СТАТЬЯ

УДК 004.4

РАЗРАБОТКА АЛГОРИТМА АДАПТИВНОГО РАСПРЕДЕЛЕНИЯ ЗАПРОСОВ В МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ С УЧЕТОМ ДИНАМИЧЕСКИХ ХАРАКТЕРИСТИК УЗЛОВ

Золотухина Д.Ю.

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

В условиях стремительного роста числа запросов и динамической нагрузки в микросервисных системах особенно важно эффективно распределять поступающие задачи между узлами. Цель представленного исследования – разработать и оценить алгоритм, способный учитывать загруженность центральных процессоров, задержки при передаче данных, количество активных соединений и типы обрабатываемых запросов. Предложенная методика реализована с помощью многопоточности на языке Java с применением контейнеризации в Docker и оркестрации в Kubernetes, что обеспечивает гибкость масштабирования и точное моделирование реальных условий. В ходе экспериментов использовался кластер из десяти виртуальных узлов с различными параметрами аппаратного обеспечения. Рассматривались три категории запросов: требующие больших вычислительных ресурсов, ориентированные на операции ввода-вывода и комбинированные варианты. Для оценки работы алгоритма анализировались метрики, включающие время обработки, пропускную способность, равномерность распределения нагрузки и долю невыполненных запросов. Сбор данных осуществлялся с помощью специализированных средств мониторинга, а их последующий анализ позволил выявить эффективность предлагаемого подхода. Результаты показали, что алгоритм достигает высокой точности распределения при низкой и средней нагрузках, сохраняя удовлетворительный уровень выполнения запросов и при больших объемах данных. Сопоставление с традиционными методами, не учитывающими динамические характеристики узлов, подтвердило преимущество предложенного решения. Полученные выводы указывают на перспективность внедрения алгоритма в высоконагруженные микросервисные системы для повышения стабильности работы и оптимизации использования ресурсов.

Ключевые слова: распределение запросов, микросервисная архитектура, высоконагруженные системы, управление ресурсами, адаптивные алгоритмы, динамическое планирование

DEVELOPMENT OF AN ALGORITHM FOR ADAPTIVE DISTRIBUTION OF REQUESTS IN MICROSERVICE ARCHITECTURE TAKING INTO ACCOUNT DYNAMIC CHARACTERISTICS OF NODES

Zolotukhina D.Yu.

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

With the rapid growth in the number of requests and dynamic load in microservice systems, it is especially important to efficiently distribute incoming tasks among nodes. The purpose of the presented research is to develop and evaluate an algorithm that is able to take into account CPU utilization, data transfer delays, number of active connections and types of requests being processed. The proposed technique is implemented using multithreading in Java with containerization in Docker and orchestration in Kubernetes, which provides scalability flexibility and accurate simulation of real-world conditions. The experiments utilized a cluster of ten virtual nodes with different hardware parameters. Three categories of queries were considered: computationally demanding, I/O-oriented, and combined variants. To evaluate the performance of the algorithm, metrics including processing time, throughput, load distribution uniformity, and the proportion of outstanding requests were analyzed. Data was collected using specialized monitoring tools, and their subsequent analysis revealed the effectiveness of the proposed approach. The results showed that the algorithm achieves high distribution accuracy at low and medium loads, while maintaining a satisfactory level of query fulfillment at large data volumes. Comparison with traditional methods, which do not take into account the dynamic characteristics of nodes, confirmed the advantage of the proposed solution. The conclusions obtained point to the promising implementation of the algorithm in highly loaded microservice systems to improve the stability of operation and optimize the use of resources.

Keywords: request distribution, microservice architecture, high-load systems, resource management, adaptive algorithms, dynamic scheduling

Введение

В современных высоконагруженных микросервисных системах распределение запросов между узлами определяет стабильность и эффективное использование ресурсов. Подобные системы обрабатывают миллионы запросов ежедневно, что требует эффективных алгоритмов балансировки. Сложность возрастает при динамически изменяющихся параметрах (загрузка про-

цессора, сетевые задержки, число активных соединений, тип запросов), а ошибки распределения приводят к снижению производительности, увеличению времени отклика и возможным отказам.

