СТАТЬИ

УДК 004.421
DOI 10.17513/snt.40317

ПЕРЕДОВОЙ ПОДХОД К РАСПРЕДЕЛЕННЫМ СИСТЕМАМ: ДИНАМИЧЕСКИЕ АЛГОРИТМЫ И ЭФФЕКТИВНАЯ ОБРАБОТКА БОЛЬШИХ ДАННЫХ

Батанов А.О., Литвинова А.С.

ФГБОУ ВО «МИРЭА – Российский технологический университет», Москва,
e-mail: arseny.batanov@yandex.ru, litvinova_a@mirea.ru

Стремительный рост объемов данных обуславливает необходимость разработки эффективных алгоритмов и технологий для их обработки в распределенных системах. Целью исследования является усовершенствование существующих алгоритмов обработки и распределения данных, учитывая текущую нагрузку на систему и общую загруженность. Анализ существующих методов обработки больших данных, таких как MapReduce, сегментирование и консистентное хеширование, подчеркивают важность развития адаптивных систем для обработки больших данных. Предложенная формула, включающая динамический коэффициент, позволяет значительно повысить производительность, масштабируемость и отказоустойчивость распределенных систем. Результаты моделирования демонстрируют, что предложенный подход позволяет достичь более эффективного использования ресурсов и предотвращает перегрузку отдельных узлов по сравнению с традиционными методами. На основе проведенного сравнительного анализа обоснована практическая применимость разработанной методики в широком спектре распределенных систем. Заключение подчеркивает стратегическую значимость развития адаптивных систем для обработки больших данных и интеграции современных технологий, таких как искусственный интеллект и машинное обучение. Предложенные усовершенствования открывают перспективы для дальнейших исследований и разработки более мощных и гибких распределенных систем, способных эффективно справляться с возрастающими требованиям современной цифровой среды.

Ключевые слова: распределенные системы, консистентное хеширование, масштабируемость, MapReduce, балансировка нагрузки, сегментирование

ADVANCED APPROACH TO DISTRIBUTED SYSTEMS: DYNAMIC ALGORITHMS AND EFFICIENT PROCESSING OF BIG DATA

Batanov A.O., Litvinova A.S.

MIREA – Russian Technological University, Moscow,
e-mail: arseny.batanov@yandex.ru, litvinova_a@mirea.ru

Rapid growth in data volumes necessitates the development of efficient algorithms and technologies for processing them in distributed systems. The aim of the research is to improve existing data processing and distribution algorithms by taking into account the current system load and overall congestion. The analysis of existing big data processing methods, such as MapReduce, Sharding, and Consistent Hashing, underlines the importance of developing adaptive systems for big data processing. The proposed formula, incorporating a dynamic coefficient, significantly enhances the performance, scalability, and fault tolerance of distributed systems. Modeling results demonstrate that the proposed approach achieves more efficient resource utilization and prevents the overloading of individual nodes compared to traditional methods. Based on a comparative analysis, the practical applicability of the developed methodology in a wide range of distributed systems is substantiated. The conclusion emphasizes the strategic significance of developing adaptive systems for big data processing and integrating modern technologies, such as artificial intelligence and machine learning. The proposed improvements open up prospects for further research and the development of more powerful and flexible distributed systems capable of effectively coping with the growing demands of the modern digital environment.

Keywords: distributed systems, consistent hashing, load balancing, scalability, mapreduce, adaptive algorithms, sharding, fault tolerance

Введение

Устойчивый рост объемов генерируемых данных продолжает сохранять внимание научного и инженерного сообществ к разработке более эффективных методов их обработки и глубокого анализа. Существующие алгоритмы и методы, такие как MapReduce (программная модель, разработанная для обработки больших объемов данных), Sharding (подход к разделению и распределению данных по множеству серверов или баз данных) и консистентное хеширование (алгоритм, предназначенный для распределенных систем, который обеспечивает равномерное распределение данных через кластер узлов с минимальными перераспределениями при добавлении или удалении узлов), хотя и демонстрируют достаточный уровень эффективности и возможностей применения, имеют ряд ограничений, связанных с масштабируемостью, производительностью и устойчивостью к

