ЛЕКЦИЯ 4
РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ И ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ

Первоначально основной предпосылкой разработки систем, использующих базы данных, являлось стремление объединить все обрабатываемые в организации данные в единое целое и обеспечить к ним контролируемый доступ. В таких централизованных системах единственная логическая база данных размещалась в пределах одной вычислительной системы и находилась под управлением одной СУБД. Хотя интеграция и предоставление контролируемого доступа имеют свои неоспоримые преимущества, централизация в ряде случаев, имеет и обратную сторону.

Исследования и разработка в области *распределённых баз данных* начались еще в 70-х годах прошлого столетия. Основной мотивацией была потребность в улучшении доступности данных, их резервировании и отказоустойчивости, а также в поддержке географически распределённых организаций. Распределённые базы данных позволили организациям управлять данными, которые физически разделены между различными узлами, иногда расположенными в разных географических регионах, что было особенно важно для многонациональных компаний и государственных учреждений. Совершенствование интернет-технологий и появление огромного количества информации повлекло за собой разработку новых типов распределенных СУБД, способных обрабатывать и быстро анализировать большие объемы данных, обеспечивая непрерывный доступ к информации в глобальной масштабе. Распределенные базы данных стали неотъемлемой частью инфраструктуры многопользовательских и высоконагруженных веб-приложений, обеспечивая высокую доступность, отказоустойчивость и горизонтальную масштабируемость.

Развитие *систем параллельной обработки данных* началось в 1980-х годах в связи с прорывом в текущих технологиях, когда стали доступны многопроцессорные и многоядерные системы. Эти методы позволили увеличить эффективность обработки больших объемов данных на единых вычислительных платфор-

мах, распределяя задачи и процессы между несколькими процессорами в рамках одной машины или кластера. Механизмы параллельного обработчика систем управления базами данных (СУБД) были направлены на улучшение производительности и пропускной способности, что стало критическим для приложений, предъявляющих высокие требования к скорости обработки больших массивов данных, например, в финансовом анализе и научных исследованиях. Сегодня эти технологии продолжают развиваться, находя применение в самых разных областях, и часто они дополняют друг друга, предоставляя мощные и гибкие инструменты для работы с данными.

Основные понятия распределенной обработки данных

Концепция распределения данных включает в себя управление данными, которые физически разделены между несколькими узлами. *Узлы* – это отдельные серверы или машины, которые хранят и обрабатывают данные в распределенной базе данных. Они могут быть физически или виртуально размещены в разных местах для улучшения производительности, обеспечения отказоустойчивости и сокращения времени задержки. Серверы могут быть объединены в кластеры для параллельной обработки данных, а также могут использовать облачные платформы для большего географического разнесения и эффективного управления ресурсами. Физическое распределение данных по разным узлам помогает в достижении лучшей масштабируемости, устойчивости к отказам и эффективности обработки данных, а также может способствовать соблюдению нормативных требований по хранению данных в определенных юрисдикциях.

Распределённая база данных (РБД) – это база данных, данные которой не хранятся централизованно, а распределены между несколькими узлами. Основной особенностью распределённой базы данных является то, что она представляет собой единую логическую базу данных для пользователя, несмотря на физическое разделение данных. Это позволяет выполнение запросов и транзакций на различных узлах таким образом, как будто они производятся на одном сервере.

Распределённая система управления базами данных (РСУБД) – это тип системы управления базами данных, который управляет распределённой базой данных и обеспечивает прозрачное управление разделёнными данными. Основной задачей РСУБД является скрытие сложностей распределённой природы данных от конечного пользователя, обеспечивая при этом эффективную обработку запросов и транзакций через различные узлы. Распределенная система управления включает в себя механизмы для обработки сбоев узлов и сетевых проблем, чтобы минимизировать влияние таких событий на общую работоспособность системы. Это требует сложной координации между узлами и часто включает в себя использование продвинутых алгоритмов для согласования данных и управления транзакциями.

