СТАТЬЯ

УДК 004.4

РАЗРАБОТКА АЛГОРИТМА АДАПТИВНОГО ПЛАНИРОВАНИЯ ЗАДАЧ ДЛЯ СИСТЕМ С ОГРАНИЧЕННЫМИ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ

Золотухина Д.Ю.

ФГБОУ ВО «Воронежский государственный университет», Воронеж, e-mail: dar.zolott@gmail.com

Цель работы заключается в разработке и оценке алгоритма адаптивного планирования задач для систем с ограниченными вычислительными ресурсами. Алгоритм реализован на языке Java и учитывает приоритет, дедлайн и доступные ресурсы для распределения задач. В ходе работы были смоделированы сценарии с различной степенью нагрузки, а для оценки производительности алгоритма фиксировались метрики завершения задач в срок, использования ресурсов и обработки проблемных задач. Результаты показали, что алгоритм способен эффективно адаптироваться к изменениям нагрузки и ограничений, поддерживая высокий уровень завершения задач в срок и минимизируя перегрузку ресурсов. Сравнение с популярными технологиями, такими как Round-Robin, Earliest Deadline First и Priority Scheduling, продемонстрировало уникальные преимущества предложенного подхода, заключающиеся в учёте адаптации к ресурсам и комплексной обработке задач. Выводы исследования подтверждают, что разработанный алгоритм обеспечивает баланс между производительностью и эффективным использованием ресурсов в системах с ограничениями. Он может быть использован в различных областях, включая облачные вычисления, робототехнику и системы реального времени. Дальнейшая оптимизация алгоритма предполагает разработку механизмов для обработки проблемных задач и интеграцию с распределёнными системами.

Ключевые слова: адаптивное планирование задач, ограниченные вычислительные ресурсы, распределение задач, высоконагруженные системы, управление ресурсами

DEVELOPMENT OF AN ADAPTIVE TASK SCHEDULING ALGORITHM FOR SYSTEMS WITH LIMITED COMPUTATIONAL RESOURCES

Zolotukhina D.Yu.

Voronezh State University, Voronezh, e-mail: dar.zolott@gmail.com

The purpose of this study is the development and evaluation of an adaptive task scheduling algorithm for systems with limited computational resources. The algorithm, implemented in Java, considers priority, deadlines, and available resources for task allocation. Scenarios with varying levels of load were simulated, and the algorithm's performance was assessed using metrics such as task completion within deadlines, resource utilization, and handling problematic tasks. The results demonstrated that the algorithm effectively adapts to changes in load and resource constraints, maintaining a high rate of task completion within deadlines and minimizing resource overload. A comparison with popular scheduling technologies such as Round-Robin, Earliest Deadline First, and Priority Scheduling revealed the unique advantages of the proposed approach, including resource adaptation and comprehensive task handling. The findings confirm that the developed algorithm achieves a balance between performance and efficient resource utilization in constrained systems. It can be applied in various domains, including cloud computing, robotics, and real-time systems. Further optimization of the algorithm is envisioned to include mechanisms for handling problematic tasks and integration with distributed systems.

Keywords: adaptive task scheduling, limited computational resources, task allocation, high-load systems, resource management

Введение

Управление задачами в системах с ограниченными вычислительными ресурсами представляет собой сложную и актуальную проблему для современных высоконагруженных приложений. Такие системы, как распределённые вычислительные платформы, системы реального времени и облачные инфраструктуры, требуют эффективного распределения ресурсов для обеспечения высокой производительности и минимизации задержек. Одним из ключевых аспектов является адаптивное планирование задач, позволяющее учитывать ограничения по памяти, процессорным ресурсам, дедлайнам и приоритетам.