Существующие подходы к распределению нагрузки, такие как Round-Robin, Least Connections и Random, имеют определенные ограничения. Например, алгоритм Round-Robin равномерно распределяет запросы, не учитывая текущее состояние узлов [1], что неэффективно при изменяющихся нагрузках. Least Connections минимизирует число соединений, игнорируя тип запроса и сетевые характеристики [2]. Random полагается на случайное распределение, утрачивая эффективность при высокой загрузке [3].

Особую сложность представляет задача интеграции параметров запросов, таких как их тип и ресурсные требования. Например, CPU-интенсивные запросы, требующие значительных вычислительных мощностей, могут перегружать узлы с ограниченными ресурсами. I/O-интенсивные запросы, напротив, нагружают операции ввода-вывода, такие как обращение к диску или базам данных. Смешанные запросы комбинируют обе нагрузки, что требует более сложной оценки доступности ресурсов. Без учета этих параметров нагрузка может распределяться неравномерно, что приводит к снижению пропускной способности системы и увеличению времени обработки запросов.

С ростом числа узлов в кластере увеличивается риск «узких мест», когда одни узлы перегружены, а другие простаивают [4]. Это может быть связано с различиями в сетевых характеристиках, состоянием оперативной памяти или задержками при обработке запросов. Разработка алгоритма, способного учитывать не только статические характеристики узлов, но и их динамические изменения в реальном времени, является актуальной задачей.

Балансировка нагрузки на основе адаптивных алгоритмов, интегрирующих множество метрик в процессе принятия решений, представляет собой перспективное направление для высоконагруженных микросервисных систем. Такие алгоритмы должны обеспечивать равномерное распределение нагрузки, минимизацию времени обработки запросов, эффективное использование ресурсов и низкий процент отказов. При этом важно, чтобы технология была устойчива к изменениям параметров нагрузки и конфигурации системы, обеспечивая стабильность работы даже при экстремальных условиях.

Целью данного исследования является разработка и оценка алгоритма оптимального распределения запросов в микросервисной архитектуре с учётом динамических параметров узлов системы, таких как загрузка процессора, сетевые задержки, количество активных соединений и тип обрабатываемого запроса для повышения эффективности использования ресурсов и равномерности распределения нагрузки, а также на минимизацию времени обработки запросов и процента отказов в условиях различных уровней нагрузки.

Материалы и методы исследования

Для проведения исследования разработан алгоритм оптимального распределения запросов в микросервисной архитектуре, учитывающий динамическое состояние узлов системы. Алгоритм реализован на языке Java версии OpenJDK 17 с использованием современных средств многопоточности и управления ресурсами. Имитация высоконагруженной среды осуществлялась на платформе Kubernetes, что обеспечило гибкость масштабирования узлов и точное моделирование нагрузок [5]. Для изоляции и стандартизации окружения использовалась контейнеризация на основе Docker [6].

Эксперименты проводились в кластере, состоящем из 10 виртуальных узлов, равномерно распределенных по характеристикам: процессор Intel Core i7-12700K (12 ядер, 20 потоков, 3,6 GHz), оперативная память 16 GB DDR4, NVMe SSD емкостью 500 GB, гигабитное сетевое соединение с искусственными задержками для эмуляции реальных условий. Система генерировала потоки запросов трех типов:

1. CPU-интенсивные запросы – задачи, требующие значительных вычислительных ресурсов, такие как сложные математические вычисления и обработка больших массивов данных.

2. I/O-интенсивные запросы – задачи, где основная нагрузка связана с операциями ввода-вывода. Они были представлены чтением и записью данных в базу и обращением к файловой системе.

3. Смешанные запросы – запросы, комбинирующие интенсивные вычисления и операции ввода-вывода, такие как обработка данных с последующей записью в базу данных.

Характеристика нагрузки варьировалась от малых запросов (время обработки менее 1 с) до больших (свыше 5 с). Для воспроизводимости и управления параметрами нагрузки использовалась библиотека JMH (Java Microbenchmark Harness) [7].

Архитектура программы представлена на рис. 1.