Рассмотрим два примера, каждый из которых отражает разные аспекты использования распределенных баз данных. Первый пример демонстрирует географическую декомпозицию, связанную с разделением данных по регионам для размещения на серверах, находящихся в близости к тому месту, где они больше всего требуются. Второй пример показывает другую сторону – логическое разделение данных по функциональным областям согласно требованиям бизнеса.

Допустим, крупная международная компания хранит информацию о своих продажах в базе данных. Все продажи происходят в разных регионах мира: в Америке, Европе, Азии и т.д. В распределенной базе данных можно хранить информацию о продажах каждого региона на серверах, которые расположены непосредственно в этих регионах. Таким образом, европейские продажи хранятся на сервере в Европе, американские продажи – н а сервере в Америке и т.д. Когда группа сотрудников компании, которые занимаются аналитикой продаж, запросит данные о продажах в определенном регионе, запрос будет направлен на сервер, на котором хранятся данные этого региона, и информация будет быстро извлечена и передана обратно. Для пользователя это будет выглядеть так, как будто все данные хранятся на одном и том же сервере, независимо от их физического местоположения. Перечитывая глобальную книгу продаж в этом распределенной «библиотеке» продаж, запрос будет направлен в правильную «комнату» (регион), чтобы прочитать конкретный «том» (дан-

ные), который он ищет. Таким образом, запросы к распределенной базе данных могут быть быстро обработаны, поскольку данные находятся вблизи от места, где они нужны, и распределены по всей системе для лучшей производительности и надежности.

Второй пример связан с распределенной базой данных для интернет-магазина, которая сегментирована на разные серверы. Первый сервер – сервер клиентов. Этот сегмент хранит всю информацию о клиентах – их профили, историю покупок, контактные данные, и т.д. Приложение CRM (Customer Relationship Management) для управления клиентами может обрабатывать эти данные, обеспечивая управление взаимоотношениями с клиентами. На втором сервере размещен сегмент данных о продуктах. Этот сегмент содержит информацию обо всех продуктах – их описания, цены, наличие на складе, и т.д. Приложение WMS (Warehouse Management System) для управления запасами может использовать эти данные для складского учета и поставок продукции. Третий сервер Сегмент данных о заказах. Этот сегмент хранит информацию обо всех заказах – что покупали, кто покупал, статус заказа, и т.д. Приложение управления заказами OMS (Order Management System) отслеживает эту информацию для обработки и доставки заказов. Серверы могут физически быть расположены в одном и том же центре обработки данных, а могут и находиться в разных местах, в зависимости от требований ситуации. Все сегменты распределенной БД работают вместе, чтобы обеспечить бесшовное функционирование интернет-магазина. При этом, несмотря на то, что каждый сегмент обрабатывается отдельно, данные из всех сегментов доступны для последующего анализа и принятия решений. Разделение данных позволяет отдельно масштабировать разные части приложения, улучшать производительность и эффективность. Смысл этого разделения - оптимизация работы с данными, а не географическое разделение. Например, если интернет-магазин видит, что объемы продаж растут, он может увеличить мощность сервера, обрабатывающего данные о заказах, без воздействия на остальные части системы.

Оба примера дополняют друг друга, подчеркивая разные преимущества распределенных баз данных – улучшенную производительность, масштабирование и легкость развития системы.

К основным стратегиям распределения данных относятся: шардинг; репликация; балансировка нагрузки.

Шардинг – это процесс разделения большой базы данных на более мелкие, управляемые части, называемые шардами (англ., shard – часть, фрагмент), которые распределяются по различным серверам или узлам. Шардинг делится на горизонтальный и вертикальный.

Горизонтальный шардинг (рисунок 26) позволяет распределить данные по узлам кластера посредством равномерного разделения строк таблиц, в то время как вертикальный шардинг (рисунок 27) делит данные по столбцам, что улучшает доступ к специфическим столбцам данных разными приложениями.

Исходная таблица

ID	Имя клиента	Адрес клиента	Пункт отправления	Пункт назначения	Дата отправления	Дата доставки
1	Иванов А.В.	пр.Мира, 5	А	К	01.02.2024	10.02.2024
2	Петров Т.М.	ул.Садовая, 7	Г	К	01.02.2024	14.02.2024
. . .						
10 001	Ребров В.К.	ул.Победы, 2	Б	Н	03.03.2024	06.03.2024
10 002	Песков Т.Ф.	пер.Крутой, 9	Д	Р	28.03.2024	02.04.2024
. . .						

