Санкт-Петербургский государственный университет
Математико-механический факультет

Математическое обеспечение и администрирование
информационных систем
Кафедра информационно аналитических систем

Слесарев Александр Германович

Исследование стратегий распределения данных в распределенных хранилищах

Дипломная работа

Научный руководитель:
к.ф.-м.н., доцент Михайлова Елена Георгиевна

Консультант:
ассистент Чернышев Георгий Алексеевич

Рецензент:
инженер ключевых проектов, ООО “Техкомпания Хуавей”
Морозов Сергей Валерьевич

Санкт-Петербург
2023

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Aleksandr Slesarev

A study of data distribution strategies in distributed storages

Graduation Thesis

Scientific supervisor:
Associate Professor, Ph.D. Mikhailova Elena

Consultant:
Assistant George Chernishev

Reviewer:
Key project engineer, Huawei Sergey Morozov

Saint-Petersburg
2023

Оглавление

Введение

Современные высоконагруженные приложения хранят данные в СУБД с несколькими узлами. Такая архитектура требует согласованности данных, а также важно обеспечить равномерное распределение данных между узлами. Для достижения этих целей используются алгоритмы балансировки данных.

Хеширование является основой практически всех систем балансировки данных. С момента появления классического консистентного хеширования для этой цели было разработано множество алгоритмов.

Одной из целей балансировщика нагрузки данных является обеспечение масштабируемости кластера с данными. Для производительности всей системы крайне важно перемещать как можно меньше записей таблиц во время добавления или удаления узла. Наибольшее влияние на этот процесс оказывает алгоритм хеширования, используемый балансировщиком нагрузки.

В данной работе произведена экспериментальная оценка нескольких алгоритмов хеширования, используемых для балансировки нагрузки. Проведен ряд экспериментов как модельного характера, так с реальными системами. Для оценки производительности алгоритмов был разработан тестовый стенд на основе Unidata MDM [20] — масштабируемого набора инструментов для решения различных задач управления мастер-данными (Master Data Management). Для оценки производительности были использованы три критерия, перечислены в порядке приоритета:

- Однородность полученного распределения данных по узлам;
- Количество перемещаемых при ребалансировке записей;
- Скорость вычислений.

По результатам экспериментов составлена таблица, в которой каждому алгоритму дана оценка по вышеуказанным критериям.

1. Постановка задачи

Целью данной работы является изучение влияния алгоритмов хеширования на процесс ребалансировки данных в распределенном хранилище. Для достижения этой цели были выделены следующие подзадачи:

- Выполнить обзор предметной области и существующих методов балансировки данных;
- Выработать критерии оценки алгоритмов балансировки;
- Реализовать подсистему ребалансировки данных в Unidata MDM;
- Провести эксперименты и оценить методы балансировки.

2. Обзор

2.1. Предметная область

По мере роста компании увеличивается и объем ее корпоративных данных. Существует два основных архитектурных подхода для решения этой проблемы [21]: вертикальное и горизонтальное масштабирование. Вертикальное масштабирование фокусируется на увеличении возможностей одного сервера, в то время как горизонтальное масштабирование предполагает добавление новых машин в кластер. Для реализации горизонтального масштабирования таблицу базы данных необходимо разделить по горизонтали на части (шарды), которые хранятся на разных узлах сервера.

Горизонтальное масштабирование имеет несколько существенных преимуществ, например возможность гибкой настройки объема хранилища путем изменения размера кластера. Также при горизонтальном масштабировании возможна репликация данных на разные узлы сервера, что снижает вероятность потери данных. Таким образом, появляется потребность в распределенном хранении данных.

Важным компонентом распределенного хранилища является балансировщик нагрузки — механизм, который определяет, какой конкретный узел будет хранить информацию об объекте (запись, часть таблицы и т. д.). Существует несколько критериев оценки балансировщиков нагрузки. Во-первых, распределение данных по серверам должно быть максимально равномерным. Далее, при изменении размера кластера количество перемещаемых объектов данных должно быть близким к оптимальному. И, наконец, затраты на вычисления балансировщика нагрузки не должны быть высокими.

Чтобы вычислить узел, назначенный объекту данных, балансировщик нагрузки использует алгоритм хеширования. С 90-х годов многие алгоритмы хэширования были разработаны специально для балансировки различных типов нагрузок, таких как управление сетевыми подключениями, оптимизация распределенных вычислений и балансиров-

ка хранилищ данных.

Одной из исследовательских дисциплин, посвященных хранению и обработке больших объемов данных, является управление основными данными или Master Data Management [1, 16] (MDM). Эта наука основана на понятии основных данных или master data. Концепция основных данных объединяет важные для бизнес-процессов организации объекты. Такими объектами могут быть запасы материалов, информация о клиентах или сотрудниках. Основными целями MDM являются унификация, согласование и обеспечение полноты основных данных компании.

2.2. Архитектура Unidata MDM

Чтобы понять специфику хранилища данных, в котором будут оцениваться балансировщики нагрузки, необходимо ввести некоторые термины MDM:

- **Эталонная запись (Golden Record)**. Одной из основных проблем в MDM является составление и поддержание "единой версии правды" или "single version of the truth" [1] для данного объекта, например человека, компании, заказа и т. д. Для достижения этой цели MDM система должна собирать информацию из многих источников данных (информационных систем конкретных организаций) в одну полную и непротиворечивую сущность, называемую эталонной записью.

- **Период валидности** — это интервал времени, в течение которого информация об объекте действительна. Для каждой эталонной записи может существовать несколько периодов валидности. Например, объект может иметь время создания и время обновления. Такие особенности необходимо учитывать при формировании запросов по обработке данных, что в свою очередь приводит к особенной схеме хранения при создании системы управления информацией.