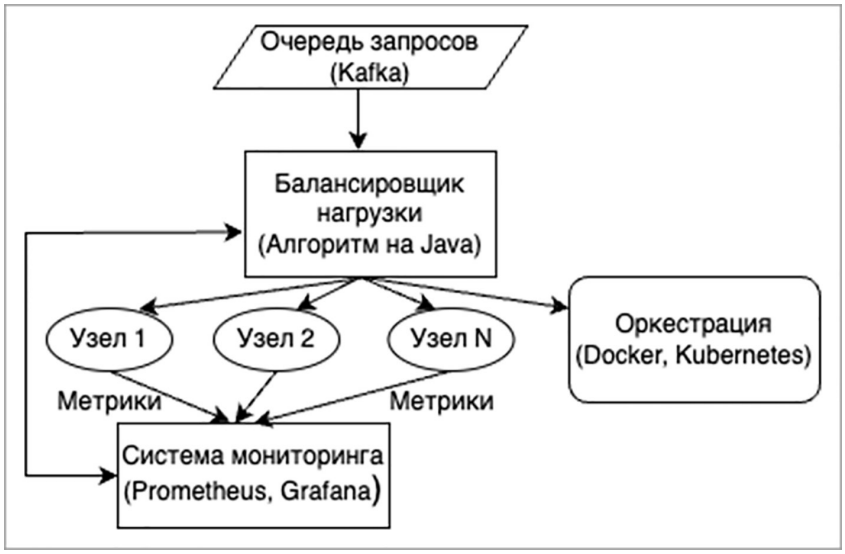

Рис. 1. Схема архитектуры программы
Источник: составлено авторами

Основой работы алгоритма является взвешенная функция, определяющая оптимальный узел для обработки запроса. Функция (1) рассчитывается по следующей формуле:

$$W_i = \alpha \cdot L_i + \beta \cdot T_r + \gamma \cdot D_i + \delta \cdot N_i,$$

где W_i – общий вес узла i, L_i – загрузка процессора узла, T_r – тип запроса, D_i – сетевые задержки узла, N_i – число активных соединений. Коэффициенты α, β, γ, δ определяют весовые приоритеты каждого параметра, они были выбраны таким образом, чтобы обеспечить баланс между производительностью и стабильностью системы. Значения коэффициентов эмпирически определены на этапе предварительного тестирования: $\alpha = 0{,}4$, $\beta = 0{,}2$, $\gamma = 0{,}3$, $\delta = 0{,}1$. Такой выбор обусловлен необходимостью учитывать загрузку процессора и сетевые задержки в первую очередь, при этом уделяя меньшее внимание числу соединений и типу запроса.

Алгоритм распределения запросов включает следующие этапы:

1. Инициализация: на первом этапе алгоритм собирает данные о текущем состоянии всех узлов системы. Если для узла доступна информация о загрузке процессора L_i, объеме доступной памяти, сетевых задержках D_i и количестве активных соединений N_i, она фиксируется в таблице состояния. На основе собранных данных вычисляются начальные метрики производительности узлов. Если хотя бы один узел оказывается недоступным или перегруженным, он временно исключается из маршрутизации запросов до восстановления доступности.

2. Получение запросов: запросы поступают в очередь обработки. Если запрос фиксируется в очереди, для него определяются следующие характеристики: тип запроса (T_r), предполагаемое время выполнения и требования к оперативной памяти. Если один из параметров выходит за пределы допустимых значений, запрос помечается как проблемный. Очередь запросов сортируется по их приоритету, чтобы более критичные запросы обрабатывались в первую очередь.

3. Анализ узлов: алгоритм анализирует состояние всех доступных узлов для текущего запроса. Если узел находится в состоянии, допускающем обработку запроса, для него вычисляется значение W_i взвешенной функции (1). Узлы с наименьшим значением W_i добавляются в список кандидатов для обработки запроса. Если вес всех узлов превышает пороговое значение, запрос остается в очереди.

4. Распределение запросов: алгоритм выбирает узел с минимальным значением функции W_i для обработки текущего запроса. Если ресурсы выбранного узла превышают порог загрузки, запрос передается следующему узлу из списка кандидатов. В случае успешного распределения ресурсов запрос отправляется на обработку. Если ресурсы недостаточны, алгоритм либо возвращает запрос в очередь, либо помечает его как проблемный, в зависимости от текущего состояния системы.