Горизонтальное шардирование

Часть 1

ID	Имя клиента	Адрес клиента	Пункт отправления	Пункт назначения	Дата отправления	Дата доставки
1	Иванов А.В.	пр.Мира, 5	А	К	01.02.2024	10.02.2024
2	Петров Т.М.	ул.Садовая, 7	Г	К	01.02.2024	14.02.2024
. . .						

Часть 2

ID	Имя клиента	Адрес клиента	Пункт отправления	Пункт назначения	Дата отправления	Дата доставки
10 001	Ребров В.К.	ул.Победы, 2	Б	Н	03.03.2024	06.03.2024
10 002	Песков Т.Ф.	пер.Крутой, 9	Д	Р	28.03.2024	02.04.2024
. . .						

Рисунок 26 – Пример горизонтального разделения таблицы

Вертикальное шардирование

Часть 1

ID	Имя клиента	Адрес клиента
1	Иванов А.В.	пр.Мира, 5
2	Петров Т.М.	ул.Садовая, 7
. . .		
10 001	Ребров В.К.	ул.Победы, 2
10 002	Песков Т.Ф.	пер.Крутой, 9
. . .		

Часть 2

ID	Пункт отправлени	Пункт назначения	Дата отправления	Дата доставки
1	А	К	01.02.2024	10.02.2024
2	Г	К	01.02.2024	14.02.2024
. . .				
10 001	Б	Н	03.03.2024	06.03.2024
10 002	Д	Р	28.03.2024	02.04.2024
. . .				

Рисунок 27 – Пример вертикального разделения таблицы

Репликация – это процесс создания и поддержания копий данных (реплик) на различных узлах или серверах для обеспечения высокой доступности, устойчивости к сбоям и улучшения производительности обработки запросов. Репликация обеспечивает копирование данных между узлами, что не только увеличивает отказоустойчивость системы, но и улучшает доступность данных. *Синхронная репликация* – это метод, при котором изменения данных на одном узле немедленно копируются на другие узлы. Транзакция считается завершённой только после того, как все копии данных успешно обновлены. Это обеспечивает высокую степень согласованности данных между узлами, но может привести к снижению производительности из-за необходимости ждать подтверждения от всех узлов перед завершением операции. *Асинхронная репликация* – это метод, при котором изменения данных, произведенные на одной копии (на одном узле) инициируют завершение транзакции без ожидания завершения этих изменений на узлах, хранящих реплики. Транзакция считается завершённой, как только данные записаны на первичный узел. Это улучшает производительность, так как операции записи завершаются быстрее, но может привести к задержкам в синхронизации между узлами, что создаёт риск некоторых несогласованностей в данных в случае сбоя. Таким образом, синхронная репликация гарантирует, что все копии данных всегда синхронизированы, в то время как асинхронная репликация может предложить более высокую производительность за счёт потенциальных задержек в синхронизации.

Балансировка нагрузки – это процесс равномерного распределения запросов и операций обработки данных между различными узлами или серверами в системе, чтобы оптимизировать ресурсы, улучшить общую производительность и предотвратить перегрузку отдельных узлов. Балансировка распределяет запросы на чтение и запись по узлам, оптимизируя использование ресурсов и предотвращая перегрузки. Запросы на чтение часто могут быть обслужены любым узлом, который содержит нужные данные, особенно если данные реплицированы. Запросы на запись требуют применения специализированных алгоритмов (или стратегий) для поддержания согласованности данных между узлами, чтобы обеспечить, что каждое изменение корректно отражается на всех копиях.

Классификация распределенных баз данных

Можно выделить следующие типы распределенных баз данных по способу организации данных (рисунок 28):

1. сегментированные РБД;
2. тиражированные или реплицированные РБД;
3. смешанные или неоднородные РБД.