MDM приложения представляют собой особый класс систем управления данными [20]. Их специфика предъявляет требования к архитектуре подсистемы хранения и обработке данных.

Во-первых, должна поддерживаться версионируемость хранимых объектов. По этой причине хранимые объекты имеют периоды действия.

Во-вторых, операции удаления объектов может выполнять только администратор, пользователь может только пометить объект как удаленный. Это необходимо для избежания потерь информации и обеспечения версионируемости. Также в некоторых случаях имеются юридические требования по хранению удаленных данных.

В-третьих, необходимо указать происхождение объекта. Это означает, что любая операция системы должна быть прослеживаемой. Например, нужно иметь способ отката всех изменений в записях после каждой операции.

Вышеописанные требования реализуются с помощью следующих четырех таблиц, три из которых нужны для описания объектов системы:

- Etalon хранит метаданные эталонной записи.
- Origin хранит метаданные, относящиеся к системе происхождения записи.
- Vistory (version history) — это период валидности соответствующей origin записи.
- External Key — это таблица, необходимая для доступа к данным из других подсистем платформы Unidata.

Отношения между этими таблицами показаны на рисунке 1. Ссылки с незакрашенными стрелками обозначают “общие” (унаследованные) атрибуты, а закрашенные стрелки показывают отношение PK-FK. Подробное описание атрибутов таблицы можно найти в [20].

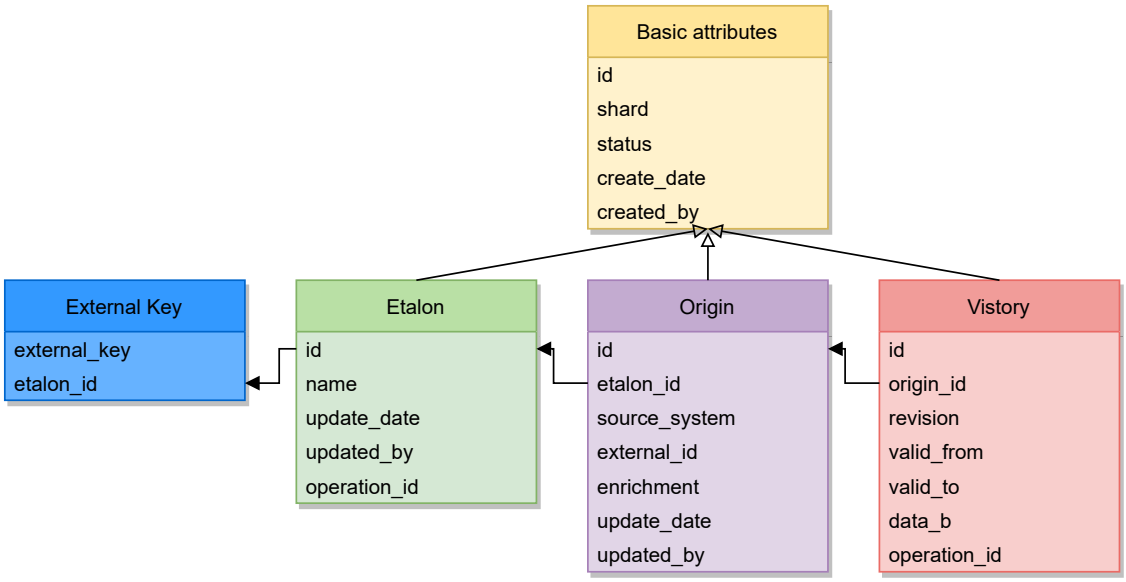


Рис. 1: Таблицы, используемые для хранения данных в платформе Unidata

3. Алгоритмы хеширования

В данной секции приведено описание алгоритмов хеширования. Каждый объект данных имеет ключ (key), по которому балансировщик нагрузки может вычислить номер шарда с данными.

- **Линейное хеширование (Linear Hashing)** — один из старейших алгоритмов. Помимо классической версии [10] существует ряд модификаций, таких как LH* [14], LH*M [13], LH*G [15], LH*S [7], LH*SA [11] и LH*RS [12]. Основная идея алгоритмов этого семейства заключается в вычислении остатка от деления ключа на количество шардов в системе. Поэтому они подходят для систем с фиксированным количеством шардов, что в данном случае является недостатком. С другой стороны, с ростом количества данных время поиска по ключу при линейном хешировании остается постоянным. В данной работе применяется версия, используемая для партиционирования в PostgreSQL[1].

[1]https://github.com/postgres/postgres/blob/master/src/backend/partitioning/partbounds.c

- **Консистентное хеширование (Consistent Hashing)**. Изначально этот алгоритм [3] был разработан для балансировки нагрузки компьютерных сетей. В настоящее время это один из самых популярных методов балансировки различных типов нагрузок. Например, распределенные системы, такие как AWS DynamoDB [4] и Cassandra [8], используют консистентное хеширование для шардирования и репликации. Этот метод основан на выборе случайных точек на кольце, которое представляет собой замкнутый отрезок вещественных чисел. Шарды и объекты данных представляются в виде точек. Точки, обозначающие ключи, присваиваются ближайшему шарду по часовой стрелке. Для обеспечения равномерного распределения данных каждый шард представлен несколькими точками. Устройство этого метода позволяет изменять количество шардов и при этом перемещать только нужные записи.

- **Rendezvous**. Подобно консистентному хешированию, этот метод [19] был разработан для оптимизации нагрузки на сеть. Также известен как алгоритм хеширования с наибольшим случайным весом (HRW). Он работает путем сопоставления ключа с шардом на основе случайно сгенерированного значения, которое затем хэшируется и сравнивается с набором заранее определенных весов для каждого шарда. Используя случайные значения, Rendezvous обеспечивает равномерное распределение нагрузки по всем шардам. Он также предоставляет механизм добавления или удаления шардов из системы без необходимости повторной балансировки всех данных. При перебалансировке Rendezvous не перемещает избыточное количество данных. Однако алгоритм требует, чтобы каждый шард (или сервер) поддерживал список всех других шардов, что может быть проблемой в крупномасштабных системах.