5. Обновление метрик: каждые 5 с алгоритм пересчитывает метрики производительности для всех узлов. Если узел временно недоступен или перегружен, он ис-

ключается из списка доступных для обработки запросов. Если узел восстанавливает свою доступность, он автоматически включается обратно в список доступных для маршрутизации.

6. Обработка отказов: если запрос не может быть обработан из-за перегрузки узлов или их отказа, он перенаправляется на резервный узел. Если резервный узел также перегружен, запрос остается в очереди для повторной обработки. Узлы, не способные обрабатывать запросы из-за отказов, проверяются на предмет восстановления. Как только доступность узла подтверждена, он снова включается в процесс маршрутизации.

Метрики измерения эффективности включали:

− среднюю задержку обработки запросов (мс);

− пропускную способность (количество запросов в секунду);

− уровень использования ресурсов (процент загрузки CPU и памяти);

− процент отказов (доля запросов, не обработанных в срок);

− равномерность распределения нагрузки между узлами.

Эксперименты выполнялись при нагрузках 100, 1000 и 5000 запросов в секунду. Каждый сценарий продолжался 60 мин с интервалами записи данных в 5 мин. Сбор данных осуществлялся с использованием Prometheus [8] и Grafana [9], что обеспечивало точное измерение метрик в реальном времени. Данные анализировались с помощью Apache Spark [10], что позволило оценить как общую производительность алгоритма, так и его поведение в условиях изменения нагрузки.

Результаты исследования и их обсуждение

Результаты экспериментов представлены в таблице, где отражены основные метрики работы разработанного алгоритма в условиях низкой, средней и высокой нагрузки. Анализ этих данных позволяет выделить ключевые закономерности, оценить производительность алгоритма и сравнить его эффективность с другими подходами.

На этапе низкой нагрузки алгоритм продемонстрировал практически идеальные результаты. Более 99,8% запросов были успешно завершены в срок, что свидетельствует о высокой стабильности и точности распределения нагрузки. Средняя задержка обработки запросов составила всего 12 мс, что соответствует минимально возможным значениям для такой системы. Пропускная способность в этих условиях достигла 98 запросов в секунду, что, хотя и не является максимальным значением, полностью удовлетворяет требованиям низкой нагрузки. Средняя загрузка процессора на узлах составила всего 40%, а равномерность распределения нагрузки между узлами достигла 0,92 указывая на сбалансированное использование ресурсов. Показатель отказов составил 0,2%, что можно считать практически нулевым уровнем. Эти результаты подтверждают, что алгоритм способен эффективно справляться с распределением запросов в условиях минимального уровня загруженности.

С увеличением интенсивности запросов до среднего уровня наблюдалось небольшое снижение эффективности алгоритма. Процент завершенных задач уменьшился до 97,5%, что на 2,3% ниже по сравнению с низкой нагрузкой. Средняя задержка обработки запросов возросла до 35 мс, что в 2,9 раза превышает показатель при низкой нагрузке. Пропускная способность, напротив, резко увеличилась до 870 запросов в секунду, демонстрируя способность алгоритма к адаптации в условиях увеличенной нагрузки. Средняя загрузка процессора составила 68%, что на 70% больше по сравнению с низкой нагрузкой, но остается в пределах допустимых значений для системы. Доля отказов выросла до 2,5%, что связано с увеличением конкуренции за ресурсы. Равномерность распределения нагрузки немного снизилась, до 0,85, что указывает на тенденцию к перераспределению ресурсов в пользу узлов с меньшей загруженностью.

Результаты работы алгоритма в условиях различных нагрузок

Тип нагрузки	% завершенных задач	Средняя задержка (мс)	Пропускная способность (запросов/с)	Средняя загрузка CPU (%)	Процент отказов (%)	Равномерность распределения
Низкая нагрузка	99,8	12	98	40	0,2	0,92
Средняя нагрузка	97,5	35	870	68	2,5	0,85
Высокая нагрузка	94,2	95	4300	85	5,8	0,78

Источник: составлено авторами.