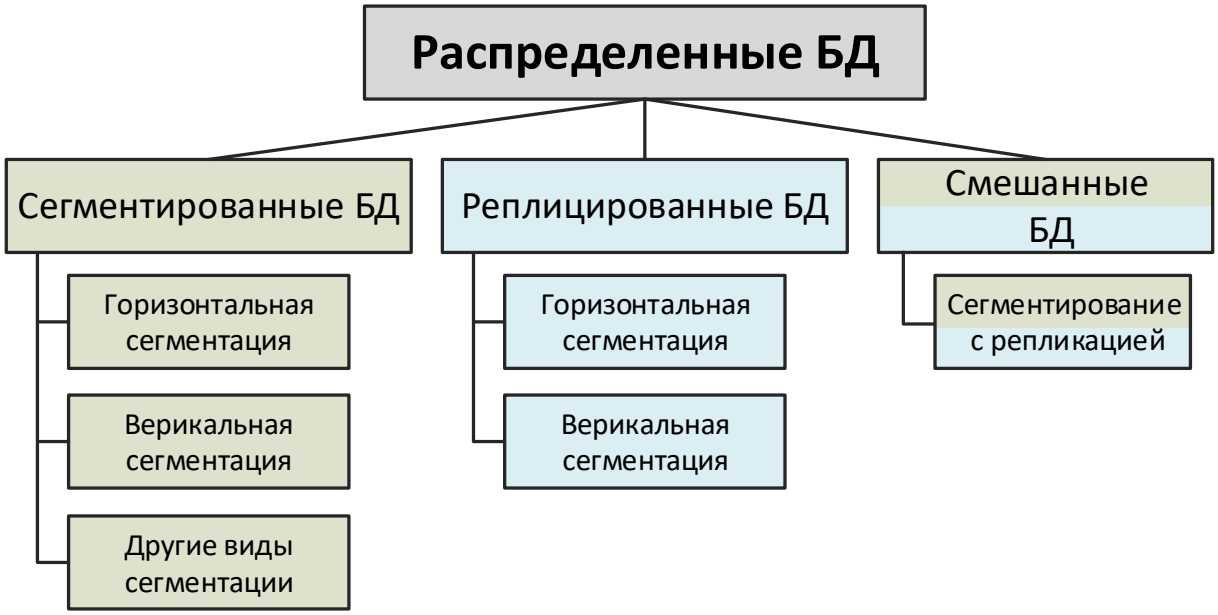

Рисунок 28 – Классификация распределенных баз данных

Сегментированные базы данных – это надежный и гибкий тип распределенной БД, в которой данные разделены на независимые сегменты, и каждый из этих сегментов управляется отдельным сервером БД. Важным моментом является то, что каждый сегмент хранит только свою часть данных и не знает о содержимом других сегментов. При распределении данных по отдельным сегментам возможно повышение эффективности обработки информации и оптимизация использования системных ресурсов. Каждый сервер сможет обрабатывать меньший объем данных, что позволит ему быстрее реагировать на запросы. Между тем, изоляция сегментов друг от друга позволяет обеспечить безопасность данных: даже при несанкционированном доступе злоумышленник сможет получить лишь часть информации. Такое разбиение данных позволяет улучшить производительность и масштабируемость системы, так как операции с данными могут выполняться параллельно в рамках каждого отдельного сегмента.

Тиражированные (или *реплицированные*) базы данных – это еще одна форма распределенной БД, которая позволяет обеспечить высокую степень надежности и доступности данных. В этом случае одни и те же данные хранятся на нескольких узлах, создавая несколько точек доступа к информации. Это позволяет обеспечить экономию времени при обработке запросов, так как система может обратиться к более близкому узлу. К тому же, в случае отказа одного из узлов, система будет продолжать функционировать, обеспечивая постоянный доступ к данным. Кроме того, репликация может улучшить производительность за счет балансировки нагрузки: запросы могут быть распределены между разными узлами, что позволяет более равномерно распределять нагрузку.

Реальные преимущества распределенных баз данных можно увидеть в совместном использовании сегментирования и репликации. Возможность сегментировать данные и затем реплицировать каждый сегмент позволяет создавать высокопроизводительные и надежные системы управления распределенными данными. Такой подход сочетает в себе преимущества обоих подходов: он обеспечивает как гибкость и масштабируемость распределенной системы, так и надежность и доступность реплицированных данных.