Примеры применения таких подходов включают системы реального времени, такие как обработка видео- или аудиопотоков, где критически важно выполнять задачи с минимальными задержками. Облачные вычисления являются ещё одной областью, где распределение ресурсов между виртуальными машинами или контейнерами требует сложного планирования для оптимизации использования ограниченной памяти и процессорного времени [1]. В интернет-сервисах, обрабатывающих большое количество запросов пользователей, например в системах рекомендаций или аналитики, эффективное управление задачами позволяет избежать перегрузок и улучшить отклик системы. Так-

же адаптивное планирование находит применение в робототехнике, где необходимо учитывать энергопотребление, пропускную способность вычислительных модулей и приоритет выполнения критических операций.

Целью данного исследования является разработка и оценка эффективности алгоритма адаптивного планирования задач, способного учитывать дедлайны, приоритеты задач и текущее состояние ресурсов. Предложенный алгоритм предназначен для использования в высоконагруженных системах, где важна гибкость и способность к динамической адаптации в условиях изменения требований и доступных ресурсов.

Материалы и методы исследования

Для проведения исследования был разработан алгоритм адаптивного планирования задач, предназначенный для систем с ограниченными вычислительными ресурсами. Алгоритм реализован на языке Java версии OpenJDK 17, что обеспечило использование современных возможностей для работы с многопоточностью и управления ресурсами. Имитация ограниченной среды была выполнена путем искусственного ограничения доступных вычислительных ресурсов, включая процессорное время и оперативную память.

Тестовая программа была модульным приложением, состоящим из компонентов для генерации задач, управления их выполнением и сбора данных о производительности алгоритма. Задачи представляли собой программные сущности, моделирующие типовые нагрузки. Каждая задача характеризовалась следующими параметрами: приоритет (высокий, средний, низкий), время выполнения, объем памяти и дедлайн. Задачи генерировались случайным образом, что обеспечивало разнообразие условий тестирования. Например, критические задачи имели жесткие дедлайны и малое время выполнения, в то время как фоновые задачи допускали длительные задержки и низкий приоритет выполнения.

Пример реализации класса задачи:

```
public class Task {
    private int priority; // Приоритет задачи: высокий, средний, низкий
    private long executionTime; // Время выполнения задачи в миллисекундах
    private int memoryUsage; // Потребление памяти в килобайтах
    private long deadline; // Дедлайн выполнения задачи

    // Конструктор
    // Геттеры и сеттеры
}
```

Генерация задач выполнялась с использованием библиотеки Random и представляла собой последовательный процесс формирования очередей задач различных типов. Все задачи сохранялись в приоритетных очередях, реализованных с помощью структуры данных PriorityQueue. Это позволило алгоритму динамически адаптировать стратегию выполнения в зависимости от текущей нагрузки.

Алгоритм планирования задач был разработан для динамического распределения ограниченных ресурсов с учетом следующих правил:

1. Инициализация: все задачи при создании добавляются в приоритетную очередь, отсортированную по дедлайнам и приоритетам. В начале работы алгоритм анализирует доступные ресурсы (оперативная память и процессорное время) и определяет текущие ограничения.

2. Выбор задачи для выполнения: задача извлекается из очереди в порядке приоритета – сначала задачи с высоким приоритетом, затем средним и наконец низким. Если доступных ресурсов недостаточно для выполнения задачи (например, не хватает памяти или процессорных ресурсов), алгоритм переходит к следующей задаче. Если задача превышает дедлайн, она перемещается в отдельный список «проблемных задач» для дальнейшего анализа.

3. Выполнение задачи: после успешного выбора задачи происходит выделение ресурсов для её выполнения. В процессе выполнения контролируются текущие ресурсы. Если ресурсы исчерпаны, выполнение низкоприоритетных задач приостанавливается.

4. Адаптация параметров: каждые 10 секунд анализируются текущие метрики (уровень загрузки CPU, объем используемой памяти, количество задач в очереди). Если выявляется критический недостаток ресурсов, алгоритм увеличивает доступность памяти за счет ограничения количества активных задач низкого приоритета. Если ресурсы используются менее чем на 70%, алгоритм увеличивает число одновременно выполняемых задач, включая задачи низкого приоритета.

