск контейнеров при сбоях);

– декларативное управление (пользователь описывает желаемое состояние системы, а Kubernetes его поддерживает).

На рисунке 2 представлена архтектура системы Kubernetes. Условно архитектуру Kubernetes можно разделить на два блока. Первый блок является управляющим, сюда входят компоненты, относящиеся к Kubernetes Master: API Server, Scheduler, Controller-Manager и хранилище данных etcd. Управляющий блок управляет рабочими узлами и подами (контейнеризированными приложениями) в кластере, следит за состоянием системы, гарантируя отказоустойчивость и высокую доступность. Компоненты управляющего блока отвечают за такие операции, как планирование задач, а также обрабатывают события кластера, поддерживая всю систему в рабочем состоянии. Второй блок — это рабочие узлы (от англ. Worker Nodes), на которых развёртываются контейнеры. Рабочие узлы запускают контейнеры и обрабатывают запросы от пользователей. Кластер Kubernetes имеет как минимум один рабочий узел.

В состав управляющего блока входят следующие компоненты:

– API Server — это API-сервер, который принимает запросы от пользователей и других компонентов. Он является центральным компонентом управляющего блока Kubernetes, который предоставляет доступ к основному интерфейсу для взаимодействия пользователей с кластером (от англ. API Kubernetes). API-сервер служит шлюзом для всех REST API-запросов к кластеру, обрабатывает аутентификацию, авторизацию и контроль допуска. API-сервер выполняет следующие функции:

1) предоставление интерфейса REST API для создания, обновления и удаления ресурсов (поддержка web-интерфейса);

2) управление состоянием кластера (он напрямую взаимодействует с распределённым хранилищем etcd кластера для сохранения и извлечения данных);

3) поддержка рабочего состояния кластера;

4) обеспечение политики проверки запросов на ресурсы перед их модификацией или сохранением на кластере.

– Scheduler — планировщик, который отслеживает недавно созданные контейнеры (поды) и определяет, на каком рабочем узле их можно запустить.

– Controller-Manager — компонент управляющего блока, который запускает процессы контроллера. Процессы контроллера включают:

1) контроллер узла для отслеживания и реагирования на сбои узла;

2) контроллер репликации для поддержки правильного количества контейнеров (подов) для каждого объекта в системе;

3) контроллер, который связывает сервисы и контейнеры;

4) контроллеры учётных записей.

– etcd — это основное хранилище всех данных кластера Kubernetes. Для доступа к данным кластера используется формат «ключ-значение».

Компоненты рабочего узла работают на каждом узле, поддерживая работу контейнеров (подов) и среды выполнения Kubernetes. Они включают:

– Kubelet — это агент, работающий на каждом узле кластера. Он следит за тем, чтобы контейнеры в поде были запущены. Агент отслеживает состояние контейнеров, отправляя обновления на API-сервер.

– Kube-proxy — это сетевой прокси, работающий на каждом узле кластера, поддерживает сетевые правила на узлах во время сетевых сессий. Компонент управляет сетевыми правилами для связи между подами и внешними сервисами в кластере.

– Container Runtime — это важный компонент, который позволяет системе Kubernetes эффективно запускать контейнеры. Он отвечает за управление выполнением и жизненным циклом контейнеров.

– Docker — это компонент, который обеспечивает загрузку образов и запуск контейнеров.

– Container — контейнер или контейнеризированное приложение. Компоненты в кластере Kubernetes работают с образами контейнеров. Образ контейнера (от англ. Container Image) — это готовый к запуску пакет программного обеспечения вместе с его runtime-зависимостями, содержащий всё необходимое для запуска приложения: код приложения, среду исполнения, системные и прикладные библиотеки, значения по умолчанию всех важных параметров.

– Pod — это группа с контейнерами (контейнеризированными приложениями), содержит в себе один или несколько контейнеров.