Принципы построения распределённых баз данных и распределенных СУБД

Принципы построения распределённых баз данных включают ряд фундаментальных концепций и методологий, которые обеспечивают их эффективность, надёжность, масштабируемость и доступность. Они тесно переплетаются с принципами распределённых СУБД, которые можно рассматривать как набор механизмов, обеспечивающих управление и координацию работы распределённой базы данных.

К основным принципам относятся:

- прозрачность распределения;
- независимость от сбоев;
- масштабируемость;
- согласованность данных;
- безопасность;
- эффективное распределение ресурсов;
- адаптивность и динамическое управление.

Прозрачность распределения – это основной принцип в проектировании и функционировании РБД и РСУБД. Этот принцип подразумевает, что сложности, связанные с распределённой природой данных, скрыты от пользователя, обеспечивая легкость в использовании и взаимодействии с системой, как если бы она была централизованной.

Для баз данных *прозрачность фрагментации* заключается в том, что пользователи не должны знать о том, как данные фрагментированы и распределены по разным узлам. Они должны считать, что взаимодействуют с базой данных, как если бы все данные были расположены локально. *Прозрачность репликации* означает, что пользователи не замечают наличия копий данных на разных узлах. Управление всеми копиями данных должно обеспечивать их актуальность и согласованность. *Прозрачность размещения* (локализации) подразумевает, что пользователи могут инициировать запросы, не задумываясь о том, где именно находятся данные. Необходимо автоматически определять местоположение данных и оптимальный маршрут для их получения.

Распределённые системы управления базами данных обеспечивают указанные виды прозрачности через ряд технических механизмов и архитектурных решений. РСУБД автоматически разделяют данные на фрагменты и управляют их распределением. Они создают и синхронизируют реплики данных, обеспечивая их актуальность и согласованность. РСУБД могут автоматически оптимизировать маршрутизацию запросов на основе географической близости или загруженности узлов, чтобы минимизировать задержки и улучшить производительность. Реализуя эти механизмы, РСУБД значительно упрощают управление и использование распределённых баз данных, делая сложную инфраструктуру невидимой для конечного пользователя и позволяя разработчикам сосредоточиться на бизнес-логике приложений, а не на технических деталях распределённой обработки данных.

Независимость от сбоя является ключевым принципом в построении распределенных баз данных, обеспечивающим безотказную работу системы даже при возникновении проблем на отдельных узлах. В большой и сложной системе какая-то часть в любой момент может перестать функционировать нормально. В данном контексте, независимость от сбоя означает, что отказ одного узла не должен приводить к сбою всей системы. Если один узел выйдет из строя, данные все равно будут доступны на других узлах. При одновременном выходе из строя нескольких узлов производительность может временно снизиться, поскольку остальные узлы должны переключиться и обработать увеличенную нагрузку, но, в общем и целом, система все еще будет функционировать. Это гораздо лучше того, что вся система вышла бы из строя, когда ее узлы с неповторяющимися данными перестали бы работать. Распределенные СУБД имеют механизмы обнаружения сбоев, которые могут определить, когда узел или сеть перестают работать нормально. После обнаружения сбоя система автоматически перераспределит запросы и задания на работающие узлы.

Горизонтальное масштабирование, часто называемое масштабированием «в ширину», является одним из самых важных плюсов использования распределенных баз данных. Оно предусматривает добавление новых узлов в распределенную сеть серверов, на которой работает РБД, для увеличения производитель-

ности и обработки большего объема данных без ограничения производительности. Вертикальная масштабируемость, хоть и возможна, обычно менее предпочтительна из-за высокой стоимости обновления оборудования и возможной простойной в работе системы.