отказам. В распределительных системах, где особенно важно обеспечить распределение нагрузки и данных между узлами, такие ограничения создают острую необходимость в создании более эффективного метода. Стандартные подходы могут приводить к неравномерному распределению, что увеличивает риск перегрузок и снижает общую производительность системы. В эпоху больших данных эффективное распределение и обработка информации становятся важными вехами для функционирования распределенных систем. Традиционные алгоритмы демонстрируют ограничения по масштабируемости, производительности и балансировке нагрузки. Проблема заключается в необходимости разработки усовершенствованных алгоритмов, способных обеспечить более эффективное распределение данных, с учетом динамических условий эксплуатации систем.

Цель исследования – усовершенствование существующих алгоритмов обработки и распределения данных, учитывая текущую нагрузку на систему и общую загруженность для обеспечения более равномерного распределения ресурсов, повышения производительности и масштабируемости распределенных систем, снижения задержки и увеличения их общей устойчивости к отказам.

Материалы и методы исследования

В данном разделе внимание направлено на детальное рассмотрение технологий, стоящих в авангарде обработки больших данных, таких как MapReduce, сегментирование (англ. sharding), и консистентное хеширование. Осмысление их роли и ограничений в современных распределенных системах позволит выявить зоны потенциальных улучшений.

MapReduce – это программная модель, предназначенная для эффективной обработки больших объемов данных [1]. Основные фазы алгоритма – Map (отображение) и Reduce (сведение) – позволяют параллельно обрабатывать данные, значительно ускоряя процесс [1].

Функция Map – принимает на вход пары ключ-значение `(*k*, *v*)` и обрабатывает их, возвращая список промежуточных пар ключ-значение `(*k'*, *v'*)` [2]. Формулу функции Map можно представить так:

$$\mathrm{Map}(k,v)\rightarrow\left[(k_1',v_1'),(k_2',v_2'),\ldots,(k_n',v_n')\right]. \quad (1)$$

Функция Reduce – принимает на вход ключ `*k*'` и список значений `[*v'*]`, связанных с этим ключом, и агрегирует эти значения, возвращая результат в виде списка значений `*v*'` [2]. Формулу функции Reduce можно изобразить так:

$$\mathrm{Reduce}\left(k',[v'']\right)\rightarrow\left[v_1'',v_2'',\ldots,v_m''\right]. \quad (2)$$

Основные ограничения связаны с высокой задержкой обработки задач, ограниченной интерактивностью и неэффективностью при вычислениях [3]. Усовершенствование MapReduce включает оптимизацию фазы Map для сокращения времени обработки и улучшение фазы Reduce для более эффективной агрегации данных. Использование машинного обучения для предварительного анализа паттернов данных может дополнительно ускорить обработку.

Сегментирование (сегментирование базы данных) – методика горизонтального разделения данных на несколько сегментов («шардов»), каждый из которых хранится в отдельной таблице или базе данных. Этот процесс позволяет распределить данные и нагрузку по множеству серверов, улучшая таким образом производительность и масштабируемость системы. Основная цель сегментирования – обеспечить равномерное распределение данных и оптимизировать скорость доступа к ним [4]. Формула для определения целевого шарда конкретной записи выглядит следующим образом:

$$\mathrm{Shard}=\mathrm{hash}\left(key\right)\quad modn, \quad (3)$$

где $\mathrm{hash}\left(key\right)$ – хеш-функция, применяемая к ключу записи, (n) – общее количество доступных шардов.

Основные ограничения связаны с проблемами перераспределения данных при изменении количества шардов, с балансировкой нагрузки и сложностью масштабирования. Возможное усовершенствование сегментирования может быть связано с внедрением искусственного интеллекта и методов машинного обучения для балансировки нагрузки (прогнозирование нагрузки и адаптация в режиме реального времени), оптимизации хеширования (определение «горячих точек» нагрузки шардов) и динамического масштабирования (изменение количества шардов) [5].