5. Обработка неудачных сценариев: если задача не может быть выполнена даже при

адаптации параметров, она перемещается в список «проблемных задач». Эти задачи могут быть выполнены вручную или перераспределены в другой момент времени.

6. Завершение работы: алгоритм завершает выполнение всех оставшихся задач, когда очередь становится пустой или при достижении заданного времени выполнения эксперимента.

Для имитации ограниченной вычислительной среды использовались настройки JVM, включавшие ограничение максимального объема оперативной памяти (-Xmx2G) и минимального (-Xms2G), а также настройка числа потоков, доступных для выполнения задач. Эти параметры обеспечивали воспроизводимость условий тестирования и моделировали реальные ограничения, характерные для высоконагруженных систем, включая ограниченное количество вычислительных ресурсов, высокую интенсивность входящих запросов, необходимость соблюдения жёстких временных ограничений и обеспечение устойчивости при пиковой нагрузке [2].

Программа была протестирована в трех сценариях: низкая нагрузка (50 задач, доступная память 1 GB), средняя нагрузка (500 задач, доступная память 500 MB) и высокая нагрузка (5000 задач, доступная память 200 MB). Каждый сценарий предусматривал различные комбинации типов задач с учетом их дедлайнов, приоритетов и времени выполнения. Для минимизации влияния случайных факторов каждый эксперимент повторялся десять раз, а результаты усреднялись.

Блок-схема работы алгоритма изображена на рисунке 1.

Метрики измерения включали:

• процент завершенных задач в дедлайн;

• среднее время выполнения задачи;

• пиковое и среднее использование оперативной памяти;

• уровень использования процессора;

• количество задач, перемещенных в список проблемных.

Мониторинг производительности осуществлялся с использованием встроенных средств Java, таких как System.nanoTime() для измерения времени выполнения задач и Runtime.getRuntime() для анализа потребления памяти. Также использовались сторонние инструменты, включая VisualVM [3] и Java Flight Recorder [4], для детального анализа характеристик работы программы.