- **RUSH** [5] был разработан для хранения данных в дисковом кластере. Он имеет две модификации: $\mathrm{RUSH_R}$ и $\mathrm{RUSH_T}$ [6]. Алгоритм состоит из двух основных компонент: хеш-функции и схемы

присвоения ключей (key assignment scheme). Авторы RUSH сосредоточились на повышении равномерности распределения данных при изменении размера кластера, поэтому в основу алгоритма положен следующий принцип: при каждом изменении размера кластера используется специальная функция, которая решает, какие объекты следует переместить чтобы сбалансировать систему.

- **Maglev** — это алгоритм [17] балансировщика нагрузки Google для веб-сервисов. Целью Maglev является улучшение равномерности распределения данных (по сравнению с консистентным хешированием) и обеспечение минимального разброса записей. Это означает, что если набор шардов изменится, записи с большой вероятностью, будут отправлены в тот же шард, где они были раньше.

  Maglev был предложен как новый тип консистентного хеширования, в котором кольцо заменяется таблицей перестановок, с помощью которой ключ может быть назначен шарду. Таблица перестановок представляет собой двумерный массив, где каждый элемент содержит информацию о сервере и порядке его обхода. Количество строк в таблице является настраиваемым параметром, от его величины зависит частота коллизий. Число строк таблицы должно быть больше, чем возможное количество шардов, чтобы уменьшить частоту коллизий. Количество столбцов таблицы поиска равняется количеству шардов в системе. Среднее время поиска составляет $O(Mlog(M))$, где M — размер таблицы поиска.

- **Jump** — еще один балансировщик нагрузки от Google [9]. Авторы представили его как улучшенную версию консистентного хеширования, которая «не требует хранения, работает быстрее и лучше справляется с задачей равномерного распределения ключей между ячейками». Jump генерирует значения для номеров шардов только в $[0; \#]$, поэтому добавление нового шарда происходит быстро. Однако удаление промежуточного шарда приведет к повторному хэшированию многих записей. Сложность Jump равна $O(log(N))$, где N — количество шардов.

- **AnchorHash**. По словам авторов, AnchorHash [2] — это «техника хэширования, которая гарантирует минимальные нарушения, балансировку, высокую скорость поиска, низкий объем памяти и быстрое время обновления после добавления и удаления ресурсов». Заметным отличием AnchorHash от других обсуждаемых алгоритмов является то, что он хранит некоторую информацию о предыдущих состояниях системы.

  В алгоритме AnchorHash каждому шарду сопоставляется диапазон хеш-значений, который называется "bucket" (бакет, ведро, корзина). Ключевая идея алгоритма состоит в следующем. Пусть k - ключ, $\mathcal{W}$ - набор доступных бакетов, $H_{\mathcal{W}}$ — хеш-функция для сопоставления ключей набору $\mathcal{W}$. При удалении бакета $b \in \mathcal{W}$ AnchorHash продолжает использовать $H_{\mathcal{W}}$ до тех пор, пока $H_{\mathcal{W}}(k) \neq b$. В противном случае используется $H_{\mathcal{W}\setminus b}(k)$.

4. Эксперименты

4.1. Подготовка экспериментов

Для ответа на поставленный вопрос была реализована тестовая система шардирования внутри платформы Unidata. исходный код алгоритмов был взят из соответствующих Github репозиториев и адаптирован для использования в Unidata. Эксперименты проводились на следующем оборудовании:

- Аппаратное обеспечение: LENOVO E15, 16GiB RAM, Intel(R) Core(TM) i7-10510U CPU @ 4.90GHz, TOSHIBA 238GiB KBG40ZNT.
- Программное обеспечение: Ubuntu 20.04.4 LTS, Postgres 11.x, JDK 11.x, Tomcat 7.x, Elasticsearch 7.6.x.

Некоторые алгоритмы имеют параметры, влияющие на производительность:

- Для консистентного хэширования было выбрано 16 точек для каждого шарда. Это число было выбрано экспериментально, как компромисс между скоростью хеширования и равномерностью начального распределения.
- Для Maglev размер таблицы перестановок был установлен равным 103. Как и в случае с консистентным хешированием, число было выбрано экспериментально. Это значение важно в процессе перебалансировки, но не для поиска.
- Для AnchorHash было выбрано значение $|\mathcal{A}|$ (количество сегментов, с которыми работает алгоритм), равное удвоенному числу шардов (т.е. 64), как это было рекомендовано в оригинальной статье [2].

4.2. Результаты

Для оценки алгоритмов балансировки данных были выработаны три критерия, ранжированные по степени важности (в порядке убывания):

1. Равномерность распределения получаемых данных.
2. Избыточное перемещение записей при добавлении или удалении шарда.
3. Скорость поиска.

Для выбора лучшего алгоритма были проведены три эксперимента. Во-первых, был проведен эксперимент по моделированию балансировки данных в Google Colab[2] (на Python). Этот шаг был необходим для проведения предварительной оценки производительности алгоритмов. Эксперимент проводился следующим образом: сначала было сгенерировано 10000 записей и распределено (через хеш-функцию) на 32 шарда. Таким образом, равномерное распределение приведет к 312 записям на шард. После этого было удалено 8 шардов. В ходе удаления система перебалансировала данные. Таким образом, при равномерном распределении должно быть 416 записей на шард. Эта процедура выполнялась для каждого из рассмотренных балансировщиков данных.

Средние значения десяти экспериментов представлены в таблице 1. В первом столбце таблицы указано среднее время расчета шарда, во втором и третьем – дисперсия распределения записей до и после ребалансировки соответственно. В последнем столбце показано отношение количества фактически перемещенных записей к оптимальному количеству.