Консистентное хеширование является фундаментальным механизмом в проектировании распределенных систем, способствующим улучшению балансировки нагрузки и уменьшению колебаний при динамических изменениях в кластере [6]. Основная идея этого подхода заключается в создании равномерно распределенного хеш-пространства, где каждый узел и каждый элемент данных имеют свой хеш, определяющий их позицию в кольце хеширования [7]. Виртуальные узлы позволяют од-

ному физическому узлу представлять собой несколько узлов в хеш-пространстве, что облегчает балансировку нагрузки между узлами. Благодаря этому, добавление или удаление узлов в системе приводит к минимальному количеству перемещений данных, что существенно уменьшает накладные расходы и повышает стабильность системы в условиях динамических изменений.

Формула, включающая коэффициент C, зависящий от текущей нагрузки системы, и общую нагрузку (total_load), позволяет оптимально распределить данные между узлами. Такой подход не только повышает эффективность обработки запросов, но и обеспечивает более равномерное распределение нагрузки, минимизируя риск перегрузки отдельных узлов.

Формула для расчета коэффициента *C* может варьироваться в зависимости от конкретных требований системы и выбранных метрик нагрузки. Однако базовая идея заключается в использовании текущей нагрузки системы для динамической адаптации распределения данных.

Формула для расчета коэффициента *C:*

$$C = \alpha\left(\frac{L_{\text{cpu,ideal}}}{L_{\text{cpu}}}\right) + \beta\left(\frac{L_{\text{ram,ideal}}}{L_{\text{ram}}}\right) + \gamma\left(\frac{L_{\text{net,ideal}}}{L_{\text{net}}}\right) + \delta\left(\frac{L_{\text{req,ideal}}}{L_{\text{req}}}\right), \quad (4)$$

где (L_{cpu}), (L_{ram}), (L_{net}), (L_{req}) – текущая нагрузка на систему по CPU, RAM, сетевой активности и количеству запросов соответственно, ($L_{\text{cpu,ideal}}$), ($L_{\text{ram,ideal}}$), ($L_{\text{net,ideal}}$), ($L_{\text{req,ideal}}$) – идеальные уровни нагрузки для каждого из этих параметров, (α), (β), (γ), (δ) – коэффициенты важности каждой метрики, отражающие их вклад в общую производительность системы. Сумма этих коэффициентов должна быть равна 1((α + β + γ + δ = 1)), чтобы обеспечить балансировку вклада каждой метрики.

Расчет общей нагрузки (total_load) в контексте консистентного хеширования и распределенных систем может быть выполнен разными способами, в зависимости от определения «нагрузки» в вашей системе и от того, какие метрики вы считаете важными. В общем случае, total_load представляет собой совокупную нагрузку на систему, которая может включать CPU, память, сеть, количество активных запросов и другие параметры. Вот несколько подходов к расчету total_load:

$$\text{total\_load} = \frac{\sum_{i=1}^{N}\left(w_{\text{cpu}} \cdot L_{\text{cpu},i} + w_{\text{ram}} \cdot L_{\text{ram},i} + w_{\text{net}} \cdot L_{\text{net},i} + w_{\text{req}} \cdot L_{\text{req},i}\right)}{w_{\text{total}}}, \quad (5)$$

где (*N*) – количество узлов в системе, ($L_{\text{cpu},i}$), ($L_{\text{ram},i}$), ($L_{\text{net},i}$), ($L_{\text{req},i}$) – значения нагрузки по CPU, RAM, сетевой активности и запросам на i-м узле соответственно, (w_{cpu}), (w_{ram}), (w_{net}), (w_{req}) – весовые коэффициенты для каждого типа нагрузки, отражающие их вклад в общую нагрузку системы, (w_{total}) – сумма всех весовых коэффициентов, используемая для нормализации результата.