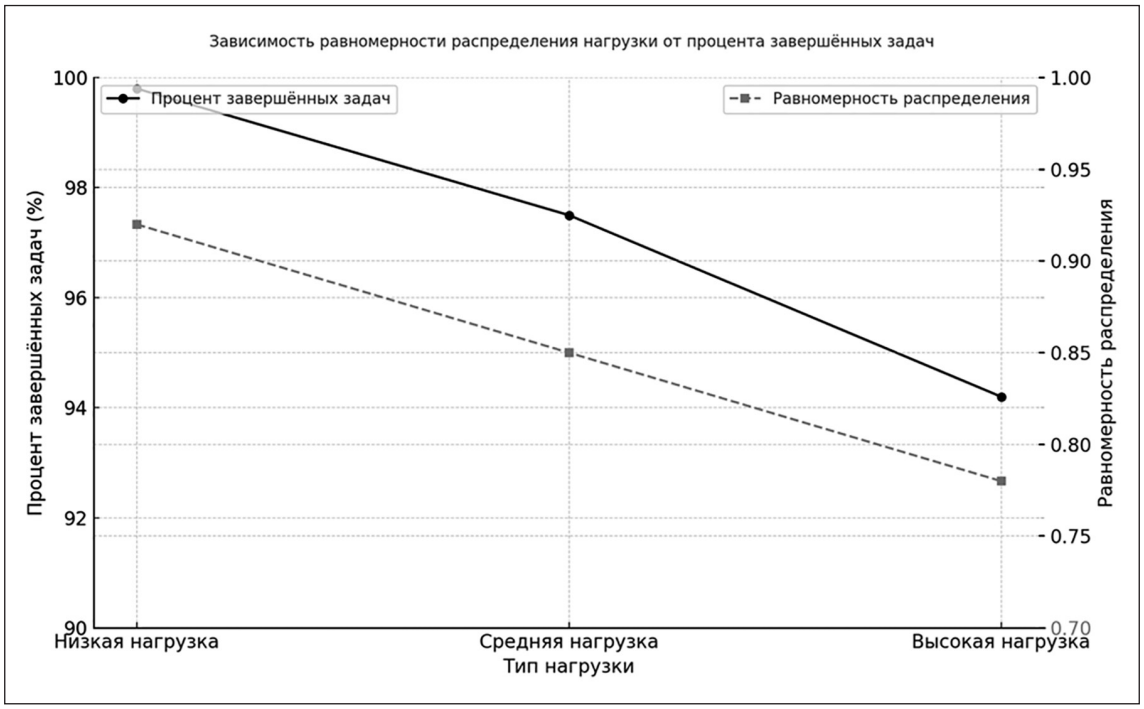

Рис. 2. Зависимость равномерности распределения нагрузки от процента завершенных задач
Источник: составлено авторами

В условиях высокой нагрузки алгоритм сохранил удовлетворительные показатели, несмотря на значительное увеличение числа запросов. Доля завершенных в срок запросов составила 94,2%, что на 5,6% ниже по сравнению с низкой нагрузкой, но остается высоким результатом для системы с экстремальными ограничениями. Средняя задержка выросла до 95 мс, что более чем в 2,7 раза превышает показатель средней нагрузки. Однако пропускная способность достигла рекордного значения – 4300 запросов в секунду, демонстрируя высокий уровень масштабируемости алгоритма. Средняя загрузка процессора увеличилась до 85%, а доля отказов возросла до 5,8%, что связано с исчерпанием доступных ресурсов при экстремальной нагрузке. Равномерность распределения снизилась до 0,78, что отражает сложность поддержания баланса ресурсов при таких условиях.

На рис. 2 представлен график зависимости равномерности распределения нагрузки от процента завершенных задач. График демонстрирует, что при низкой нагрузке алгоритм достигает высокого уровня равномерности (0,92), поддерживая практически полное выполнение задач. С увеличением нагрузки равномерность снижается на 7,6 и 15,2% при средней и высокой нагрузках соответственно, что коррелирует с ростом процента отказов.