Основываясь на результатах эксперимента, было решено исключить $\mathrm{RUSH_R}$ из дальнейшего рассмотрения, так как он не соответствует всем трем критериям. Также был исключен Jump из-за плохого качества ребалансировки.

[2]https://colab.research.google.com/drive/1pbJUFFP9JsSTSn7nrWv0tYdiUg2uRxAv?usp=sharing

Таблица 1: Первый эксперимент, симуляция в Colab

Алгоритм	Время вычисления шарда (ns)	Дисперсия перед удалением	Дисперсия после удаления	Отношение перемещенных записей
Consistent	55049	72	94	1.00
Rendezvous	105331	18	21	1.00
RUSH_R	547044	95	125	1.57
Maglev	1146	16	21	1.39
Jump	18077	21	29	3.63
AnchorHash	3539	17	20	1.00

Следующие эксперименты связаны с платформой Unidata, которая реализована на Java. Чтобы проверить согласованность и переносимость ранее полученных результатов, было решено провести оценку времени расчета шарда внутри платформы. Поэтому второй эксперимент также заключался в распределении 10000 записей по шардам. Измеренные средние значения были следующими:

- Linear Hashing — 808ns
- Consistent Hashing — 2419ns
- Rendezvous — 5945ns
- Maglev — 807ns
- AnchorHash — 2015ns

Можно сделать вывод, что порядок времени выполнения алгоритма не изменился по сравнению с предыдущим экспериментом. Таким образом, можно сделать вывод, что смена языка программирования не повлияла на результаты предыдущего эксперимента, можно продолжать использовать платформу Unidata.

Третий эксперимент был выполнен в платформе Unidata, конфигурация хранилища была следующей: четыре Docker контейнера с PostgreSQL по восемь шардов на каждом узле. В качестве рабочей нагрузки было сгенерировано 10000 внешних ключей и эталонных запи-

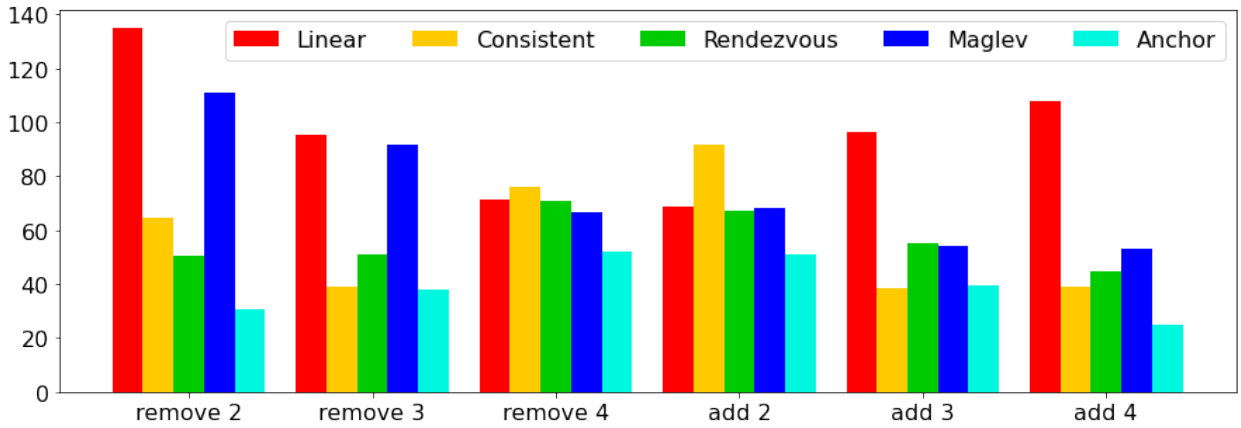


Рис. 2: Время ребалансировки (секунды).