Данный подход позволяет учесть вклад различных типов ресурсов в общую нагрузку системы, давая возможность более точно адаптировать распределение данных и нагрузку. Веса (w_{cpu}, w_{ram}, w_{net}, w_{req}) позволяют модифицировать важность каждого параметра в зависимости от специфики работы системы и требований к производительности. Для реализации такого расчета важно регулярно осуществлять сбор актуальных данных о загруженности системы по всем учтенным параметрам и адаптировать весовые коэффициенты на основе изменений в требованиях к системе или наблюдаемой производительности. Такой подход обеспечит гибкость системы в реагировании на изменения в нагрузке и оптимизировать распределение ресурсов для поддержания высокой производительности и стабильности. Консистентное хеширование находит широкое применение в различных областях, включая системы кеширования, распределенные базы данных и сети доставки контента (CDN) [8]. Этот подход обеспечивает надежность и доступность данных, минимизируя влияние изменений в топологии системы на производительность и устойчивость к отказам. Особенно значима данная модель для облачных вычислений и микросервисной архитектуры, где требуется высокая степень масштабируемости и гибкости в управлении ресурсами.

Результаты исследования и их обсуждение

Ниже представлена улучшенная формула консистентного хеширования с виртуальными узлами и адаптивным распределением, учитывающая текущую нагрузку

системы. Предложенный подход включает использование виртуальных узлов и адаптивного перераспределения данных с учетом текущей нагрузки системы, что позволяет минимизировать пиковые нагрузки и улучшить общую производительность системы.

1. Традиционная формула консистентного хеширования [9]:

$$\text{node} = \text{hash}\left(\text{data}\right) \quad modN, \tag{6}$$

где hash(data) – это хеш-функция для данных, N – количество физических узлов в системе.

2. Улучшенная формула с виртуальными узлами:

$$\text{virtual node} = \text{hash}\left(\text{data}\right) \quad mod\left(N \times V\right), \tag{7}$$

где V – количество виртуальных узлов, ассоциированных с каждым физическим узлом, что позволяет более равномерно распределить данные по физическим узлам и упрощает балансировку нагрузки.

3. Адаптивное распределение с учетом нагрузки.

Основываясь на анализе существующих алгоритмов и выявленных недостатках, предлагается ряд улучшений, направленных на оптимизацию процессов распределения и обработки данных в распределенных системах. Одним из ключевых предложений является введение динамического коэффициента в формулы распределения данных, который корректируется на основе текущей нагрузки и объема данных на каждом из узлов. Такой подход позволяет системе более гибко реагировать на изменения и поддерживать оптимальное распределение данных, минимизируя риски перегрузки отдельных узлов. Предлагается соотношение, обеспечивающее адаптацию системы к изменяющимся условиям, гарантируя равномерное распределение нагрузки и оптимальное использование ресурсов:

$$\text{Узел} = \left(\text{hash}\left(\text{data}\right) \times \frac{C}{\text{total\_load}}\right) \quad modN, \tag{8}$$

где hash(data) – хэш-функция данных; (C) – динамический коэффициент, зависящий от текущей нагрузки системы, (total_load) – суммарная нагрузка на систему; (N) – количество узлов в системе.

Эффективность предложенного подхода целесообразно оценить посредством анализа, представленного на рис. 1, где осуществляется сравнительное исследование традиционной и усовершенствованной формулы распределения нагрузки. Визуализированные результаты демонстрируют преимущества предложенного подхода, выраженные в более равномерном распределении ресурсов и улучшении ее адаптивности к динамическим изменениям эксплуатационных условий.