Согласованность данных – это одно из ключевых свойств, которым должна обладать распределенная база данных. Оно обозначает, что все изменения данных должны быть согласованными по всем узлам в системе распределенных баз данных. Это означает, что когда происходят обновления, все узлы должны видеть те же самые данные. В РБД ситуация становится более сложной, поскольку данные могут быть распределены по множеству узлов. Если данные обновляются на одном узле, эти изменения должны быть переданы и приняты всеми остальными узлами. Эта операция может стать дорогостоящей и затруднительной, особенно в больших системах с массой записей, и может вызвать задержку в обновлении данных во всей системе, что может теоретически привести к проблемам с согласованностью. Стратегии для поддержания согласованности данных могут варьироваться. Существует множество различных протоколов согласованности, например, согласованное хеширование, двухфазная фиксация (2PC), трехфазная фиксация (3PC), протоколы Paxos и Raft. Они обеспечивают, что все копии данных остаются согласованными в течение всего времени жизни распределенной системы, даже если во время обновления данных возникают сбои или ошибки.

Однако, стоит заметить, что в некоторых системах согласованность может быть намеренно немного ослаблена в обмен на улучшение производительности или доступности. Такой подход называется «согласованность в конечном счете», когда система может временно допустить несогласованность данных, но после некоторого времени на всех узлах данные все же будут одинаковыми.

В целом, *безопасность в РБД* и РСУБД требует более тщательного планирования и реализации, чем в централизованных системах, из-за распределенной природы этих систем. Как и в централизованных системах, в распределенных базах данных необходимы механизмы контроля доступа. Пользователи или процессы имеют определенные права доступа к данным. Эти пра-

ва должны быть проверены при каждом запросе к базе данных. В распределенной системе это означает, что права должны проверяться на каждом узле, который участвует в выполнении запроса. При передаче данных между узлами могут использоваться открытые сети, поэтому шифрование в распределенной системе особо важно, поскольку помогает предотвратить неправомочный доступ к данным в случае их перехвата. Операции резервного копирования и восстановления в распределенной системе существенно усложняются, так как данные распределены между множеством узлов. Реляционные СУБД должны гарантировать, что все данные правильно резервируются и что в случае сбоя системы все данные могут быть восстановлены.

Принцип *эффективного распределения ресурсов* является еще одним ключевым принципом распределенных баз данных и систем управления базами данных. Этот принцип основывается на оптимальном распределении данных и процессов по различным узлам сети, чтобы улучшить производительность, уменьшить время ответа и повысить устойчивость к отказам. Одной из главных задач РСУБД при управлении ресурсами является сбалансированное распределение данных и процессов. Система управления определяет, какие фрагменты данных будут размещены на каких узлах для удобства доступа, равномерного распределения нагрузки и улучшения общей производительности. Это может быть основано на различных критериях, таких как частота использования данных, скорость доступа, физические характеристики устройства и др. Важно отметить, что эффективное распределение ресурсов может требовать от системы *способности к адаптивности и динамическому управлению*, что также является ключевыми принципами РСУБД. Это позволяет системе динамически изменять распределение ресурсов в ответ на изменение условий, таких как изменение нагрузки, сбои в работе отдельных узлов и т.д.

Поддержка всех этих принципов и требований обуславливает бóльшую сложность распределенных систем управления базами данных по сравнению с централизованными СУБД. Любые стратегии, механизмы и алгоритмы управления распределенной базой данных более трудоемки в проектировании и реализации, по сравнению с алгоритмами управления централизованной базой

данных. Однако преимущества в виде повышенной производительности, легкого масштабирования и доступности полностью оправдывают затраты на создание и поддержку РСУБД и обуславливают все более широкое применение распределенных баз данных во многих высоконагруженных и масштабируемых системах.

Теорема CAP и следствия из неё

В 2000 г. профессором Калифорнийского университета в Беркли Э. Брюером был предложен принцип организации вычислений в распределенных базах данных. Впоследствии он получил широкое признание в среде специалистов в области организации, хранения и использования данных. Теорема CAP играет важную роль при проектировании и использовании распределенных баз данных и систем управления базами данных, потому что она отражает ключевые трудности, которые возникают при одновременном обеспечении согласованности данных, доступности и устойчивости к разделению в таких системах. Следует отметить, что распределенные базы данных и СУБД являются всего лишь одним из примеров распределенных вычислительных систем, к которым применима данная теорема. Помимо СУБД, под распределенные системы попадают и другие технологии, например, системы обработки потоковых данных, распределенные файловые системы и системы координации.