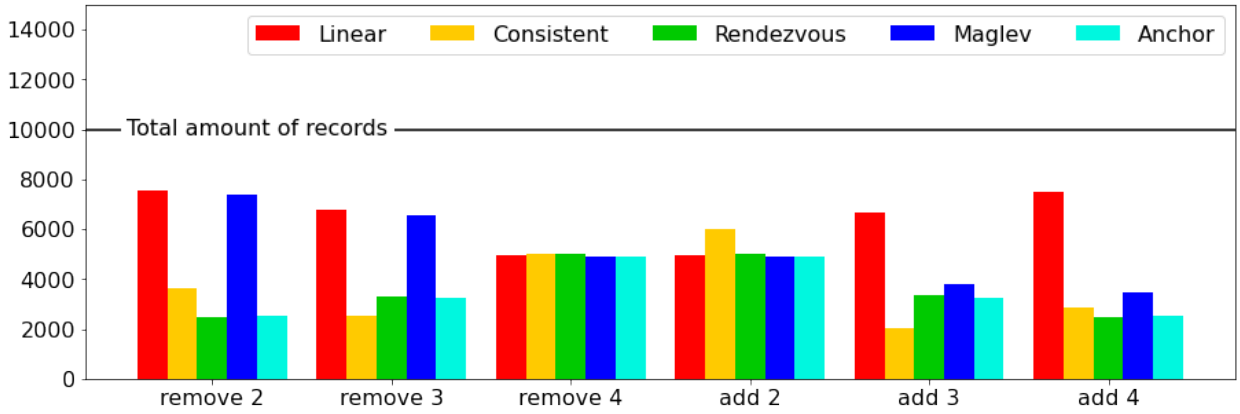


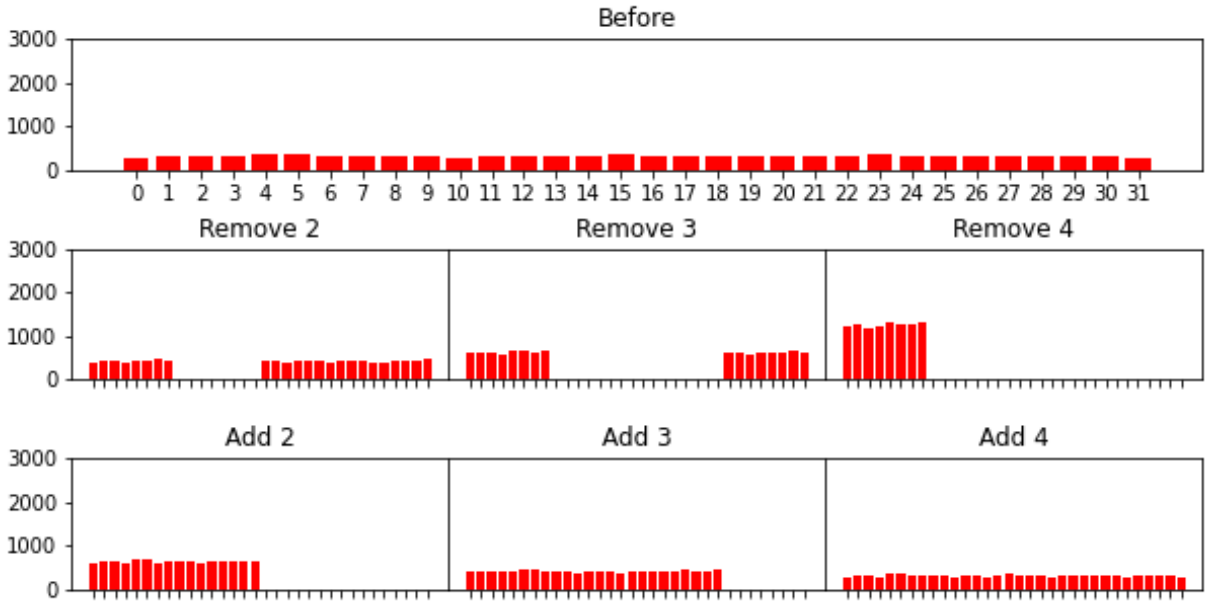
Рис. 3: Количество эталонных записей, перемещенных в процессе ребалансировки.

сей. Идея эксперимента заключалась в следующем: удалить три узла один за другим, а затем аналогичным образом добавить их обратно.

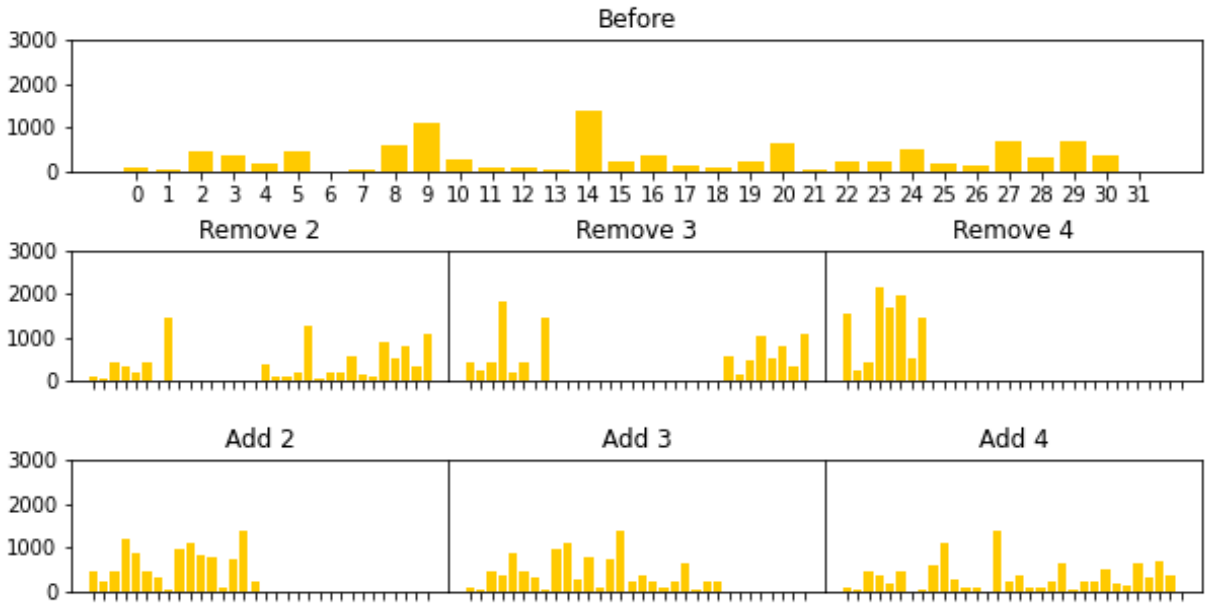
Результаты эксперимента показаны на следующих рисунках. Общее время, затраченное на каждый шаг перебалансировки, показано на рисунке 2, количество перемещенных эталонных записей показано на рисунке 3. Количество перемещенных внешних ключей опущено, так как оно практически такое же (это отображение 1:1). Распределение данных между шардами на каждом шаге показано на рисунках 5 и 6.