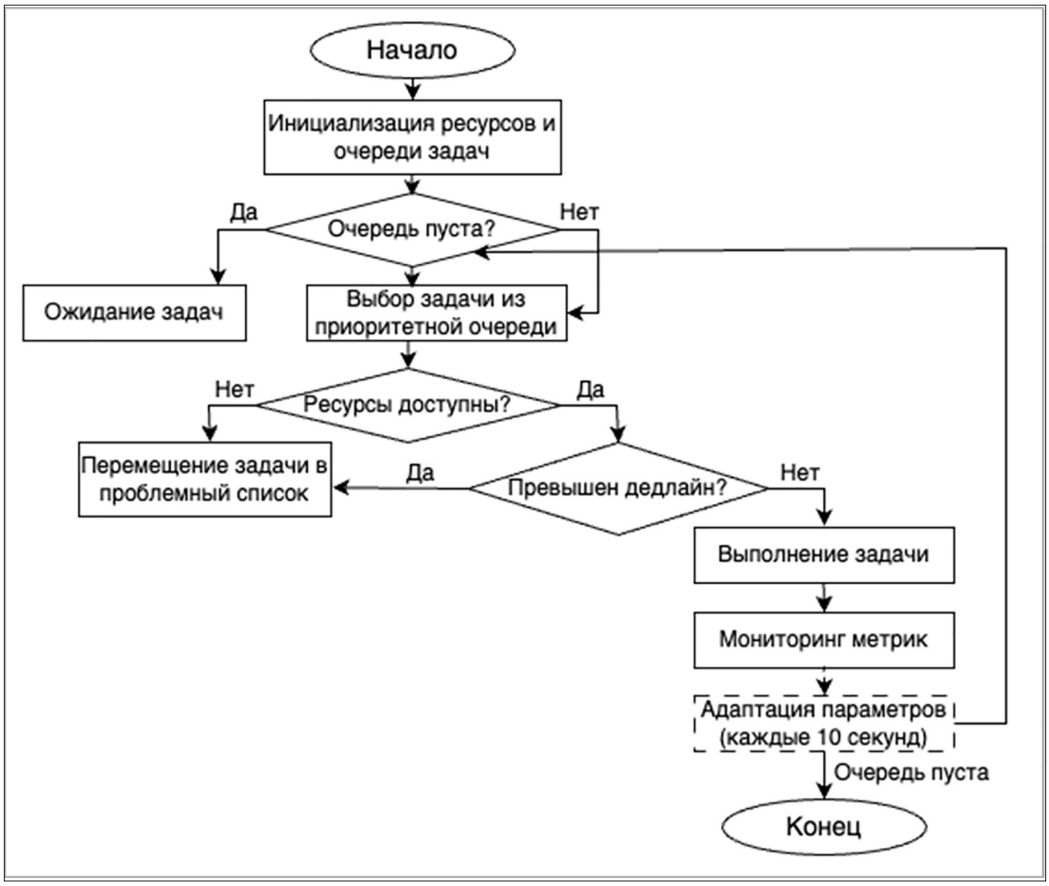

Рис. 1. Схема работы алгоритма

Таким образом, разработанный алгоритм включал гибкие адаптационные механизмы и учитывал специфику задач с ограниченными ресурсами, что позволило обеспечить их выполнение с минимальными затратами и максимальной эффективностью.

Результаты исследования и их обсуждение

Результаты исследования представлены в таблице, которая включает основные метрики эффективности работы разработанного алгоритма в условиях низкой, средней и высокой нагрузки. Анализ данных таблицы позволяет выявить ключевые закономерности и особенности работы алгоритма в различных условиях.

В условиях низкой нагрузки алгоритм продемонстрировал максимальную эффективность, когда 100% задач были завершены в рамках заданного дедлайна, а среднее время выполнения одной задачи составило 12 мс. Пиковое использование памяти в этих условиях составило 850 MB, при среднем значении 640 MB. Уровень загрузки процессора оказался минимальным – 45%, что подтверждает оптимальное распределение ресурсов.

Во время средней нагрузки эффективность алгоритма несколько снизилась: 98% задач были выполнены в рамках дедлайна, а среднее время выполнения одной задачи возросло до 35 мс. Пиковое использование памяти уменьшилось до 460 MB, что связано с увеличением числа задач и перераспределением ресурсов. Уровень загрузки CPU достиг 70%, а число задач, перемещённых в список проблемных, составило 9. Это указывает на устойчивую работу алгоритма в условиях увеличенной нагрузки.

Результаты работы алгоритма в условиях различных нагрузок

Тип нагрузки	% завершённых задач в дедлайн	Среднее время выполнения задачи (мс)	Пиковое использование памяти (MB)	Среднее использование памяти (MB)	Уровень загрузки CPU (%)	Число проблемных задач
Низкая	100	12	850	640	45	0
Средняя	98	35	460	320	70	9
Высокая	95	95	180	140	85	210