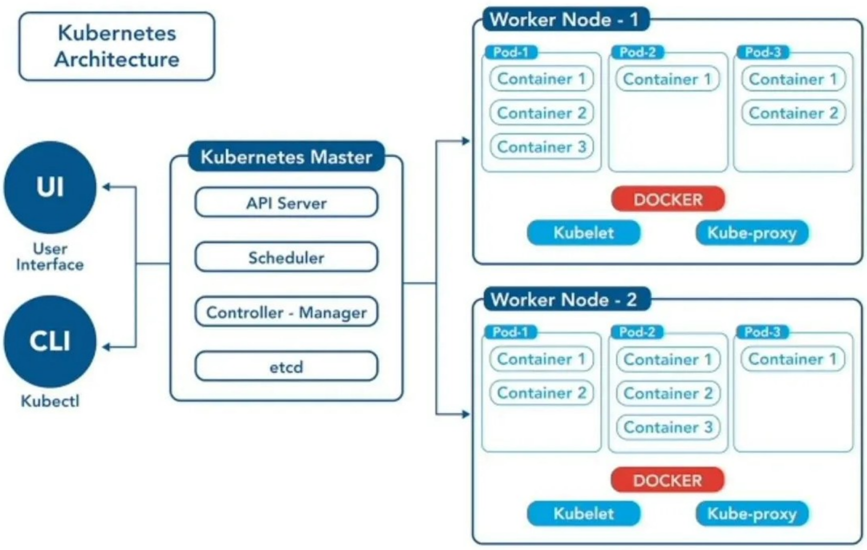

Рисунок 2 - Архитектура системы Kubernetes

DOI: https://doi.org/10.60797/IRJ.2025.161.56.2

В отличие от системы Google Borg, система Kubernetes открыта и активно развивается сообществом. Её планировщик учитывает ресурсы, приоритеты задач и политику качества обслуживания QoS, позволяет задавать правила для размещения контейнеров рядом с другими контейнерами (от англ. Pod Affinity) или правила для предотвращения размещения контейнеров на тех же узлах, где уже работают другие контейнеры (от англ. Pod Anti-Affinity).

Workflow-системы

Workflow-системы — это системы управления рабочими процессами (вычислительными потоками). Они позволяют описывать сложные вычислительные задачи в виде ориентированного ациклического графа DAG и автоматически распределяют их по ресурсам. Примеры таких систем: Pegasus, Kepler, Swift/T, Airflow. Они реализуют собственные планировщики, учитывающие DAG-нагрузку.

Методы планирования на основе машинного обучения

9.1. Decima

Decima [22] — это планировщик, использующий обучение с подкреплением для построения DAG-расписаний. Он обучается на симуляторе, где агент получает вознаграждения за хорошие решения (например, снижение суммарного времени выполнения всех задач). В отличие от эвристик, Decima может адаптироваться к новым типам DAG и нагрузок.

9.2. RL-подходы

RL-подходы (от англ. Reinforcement Learning) — «подходы обучения с подкреплением». Алгоритмы, где агент учится принимать решения, получая вознаграждения. В работах [23], [24] применяются методы Q-learning и Policy gradient:

– Q-learning (от англ. Learning) — это метод обучения, который строит функцию ценности Q(s,a), где *s* — состояние системы, а *a* — действие (назначение задачи). Агент пробует разные действия и получает вознаграждения (от англ. rewards) или штрафы, например:

• вознаграждение за минимальное значение времени выполнения задач (чем меньше время, тем лучше);

• вознаграждение за соблюдение справедливого распределения ресурсов между задачами;

• штраф за несоблюдение соглашения об уровне сервиса (например, задержка с ответом системы на запрос превысила 200 мс).

Постепенно Q-функция сходится к оптимальной стратегии.

– Policy gradient — это множество методов (алгоритмов) градиентного обучения политики агента, которые обучают стратегию (политику) агента напрямую. Для реализации алгоритма используются, например, нейронные сети для параметризации политики. В нейронной сети агент имеет параметризованную политику, которая выдаёт вероятность выбора действия. Параметры политики оптимизируются и обновляются с помощью метода градиентного спуска. Метод на каждой итерации корректирует веса сети так, чтобы максимизировать ожидаемое суммарное вознаграждение и улучшить стратегию агента до оптимальной.