Третий эксперимент позволяет сделать следующие выводы:

- Consistent Hashing, Rendezvous и AnchorHash перемещают на 50% меньше записей чем Linear Hashing.
- В течение первых двух этапов ребалансировки Maglev переместил

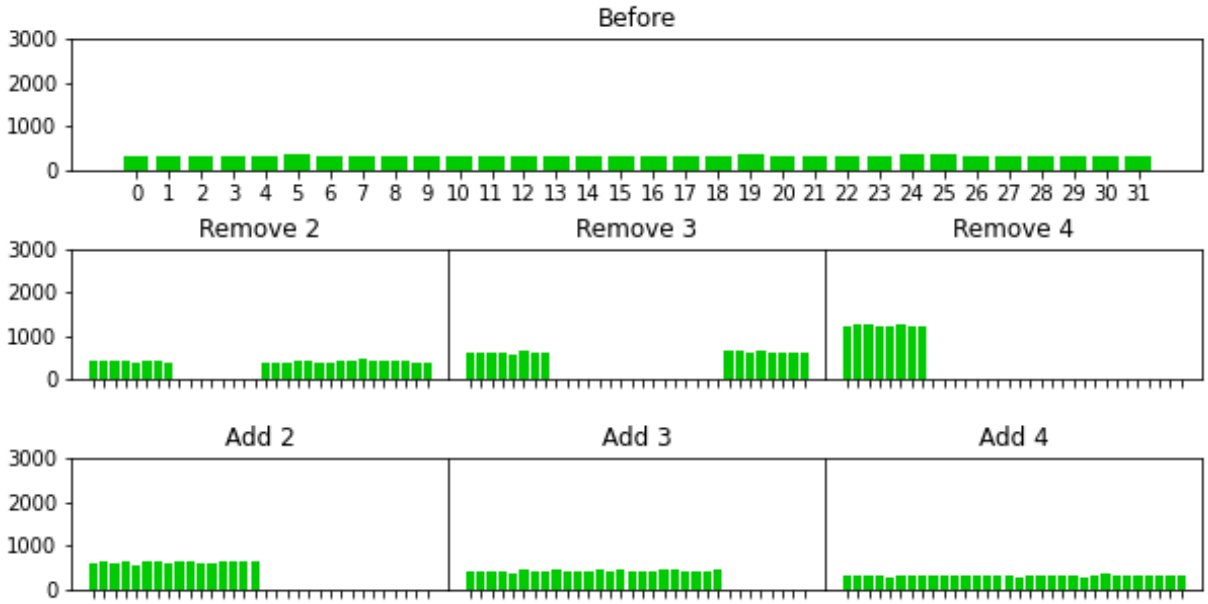


(a) Linear Hashing.

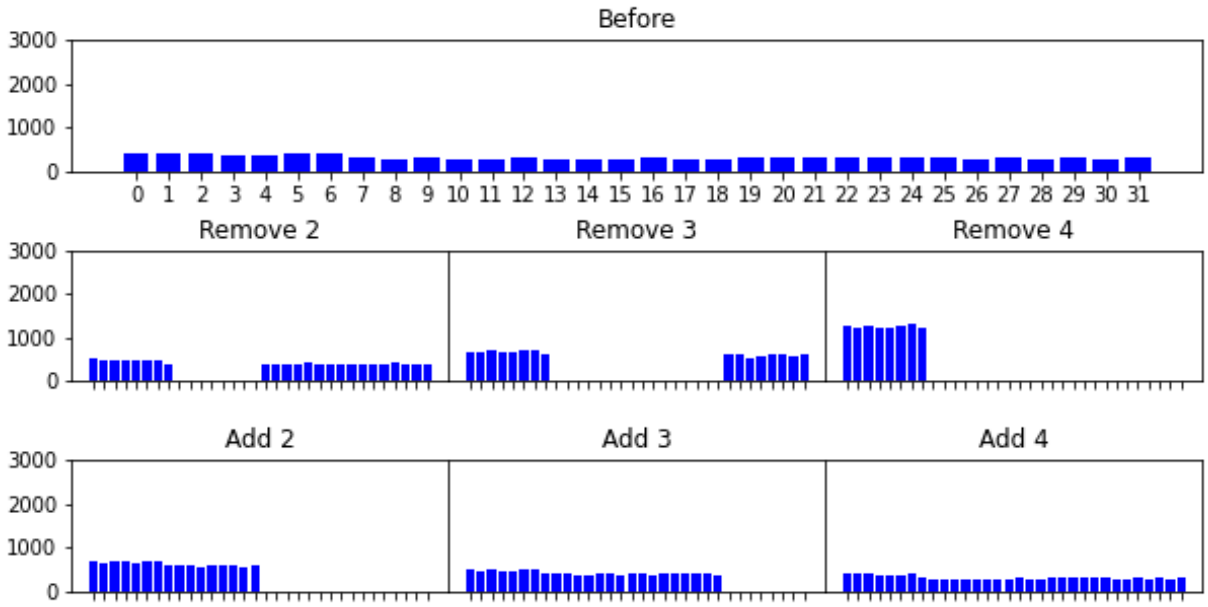


(b) Consistent Hashing.

Рис. 4: Распределение эталонных записей по шардам.