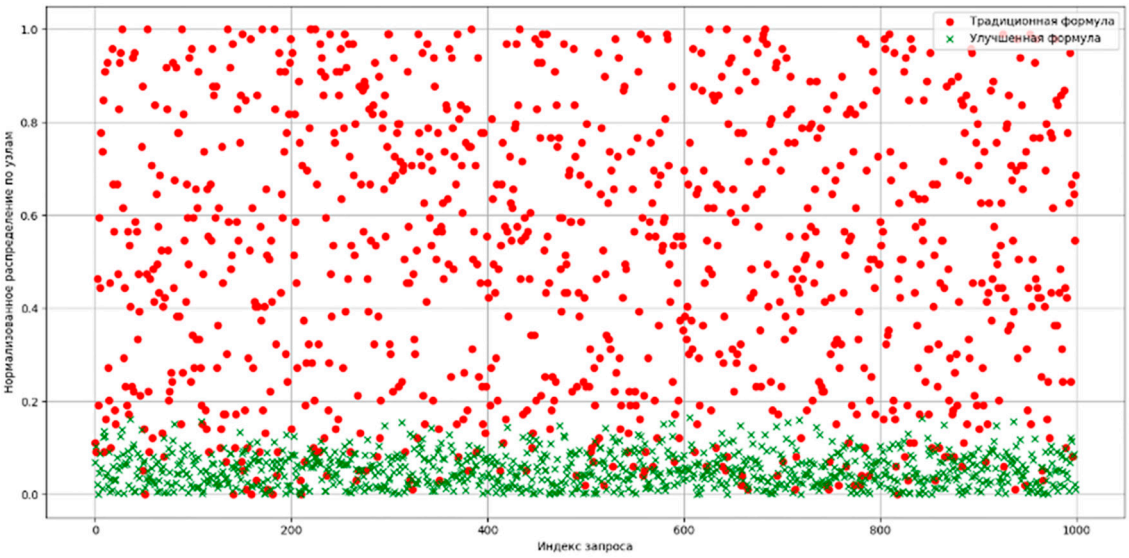

Рис. 1. Сравнение традиционного подхода и предлагаемого
Источник: разработано авторами

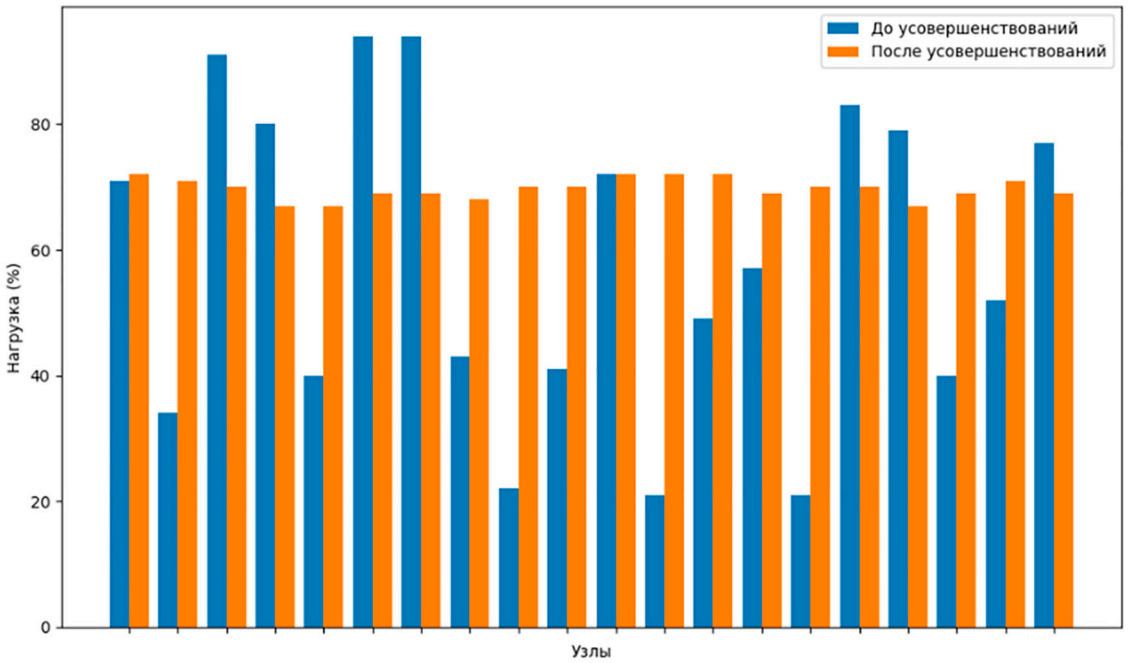

Рис. 2. Сравнение нагрузки на узлы при традиционном подходе и с учетом улучшений, предложенных в данной работе
Источник: разработано авторами