График подтверждает адаптивные возможности алгоритма: несмотря на снижение равномерности распределения, система сохраняет высокий уровень выполнения задач в условиях изменения нагрузки.

Сравнение с существующими алгоритмами управления запросами подчеркивает уникальность предложенного подхода. Round-Robin равномерно распределяет запросы между узлами, но не учитывает текущие параметры узлов, такие как загрузка CPU и сетевые задержки. Least Connections минимизирует активные соединения, но не адаптируется к типу запросов и не анализирует их сетевые характеристики. Random полностью игнорирует состояние системы, что делает его малопригодным для высоконагруженных сред. Разработанный алгоритм демонстрирует преимущества, включая динамическую адаптацию и учет комплексных метрик узлов, что делает его эффективным и универсальным в условиях высоконагруженных систем.

Выводы

1. Проведенное исследование подтвердило эффективность разработанного алгоритма оптимального распределения запросов в микросервисной архитектуре для высоконагруженных систем. Алгоритм продемонстрировал способность адаптироваться к динамическим изменениям состоя-

ния узлов и нагрузки, обеспечивая высокую производительность даже в условиях ограниченных ресурсов.

2. Результаты экспериментов показали, что алгоритм поддерживает выполнение более 94 % запросов в срок даже при высокой нагрузке, минимизирует среднюю задержку и обеспечивает равномерное распределение нагрузки. Применение взвешенной функции, учитывающей загрузку процессора, сетевые задержки, тип запросов и число соединений, позволило эффективно перераспределять запросы, адаптируясь к изменениям в реальном времени. По сравнению с традиционными подходами, такими как Round-Robin и Least Connections, предложенный алгоритм демонстрирует преимущество за счет учета состояния узлов и характеристик запросов.

3. Несмотря на высокую производительность, выявлено снижение равномерности распределения нагрузки при экстремальных условиях. Это подчеркивает необходимость дальнейшей оптимизации алгоритма, включая улучшение механизмов обновления метрик и внедрение предиктивных методов для более точного управления запросами.

4. Результаты исследования подтверждают прикладную значимость алгоритма для высоконагруженных систем, таких как облачные платформы, веб-приложения и распределенные вычислительные системы. Алгоритм обеспечивает баланс между производительностью, равномерностью распределения и стабильностью работы, что делает его перспективным инструментом для современных микросервисных архитектур.

Список литературы

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

2. Власова В.А., Олейникова А.А. Сравнение методов балансировки нагрузки в беспроводных сенсорных сетях // Радиоэлектроника и информатика. 2018. № 3. С. 40–45. DOI: 10.30837/1563-0064.3.2018.162780.

3. Mitzenmacher M. On the Analysis of Randomized Load Balancing Schemes // Theory of Computing Systems. 1999. № 32. P. 361–386. DOI: 10.1007/s002240000122.

4. Цветков В.Я., Алпатов А.Н. Проблемы распределенных систем // Перспективы науки и образования. 2014. № 6 (12). С. 31–36.

5. Липатова С.Е., Федоров В.О., Белов Ю.С. Kubernetes как элемент человеко-компьютерного взаимодействия // E-Scio. 2023. № 1 (76). С. 49–55.

6. Чиганов Д.Р. Docker: ключ к контейнеризации и масштабируемости // Вестник науки. 2023. № 7 (64). С. 270–272.

7. Laaber C., Leitner P. An evaluation of open-source software microbenchmark suites continuous performance assessment // Proceedings of the 15th International Conference on Mining Software Repositories. 2018. P. 119–130. DOI: 10.1145/3196398.3196407.

8. Маратканов А.С., Суханов А.А., Воробьева А.А. Средства анализа и визуализации метрик работы приложения // International scientific review. 2019. № LIX. С. 41–43.

9. Средство визуализации данных Grafana. [Электронный ресурс]. URL: https://github.com/grafana/grafana (дата обращения: 10.01.2025).

10. Salloum S., Dautov R., Chen X. Big data analytics on Apache Spark // Int J Data Sci Anal. 2016. № 1. P. 145–164. DOI: 10.1007/s41060-016-0027-9.