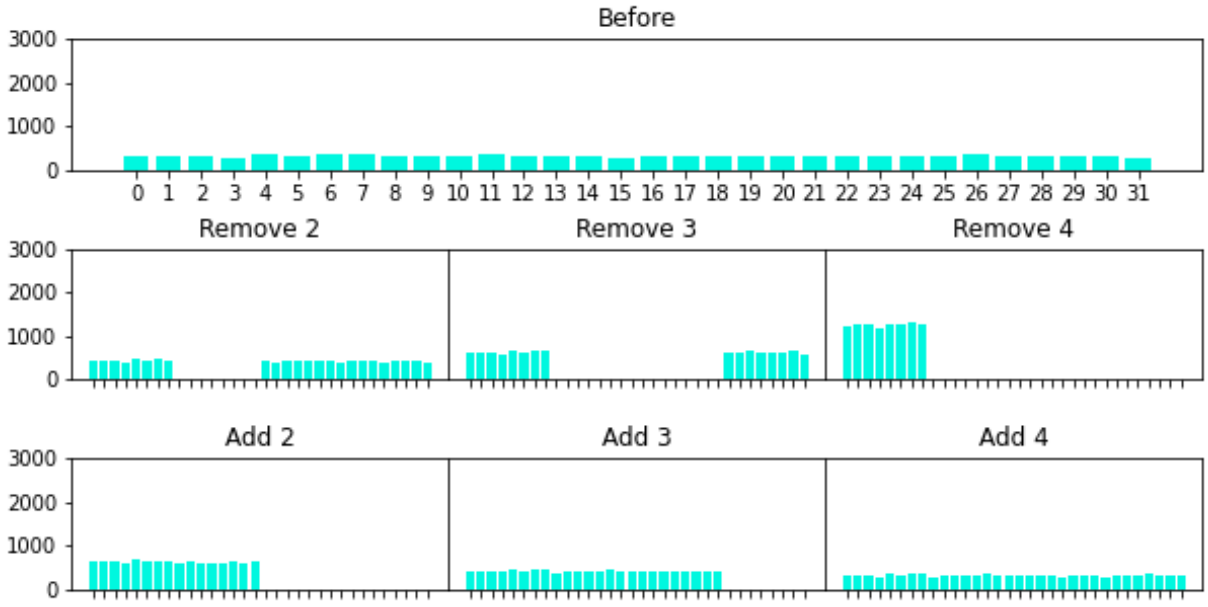


(a) Rendezvous.



(b) Maglev.

Рис. 5: Распределение эталонных записей по шардам.



(a) AnchorHash.

Рис. 6: Распределение эталонных записей по шардам.

примерно такое же количество записей как Linear Hashing, но на последних двух этапах Maglev переместил значительно меньше записей и приблизился по этому показателю к оставшимся трем методам.

- Linear Hashing, Rendezvous, Maglev и AnchorHash имеют достаточно равномерное распределение данных, в то время как Consistent Hashing распределяет записи недостаточно равномерно.

Далее обсуждается соответствие каждого из рассмотренных алгоритмов критериям, определенным в начале раздела.

- **Linear Hashing** имеет равномерное распределение записей по шардам и высокую скорость поиска, но перемещает до 80% записей на каждом шаге перебалансировки, поэтому этот метод не соответствует критериям. Однако Linear Hashing можно применять в системах с постоянным количеством шардов.

- **Consistent Hashing** имеет приемлемое время поиска и перемещает оптимальное количество записей, но распределяет данные крайне неравномерно. Следует отдавать предпочтение методам

с более равномерным распределением. Для улучшения качества раздачи можно увеличить количество точек для каждого шарда на кольце, но это замедлит поиск.

- **Rendezvous** оптимален в случае ребалансировки и распределения данных, но имеет самое большое время поиска. Поскольку скорость поиска является наименее приоритетным критерием, этот метод считается подходящим.

- **RUSH$_\mathbf{R}$** не удовлетворяет всем трем критериям, поэтому не подходит для поставленной задачи.

- **Maglev** обеспечивает быстрый поиск и относительно равномерное распределение, но в некоторых случаях он может перемещать более 50% всех записей (см. горизонтальную линию на рисунке 3). Поэтому Maglev подходит для систем с фиксированным количеством шардов.

- **Jump** переместил наибольшее количество записей (Таблица 1), так что этот метод тоже не подходит.

- **AnchorHash** кажется наиболее эффективным, поскольку удовлетворяет всем требованиям.

По результатам всех трех экспериментов создана таблица, в которой перечислены все оцененные алгоритмы (Таблица 2). Алгоритмы оценены в соответствии с тремя критериями, им присвоен рейтинг: низкий, средний и высокий.

Из таблицы видно, что есть два алгоритма-победителя — Maglev и AnchorHash, которые при этом не достигают ни высшего качества ребалансировки (количество перемещенных записей), ни максимальной скорости поиска.

AnchorHash равномерно распределяет данные, перемещает оптимальное количество записей, а время поиска достаточно мало. Rendezvous также соответствует первому и второму критерию, но его время поис-

Таблица 2: Таблица соответствия критериям балансировщиков нагрузки

Алгоритм	Равномерность распределения данных	Качество ребалансировки	Скорость поиска
Linear	Высокий	Низкий	Высокий
Consistent	Низкий	Высокий	Средний
Rendezvous	Высокий	Высокий	Низкий
RUSH	Низкий	Низкий	Низкий
Maglev	Высокий	Средний	Высокий
Jump	Средний	Низкий	Средний
AnchorHash	Высокий	Высокий	Средний

ка более чем в два раза больше, чем у AnchorHash. Эти два метода подходят для систем с частым добавлением или удалением шардов.

С другой стороны, поиск в Maglev более чем в два раза быстрее, поэтому он подходит для статических систем, аналогично Jump и Linear Hashing.

Консистентное хеширование кажется эффективным для обоих типов систем, но его главный недостаток — неравномерное распределение данных между шардами.

Также стоит отметить, что $\mathrm{RUSH_R}$ является худшим алгоритмом из всех.

Заключение

В данной работе было изучено влияние алгоритмов хеширования на процесс ребалансировки данных в распределенном хранилище. Для этого был выполнен обзор предметной области и изучено устройство следующих алгоритмов: Linear Hashing, Consistent Hashing, Rendezvous, RUSH, Maglev и Jump. Было выработано 3 критерия оцени применимости алгоритмов, а именно равномерность распределения полученных, количество перемещенных записей и вычислительные затраты. Также была реализована подсистема ребалансировки данных в платформе Unidata MDM и проведен ряд экспериментов: моделирование балансировки данных в Colab, оценка времени расчета шарда внутри платформы и ребалансировка платформы с использованием распределенного хранилища.

Эксперименты показали, что из семи рассмотренных алгоритмов есть два явных победителя — AnchorHash и Maglev. Еще два, Linear Hashing и Jump, также могут быть применимы.