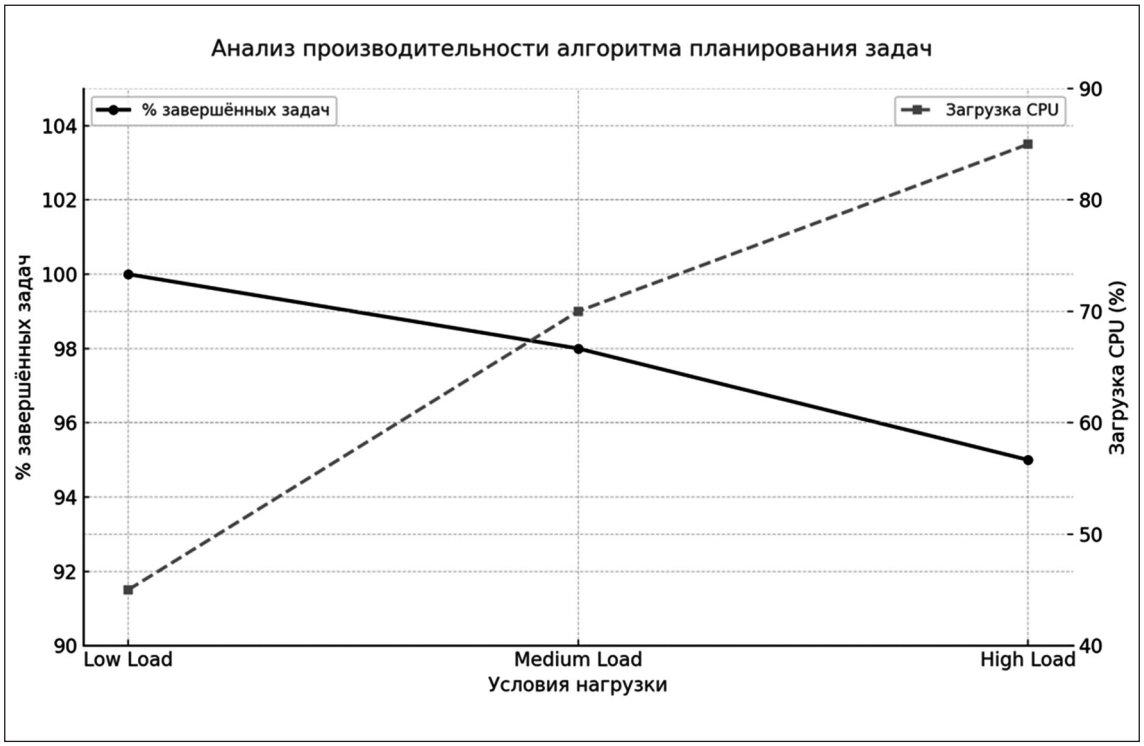

Рис. 2. График производительности алгоритма

При высокой нагрузке алгоритм сохранил приемлемую производительность, хотя доля задач, выполненных в рамках дедлайна, снизилась до 95%. Среднее время выполнения одной задачи увеличилось до 95 мс, что обусловлено перераспределением ресурсов в пользу задач с более высоким приоритетом. Пиковое использование памяти снизилось до 180 MB, а среднее – до 140 MB, что указывает на активное использование адаптивных механизмов алгоритма. Однако количество проблемных задач значительно возросло и составило 210, что связано с тем, что ресурсы системы становятся предельно ограниченными и алгоритм вынужден перемещать задачи с высоким потреблением ресурсов в список проблемных. Анализ уровня загрузки CPU показал его закономерное увеличение от 45% при низкой нагрузке до 85% при высокой, что свидетельствует о том, что алгоритм эффективно использует доступные вычислительные ресурсы, минимизируя их простаивание.

На рисунке 2 представлена зависимость процента задач, завершённых в дедлайн, от уровня загрузки CPU при различных сценариях нагрузки. График демонстрирует способность алгоритма к адаптации. На этапе низкой нагрузки алгоритм демонстрирует баланс между высокой эффективностью выполнения задач и минимальным использованием ресурсов. Средняя нагрузка иллюстрирует переходный этап, когда алгоритм перераспределяет ресурсы для поддержания высокого процента завершённых задач (98%) при возросшей загрузке CPU (70%). Высокая нагрузка подчёркивает ограничения системы: алгоритм сохраняет удовлетворительные показатели выполнения задач (95%), но достигает пикового использования ресурсов (85% загрузки CPU), что сопровождается ростом количества задач в списке проблемных.

Сравнение с популярными технологиями управления задачами, такими как Round-Robin, Earliest Deadline First (EDF) и Priority Scheduling, подчёркивает уникальность предложенного алгоритма. Round-Robin обеспечивает равномерное распределение процессорного времени, но не учитывает дедлайны и приоритеты, что делает его непригодным для систем с ограниченными ресурсами [5]. Earliest Deadline First (EDF) фокусируется на дедлайнах, однако игнорирует приоритеты и текущее состояние ресурсов, что ограничивает его применимость в сложных сценариях [6]. Priority Scheduling, в свою очередь, ориентируется только на приоритеты задач, не обращая внимания на дедлайны и адаптацию к ресурсам [7]. В отличие от них предложенный алгоритм комплексно учитывает дедлайны, приоритеты и текущее состояние ресурсов, что делает его гибким инструментом для управления задачами в высоконагруженных системах с ограничениями.