Оба подхода позволяют адаптировать планировщик к меняющимся условиям, что важно в облачных сервисах.

9.3. Прогнозирование нагрузки

Машинное обучение ML (от англ. Machine Learning) применяется для предсказания времени выполнения задач и адаптивного распределения ресурсов.

Некоторые работы используют ML для предсказания времени выполнения задач и оптимизации уровня качества обслуживания QoS (от англ. Quality of Service). Например, планировщик может оценить задержки и заранее перераспределить задачи. В отличие от эвристик, ML-модели могут учитывать нелинейные зависимости и быть «объяснимыми» (от англ. explainable ML), что важно для эксплуатации.

Бенчмарки и тестовые данные

10.1. Синтетические бенчмарки

К синтетическим бенчмаркам (от англ. Synthetic benchmarks) относятся:

– Суммарное время выполнения всех задач (от англ. Makespan).

– «Пропускная способность» (число задач за единицу времени) (от англ. Throughput).

– «Задержка в хвосте распределения» (от англ. Tail latency). Это время ответа системы для самых «медленных» запросов.

– Минимизация энергопотребления (от англ. Energy).

10.2. Следы из реальных систем

Следы (от англ. Traces) — это наборы логов, записывающих реальные события в дата-центрах: запуск задач, использование CPU, RAM, GPU, время жизни процессов, частоту отказов. Следы Google Borg и Microsoft Azure:

– Google Borg traces [25] содержат информацию о миллионах задач, их приоритетах, SLA, нагрузках на узлы.

– Microsoft Azure traces [26] включают данные о поведении облачных клиентов, их нагрузках и изменениях конфигураций.

Такие следы позволяют исследователям проверять алгоритмы планирования в условиях, приближенных к реальным, а не только на синтетических графах.

Открытые проблемы

Существуют следующие проблемы, которые влияют на принятие решений по внедрению технологий оптимального планирования выполнения задач в больших системах:

1. Энергоэффективность вычислительной системы, снижение энергозатрат. Решение проблемы энергоэффективности требует новых подходов к охлаждению вычислительной установки (ВУ). В настоящее время

широко используется жидкостное охлаждение ВУ вместо традиционного воздушного охлаждения. В качестве теплоносителя в жидкостной системе охлаждения используется жидкость с нулевой электропроводностью. Отметим, что, по сравнению со схемами охлаждения, где в качестве теплоносителя используется воздух, у жидкостного охлаждения имеется ряд серьёзных преимуществ:

– данная схема охлаждения требует, как минимум, в 2 раза меньше энергозатрат;

– система охлаждения в вычислителе не содержит ни одной механической подвижной части. Это повышает надёжность ВУ и её эргономические качества (бесшумность).

2. Планирование при неопределённости. При таком планировании планировщик не знает точно:

– время выполнения задач (оно может зависеть от входных данных и конкуренции за ресурсы);

– будущую нагрузку (когда пользователи сделают новые запросы);

– доступность ресурсов (узлы могут выйти из строя, сеть — замедлиться).

Задача состоит в том, чтобы построить расписание, которое будет устойчиво к таким неопределённостям.

3. Explainability ML-моделей — «объяснимость моделей машинного обучения». Возможность понять, почему алгоритм принял то или иное решение.

4. Объединение высокопроизводительных вычислений HPC (от англ. High-Performance Computing), больших данных (от англ. Big Data) и AI-нагрузок. AI-нагрузка (от англ. Artificial Intelligence) — «нагрузка, связанная с задачами искусственного интеллекта» (например, обучение нейросетей, инференс моделей).

5. Мультикритериальная оптимизация работы вычислительной системы или дата-центра.