По результатам работы опубликована статья “Benchmarking Hashing Algorithms for Load Balancing in a Distributed Database Environment” [18] на конференции MEDI 2022.

Список литературы

[1] Allen Mark and Cervo Dalton. Multi-Domain Master Data Management: Advanced MDM and Data Governance in Practice. — 1st ed. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2015. — ISBN: 0128008350.

[2] Mendelson Gal, Vargaftik Shay, Barabash Katherine, Lorenz Dean H., Keslassy Isaac, and Orda Ariel. AnchorHash: A Scalable Consistent Hash // IEEE/ACM Transactions on Networking. — 2021. — Vol. 29, no. 2. — P. 517–528.

[3] Karger David, Lehman Eric, Leighton Tom, Panigrahy Rina, Levine Matthew, and Lewin Daniel. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web // Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing. — New York, NY, USA : Association for Computing Machinery. — 1997. — STOC '97. — P. 654–663.

[4] DeCandia Giuseppe, Hastorun Deniz, Jampani Madan, Kakulapati Gunavardhan, Lakshman Avinash, Pilchin Alex, Sivasubramanian Swaminathan, Vosshall Peter, and Vogels Werner. Dynamo: Amazon's Highly Available Key-Value Store // SIGOPS Oper. Syst. Rev. — 2007. — oct. — Vol. 41, no. 6. — P. 205–220.

[5] Honicky R.J. and Miller E.L. A fast algorithm for online placement and reorganization of replicated data // Proceedings International Parallel and Distributed Processing Symposium. — 2003. — P. 10 pp.–.

[6] Honicky R.J. and Miller E.L. Replication under scalable hashing: a family of algorithms for scalable decentralized data distribution // 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings. — 2004. — P. 96–.

[7] Litwin W., Neimat M.-A., Lev G., Ndiaye S., and Seck T. LH*s: a high-availability and high-security scalable distributed data structure // Proceedings Seventh International Workshop on Research Issues in Data Engineering. High Performance Database Management for Large-Scale Applications. — 1997. — P. 141–150.

[8] Lakshman Avinash and Malik Prashant. Cassandra: A Decentralized Structured Storage System // SIGOPS Oper. Syst. Rev. — 2010. — apr. — Vol. 44, no. 2. — P. 35–40.

[9] Lamping John and Veach Eric. A Fast, Minimal Memory, Consistent Hash Algorithm. — 2014. — 06.

[10] Linear Hashing : Rep. ; executor: Alon Noga, Dietzfelbinger Martin, Miltersen Peter B, Petrank Erez, and Tardos Gabor : 1997.

[11] Litwin W., Menon J., and Risch T. LH* Schemes with Scalable Availability. — 2001. — 01.

[12] Litwin Witold, Moussa Rim, and Schwarz Thomas. $LH*_{RS}$—a Highly-Available Scalable Distributed Data Structure // ACM Trans. Database Syst. — 2005. — sep. — Vol. 30, no. 3. — P. 769–811.

[13] Litwin W. and Neimat M.-A. High-Availability LH* Schemes with Mirroring // Proceedings of the First IFCIS International Conference on Cooperative Information Systems. — USA : IEEE Computer Society. — 1996. — COOPIS '96. — P. 196.

[14] Litwin Witold, Neimat Marie-Anne, and Schneider Donovan A. LH: Linear Hashing for Distributed Files // Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. — New York, NY, USA : Association for Computing Machinery. — 1993. — SIGMOD '93. — P. 327–336.

[15] Litwin W. and Risch T. LH*g: a high-availability scalable distributed data structure by record grouping // IEEE Transactions on Knowledge and Data Engineering. — 2002. — Vol. 14, no. 4. — P. 923–927.

[16] Loshin David. Master Data Management. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2009. — ISBN: 978-0-12-374225-4.

[17] Eisenbud Daniel E., Yi Cheng, Contavalli Carlo, Smith Cody, Kononov Roman, Mann-Hielscher Eric, Cilingiroglu Ardas, Cheyney Bin, Shang Wentao, and Hosein Jinnah Dylan. Maglev: A Fast and Reliable Software Network Load Balancer // Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation. — USA : USENIX Association. — 2016. — NSDI'16. — P. 523–535.

[18] Slesarev Alexander, Mikhailov Mikhail, and Chernishev George. Benchmarking Hashing Algorithms for Load Balancing in a Distributed Database Environment // Advances in Model and Data Engineering in the Digitalization Era / ed. by Fournier-Viger Philippe, Hassan Ahmed, Bellatreche Ladjel, Awad Ahmed, Ait Wakrime Abderrahim, Ouhammou Yassine, and Ait Sadoune Idir. — Cham : Springer Nature Switzerland. — 2022. — P. 105–118.

[19] A Name-Based Mapping Scheme for Rendezvous : Rep. ; executor: Thaler David and Ravishankar Chinya. — Access mode: `https://www.eecs.umich.edu/techreports/cse/96/CSE-TR-316-96.pdf`.

[20] Kuznetsov Sergey, Tsyryulnikovv Alexey, Kamensky Vlad, Trachuk Ruslan, Mikhailov Mikhail, Murskiy Sergey, Koznov Dmitrij, and Chernishev George. Unidata — A Modern Master Data Management Platform // Proceedings of the 1st International Workshop on Data Platform Design, Management, and Optimization (DATAPLAT) co-located with the 25th International Conference on Extending Database Technology and the 25th International Conference on Database Theory (EDBT/ICDT 2022), Edinburgh, UK, March 29, 2022. — CEUR-WS.org. — 2022. — CEUR Workshop Proceedings.

[21] Özsu M. Tamer and Valduriez Patrick. Principles of Distributed Database Systems. — 3rd ed. — Springer Publishing Company, Incorporated, 2011. — ISBN: 1441988335.