Выводы

Проведённое исследование продемонстрировало, что разработанный алгоритм адаптивного планирования задач обладает уникальными свойствами, которые делают его эффективным инструментом для управления задачами в условиях ограниченных вычислительных ресурсов. Алгоритм показал высокую гибкость и адаптивность, позволяя учитывать текущие ограничения памяти и процессорного времени, а также приоритеты и дедлайны задач. Такой подход обеспечивает баланс между производительностью и эффективным использованием ресурсов, что особенно важно для высоконагруженных систем.

Анализ работы алгоритма показал его способность поддерживать высокий уровень завершения задач в рамках дедлайна даже при увеличении нагрузки. Это стало возможным благодаря интеграции адаптивных механизмов, которые позволяют динамически изменять параметры выполнения задач в зависимости от текущей ситуации. В условиях низкой нагрузки алгоритм обеспечивает оптимальное распределение ресурсов, минимизируя задержки и снижая нагрузку на процессор. При увеличении нагрузки алгоритм демонстрирует устойчивость, перераспределяя ресурсы и поддерживая выполнение приоритетных задач в рамках дедлайнов.

Основное преимущество алгоритма заключается в его комплексном подходе, который сочетает учёт дедлайнов, приоритетов задач и адаптацию к текущему состоянию системы. Это делает его значительно более гибким и универсальным по сравнению с существующими технологиями, такими как Round-Robin, Earliest Deadline First и Priority Scheduling, которые либо не учитывают дедлайны, либо не адаптируются к текущим ограничениям ресурсов. Разработанный алгоритм также справляется с обработкой задач, которые в других подходах могли бы оставаться невыполненными.

Вместе с тем исследование выявило и ограничения алгоритма, связанные с увеличением числа проблемных задач при высокой нагрузке. Это подчёркивает необходимость дальнейшего улучшения алгоритма, включая разработку механизмов для более эффективной обработки задач, перемещённых в список проблемных. Будущие исследования могут быть направлены на интеграцию алгоритма с облачными решениями

для масштабирования ресурсов, что позволит снизить влияние ограничений вычислительных мощностей.

Таким образом, разработанный алгоритм демонстрирует значительный потенциал для применения в высоконагруженных системах, таких как облачные вычисления, системы реального времени и распределённые платформы. Его использование может существенно повысить эффективность управления задачами и улучшить общую производительность систем с ограниченными ресурсами.

Список литературы

1. Рубашенков А.М., Бобров А.В. Облачные вычисления // Academy. 2018. № 6(33). С. 33-36.

2. Филисов Д.А. Архитектура высоконагруженных приложений // Universum: технические науки. 2023. №10-1 (115). С. 49-53. DOI: 10.32743/UniTech.2023.115.10.16138.

3. Java VisualVM documentation. [Электронный ресурс]. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm (дата обращения: 20.11.2024).

4. About Java Flight Recorder. [Электронный ресурс]. URL: https://docs.oracle.com/javacomponents/jmc-5-4/jfr-runtime-guide/about.htm (дата обращения: 20.11.2024).

5. Шуляк А.В. Сравнительный анализ алгоритмов балансировки нагрузки в среде облачных вычислений // Научный журнал. 2021. № 6 (61). С. 6-11.

6. Verma C., Stoffová V., Illés Z. Rate-monotonic vs early deadline first scheduling: A review // Proceeding of Education Technology-Computer science in building better future. 2018. P. 188-193.

7. Alistarh D., Kopinsky J., Li J.Z., Nadiradze G. The power of choice in priority scheduling // Proceedings of the ACM Symposium on Principles of Distributed Computing. 2017. P. 283-292. DOI: 10.1145/3087801.3087810.