Согласно теореме CAP при любой реализации распределённых вычислений возможно обеспечить не более двух из трёх следующих свойств:

– *согласованность данных* (англ. consistency) – во всех вычислительных узлах в один момент времени данные не противоречат друг другу;

– *доступность* (англ. availability) – лю бой запрос к распределённой системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают;

– *устойчивость к разделению* (англ. partition tolerance) – расщепление распределённой системы на несколько изолирован-

ных секций не приводит к некорректности отклика от каждой из секций.

Распределённые системы в зависимости от пары практически поддерживаемых свойств из трёх возможных распадаются на три класса – CA, CP, AP (рисунок 29)

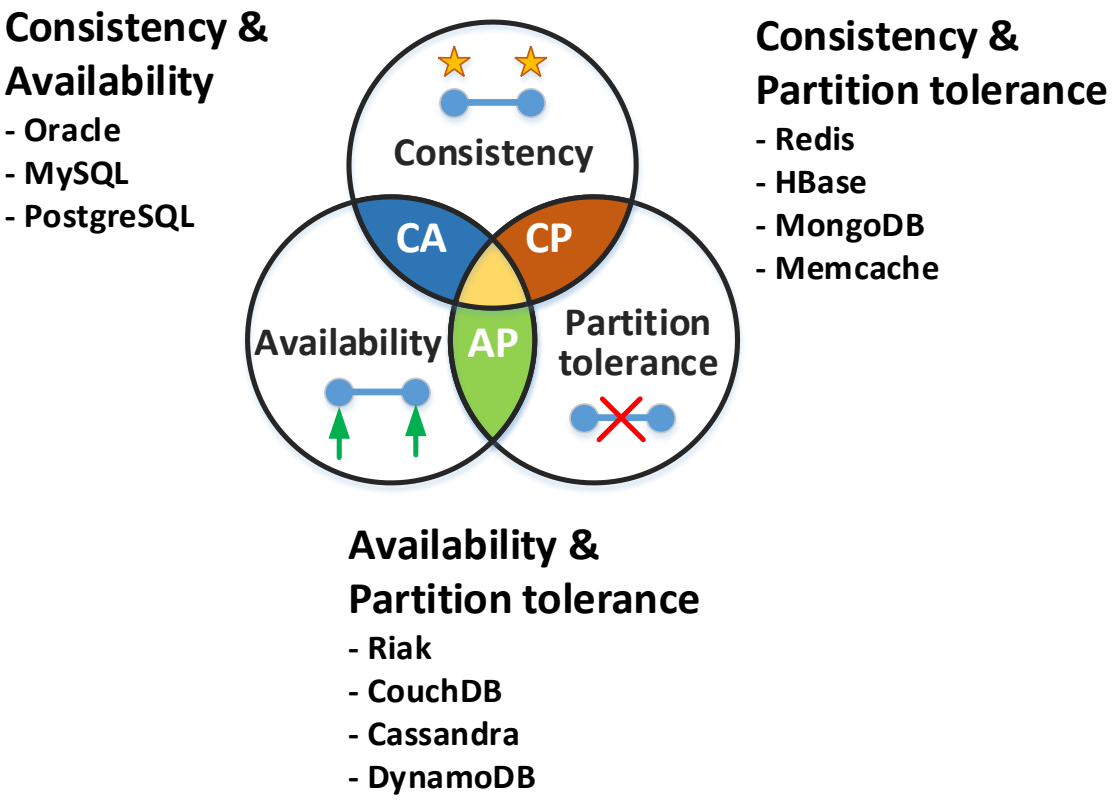

Рисунок 29 – Иллюстрация CAP-теоремы

В *системах класса CA* во всех узлах данные согласованы и обеспечена высокая доступность, при этом CA-системы жертвуют устойчивостью к распаду на секции. Каждый запрос на чтение данных получает самую свежую запись независимо от того, в какой узел поступит запрос. Например, если в БД записали данные «X = 1» в систему, то любой последующий запрос на чтение вернет «X = 1», при условии, что между этими двумя операциями не было другой записи. Это гарантия согласованности. В контексте систем класса CA, если система не может обеспечить согласованность данных из-за проблемы, она предпочтет вернуть ошибку, а не предоставить неконсистентные или устаревшие данные. Также эти системы обеспечивают высокую доступность. Это

означает, что в любой момент времени система готова принять запрос на чтение или запись. Системы класса CA способны обработать любой запрос от клиента в любое время.