Заключение

Алгоритмы планировщиков параллельных задач эволюционировали от простых статических методов до сложных подходов, основанных на машинном обучении. В будущем ожидается развитие гибридных решений, объединяющих эвристики, динамику и машинное обучение. Внедрение решений на основе инструментов искусственного интеллекта позволяет более эффективно решать задачи планирования по сравнению с традиционными методами. Такие решения, как правило, принимаются на уровне отдельных производственных или бизнес-процессов, при проектировании сложных систем и оптимизации распределения задач и ресурсов между ними.

Представленный в работе обзор основан на анализе литературы о существующих системах и алгоритмах планирования. Он может быть расширен и дополнен, потому что алгоритмы и методы планирования не являются универсальным средством и могут отличаться в разных сферах бизнеса, экономики и финансов. Каждая компания, внедряющая планирование задач, должна исходить из собственной специфики производственных и иных процессов внутри компании.

Обзор может быть полезен в компаниях для принятия решений по внедрению технологий оптимального планирования выполнения задач.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы на английском языке / References in English

1. Garey M.R. Computers and Intractability: A Guide to the Theory of NP-Completeness / M.R. Garey, D.S. Johnson. — New York : W.H. Freeman and Company, 1979. — 338 p. — DOI: 10.1137/1024022.

2. Coffman E.G. Optimal scheduling for two-processor systems / E.G. Coffman, R.L. Graham // Acta Informatica. — 1972. — Vol. 1. — P. 200–213. — DOI: 10.1007/BF00288685.

3. Różycki R. Energy-Aware Machine Learning Models—A Review of Recent Techniques and Perspectives / R. Różycki, D.A. Solarska, G. Waligóra // Energies. — 2025. — Vol. 18. — № 11. — Art. 2810. — DOI: 10.3390/en18112810.

4. Sinnen O. Task Scheduling for Parallel Systems / O. Sinnen. — Hoboken : John Wiley & Sons, 2007. — 320 p. — DOI: 10.1002/047012117.

5. Casavant T.L. A taxonomy of scheduling in general-purpose distributed computing systems / T.L. Casavant, J.G. Kuhl // IEEE Transactions on Software Engineering. — 1988. — Vol. 14. — № 2. — P. 141–154. — DOI: 10.1109/32.4634.

6. Kwok Y.K. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors / Y.K. Kwok, I. Ahmad // IEEE Transactions on Parallel and Distributed Systems. — 1996. — Vol. 7. — № 5. — P. 506–521. — DOI: 10.1109/71.503776.

7. Topcuoglu H. Task scheduling algorithms for heterogeneous processors / H. Topcuoglu, S. Hariri, M.-Y. Wu // Proceedings of the Eighth Heterogeneous Computing Workshop (HCW'99). — 1999. — P. 3–14. — DOI: 10.1109/HCW.1999.765092

8. Maurya A.K. Performance Comparison of HEFT, Lookahead, CEFT and PEFT Scheduling Algorithms for Heterogeneous Computing Systems / A.K. Maurya, A.K. Tripathi // Proceedings of the 7th International Conference on Computer and Communication Technology (ICCCT-2017). — 2017. — P. 128–132. — DOI: 10.1145/3154979.3154985

9. Braun T.D. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems / T.D. Braun, H.J. Siegel, N. Beck [et al.] // Journal of Parallel and Distributed Computing. — 2001. — Vol. 61. — № 6. — P. 810–837. — DOI: 10.1006/jpdc.2000.1714.

10. Li G. Ant Colony Optimization Task Scheduling Algorithm for SWIM Based on Load Balancing / G. Li, Z. Wu // Future Internet. — 2019. — Vol. 11. — № 4. — Art. 90. — DOI: 10.3390/fi11040090.

11. Pongchairerks P. Particle swarm optimization algorithm applied to scheduling problems / P. Pongchairerks // ScienceAsia. — 2009. — Vol. 35. — № 1. — P. 89–94. — DOI: 10.2306/scienceasia1513-1874.2009.35.089.