Применение традиционного подхода консистентного хеширования может привести к возникновению неравномерного распределения нагрузки между узлами, что иллюстрируется следующими показателями: 80, 60, 90 и 70 % соответственно. Такая диспропорция приводит к перегрузке одних узлов при недогрузке других, что негативно сказывается на эффективности использования ресурсов. Предложенный метод распределения данных, учитывающий как текущую загрузку системы, так и общую нагрузку, обеспечивает более равномерное распределение ресурсов между узлами. Это позволяет избежать дисбаланса, повышая справедливость и эффективность распределения, и обеспечивает стабильную работу системы без перегрузки или недогрузки отдельных узлов. Таким образом, предложенный метод позволяет достичь более справедливого и эффективного распределения ресурсов по сравнению со старым, гарантируя, что ни один узел не будет перегружен или недогружен. Такой подход улучшает устойчивость системы к внезапным изменениям в нагрузке и способствует более равномерному использованию ресурсов, что особенно важно для облачных вычислений и микросервисных архитектур, что подтверждается результатами моделирования, представленными на рис. 2.

Заключение

Анализ существующих методов и предложенные усовершенствования подчеркивают важность развития адаптивных систем для обработки больших данных. Преодоление ограничений традиционных алгоритмов через инновационные подходы к распределению нагрузки позволяет достичь оптимальной работы распределенных систем. Внедрение усовершенствованных алгоритмов и современных технологий обработки данных создает перспективы для дальнейших исследований и разработок. Интеграция искусственного интеллекта, методов машинного обучения и новейших научных достижений в существующие подходы позволяет существенно повысить эффективность и качество обработки данных. Это, в свою очередь, может создать условия для разработки высокоэффективных интеллектуальных систем, способных с максимальной точностью и скоростью решать сложнейшие задачи, что откроет перед человечеством новые перспективы для развития науки и технологий.

Список литературы

1. Jeffrey D., Sanjay G. MapReduce: Simplified Data Processing on Large Clusters // OSDI'04: Sixth Symposium on Operating System Design and Implementation. 2004. San Francisco. CA. P. 137–150. DOI: 10.1145/1327452.1327492.

2. Dean J., Ghemawat S., MapReduce: Simplified Data Processing on Large Clusters // Proc. of the 6th Symposium on Operating Systems Design and Implementation. USA. 2004. P. 137–150. DOI: 10.1145/1327452.1327492.

3. Богданов А.В., Тхуреин К.Л., Пья С.К.К., Чжо З. Сравнение производительности инструментов для обработки больших данных // Современные наукоемкие технологии. 2020. № 6–1. С. 9–14. DOI: 10.17513/snt.38064.

4. Andrew P., Matthew A. What's Really New with NewSQL? // SIGMOD Rec. 2016. Т. 45, № 2. P. 45–55. DOI: 10.1145/3003665.3003674.

5. Батанов А.О., Тюшкевич Н.М., Лашков С.А. Применение технологий искусственного интеллекта для улучшения процессов масштабирования в облачных экспертных системах // Научно-технический вестник Поволжья. 2024. № 11. С. 151–153.

6. Geetha J.J., Jaya Lakshmi D.S., Keerthana Ningaraju L.N. Consistent Hashing and Real-Time Task Scheduling in Fog Computing // Deep Learning Applications for Cyber-Physical Systems 2022. P. 245–261. DOI: 10.4018/978-1-7998-8161-2.ch013.

7. Greg H. Building a Consistent Hashing Ring // The OpenStack project. 2018. URL: https://docs.openstack.org/swift/newton/ring_background.html (дата обращения: 20.11.2024).

8. Dong C., Wang F., Feng D. DxHash: A Scalable Consistent Hashing Based on the Pseudo-Random Sequence // arXiv preprint arXiv. 2021. DOI: 10.48550/arXiv.2107.07930.

9. Karger D.R., Lehman E., Leighton F.T., Panigrahy R., Levine M.S., Lewin D. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. 1997. P. 654–663. URL: https://www.researchgate.net/publication/221590687_Consistent_Hashing_and_Random_Trees_Distributed_Caching_Protocols_for_Relieving_Hot_Spots_on_the_World_Wide_Web (дата обращения: 24.01.2025).