12. Xhafa F. Metaheuristics for Scheduling in Distributed Computing Environments / F. Xhafa, A. Abraham. — Berlin: Springer, 2010. — 360 p. — DOI: 10.1007/978-3-540-69277-5.

13. Blumofe R.D. Scheduling multithreaded computations by work stealing / R.D. Blumofe, C.E. Leiserson // Journal of the ACM. — 1999. — Vol. 46. — № 5. — P. 720–748. — DOI: 10.1145/324133.324234.

14. Acar U.A. Scheduling parallel programs by work stealing with private deques / U.A. Acar, A. Charguerauld, M. Rainey // ACM SIGPLAN Notices. — 2013. — Vol. 48. — № 8. — P. 219–228. — DOI: 10.1145/2442516.2442538.

15. Arora N.S. Thread Scheduling for Multiprogrammed Multiprocessors / N.S. Arora, R.D. Blumofe, C.G. Plaxton // Theory of Computing Systems. — 2001. — Vol. 34. — № 2. — P. 115–144. — DOI: 10.1007/s00224-001-0004-z.

16. Feitelson D. Job scheduling in multiprogrammed parallel systems / D. Feitelson. — 1997. — URL: https://www.cs.huji.ac.il/w~feit/papers/SchedSurvey97TRshort.pdf (accessed: 09.09.2025).

17. Verma A. Large-scale cluster management at Google with Borg / A. Verma, L. Pedrosa, M. Korupolu [et al.] // Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15). — 2015. — Art. 18. — P. 1–17. — DOI: 10.1145/2741948.2741964.

18. Schwarzkopf M. Omega: flexible, scalable schedulers for large compute clusters / M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, J. Wilkes // Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13). — 2013. — P. 351–364. — DOI: 10.1145/2465351.2465386.

19. Ousterhout K. Sparrow: distributed, low latency scheduling / K. Ousterhout, P. Wendell, M. Zaharia, I. Stoica // Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). — 2013. — P. 69–84. — DOI: 10.1145/2517349.2522716.

20. Hindman B. Mesos: a platform for fine-grained resource sharing in the data center / B. Hindman, A. Konwinski, M. Zaharia [et al.] // Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11). — 2011. — P. 295–308. — URL: https://amplab.cs.berkeley.edu/wp-content/uploads/2011/06/Mesos-A-Platform-for-Fine-Grained-Resource-Sharing-in-the-Data-Center.pdf (accessed: 09.09.2025).

21. Kubernetes Documentation // Kubernetes. — URL: https://kubernetes.io/docs/home/ (accessed: 09.09.2025).

22. Mao H. Learning scheduling algorithms for data processing clusters / H. Mao, M. Schwarzkopf, S.B. Venkatakrishnan [et al.] // Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19). — 2019. — P. 270–288. — DOI: 10.1145/3341302.3342080.

23. Shyalika C. Reinforcement Learning in Dynamic Task Scheduling: A Review / C. Shyalika, T. Silva, A. Karunananda // SN Computer Science. — 2020. — Vol. 1. — № 6. — Art. 306. — DOI: 10.1007/s42979-020-00326-5.

24. Różycki R. Energy-Aware Machine Learning Models — A Review of Recent Techniques and Perspectives / R. Różycki, D.A. Solarska, G. Waligóra // Energies. — 2025. — Vol. 18. — № 11. — Art. 2810. — DOI: 10.3390/en18112810.

25. Reiss C. Heterogeneity and dynamicity of clouds at scale: Google trace analysis / C. Reiss, A. Tumanov, G.R. Ganger [et al.] // Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12). — 2012. — Art. 7. — P. 1–13. — DOI: 10.1145/2391229.2391236.

26. Cortez E. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms / E. Cortez, A. Bonde, A. Muzio [et al.] // Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17). — 2017. — P. 153–167. — URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/10/Resource-Central-SOSP17.pdf (accessed: 09.09.2025).