Однако у систем версии CA есть один недостаток: они могут страдать от проблем в случаях *сетевых разделений*, т.е. в случае, когда сеть сообщающихся серверов перестаёт функционировать нормально и внезапно разделяется на две и более изолированных группы. Это может произойти из-за нескольких причин, таких как оборудование, сбои в программном обеспечении или даже физические причины, такие как подводный кабель, который поврежден акулой. Когда происходит такое разделение, узлы в одном сегменте не могут общаться с узлами в другом сегменте. Это может привести к тому, что данные не будут согласованы между узлами и, следовательно, привести к некорректной работе приложения и неожиданным ошибкам.

Классический пример системы CA – это традиционные SQL базы данных, такие как Oracle или MySQL. Они предоставляют согласованность данных и высокую доступность, но могут потерпеть небольшие проблемы при сетевых разделениях.

Система класса CP это тип систем, которые в первую очередь фокусируются на обеспечении согласованности данных и устойчивости к разделению системы на секции или сетевым разделениям. Это означает, что при работе, эти системы всегда обеспечивают целостный и последний результат. Если область сети исчезает или отделена, система все равно сможет функционировать корректно. Системы класса CP ориентированы на обеспечение того, что все клиенты видят одни и те же данные, даже в условиях, когда части сети не могут общаться друг с другом.

Однако такие системы достигают этого за счет доступности. Это означает, что системы могут временно отказаться откликаться на запросы на чтение или запись в попытке предотвратить возможное рассогласование данных между узлами. Когда происходит сетевое разделение, системы класса CP не гарантируют, что все запросы будут успешно завершены, но они гарантируют, что все завершенные запросы будут согласованны и актуальны.

Для поддержания целостности данных в таких системах часто используются *распределенные пессимистические блокировки*. Под «пессимистической блокировкой» понимается стратегия, при

которой система предполагает, что конфликты или проблемы будут происходить часто, и поэтому при одновременном обращении нескольких клиентов к одному и тому же ресурсу СУБД приостанавливает операции до разрешения конфликта. Это помогает предотвратить проблемы с целостностью данных.

К системам класса CP относятся распределённая нереляционная СУБД Apache HBase; нереляционные СУБД MongoDB и Redis, которые можно настроить для работы в режиме CP.

Системы класса AP – это те системы, которые, прежде всего, обеспечивают непрерывную доступность сервиса для пользователей и устойчивость к возникновению проблем с сетью или разделению секций. Это означает, что они всегда в состоянии отвечать на пользовательские запросы, даже если это может привести к рассогласованности данных между узлами в сети. В таких системах целостность данных не гарантируется полностью, иногда это может привести к тому, что данные, полученные в ответ на запрос, могут не быть актуальными в момент получения. По этой причине они особенно полезны для приложений, где доступность является критически важной характеристикой и в которых немного устаревшие данные не являются критической проблемой. Задачей при построении AP-систем становится обеспечение некоторого практически целесообразного уровня целостности данных, в этом смысле про AP-системы говорят как о *целостных в конечном итоге* (англ., eventually consistent) или как о *слабо целостных* (англ., weak consistent).

И хотя системы такого рода известны задолго до формулировки принципа CAP, рост их популярности связывается именно с распространением теоремы CAP. Поскольку большинство NoSQL-систем принципиально не гарантируют целостности данных, то они ссылаются на теорему CAP как на мотив такого ограничения. Примерами AP-систем являются нереляционные СУБД: Amazon DynamoDB, Apache Cassandra, Riak, которые обеспечивают высокую доступность и устойчивость к разделению данных, но это не всегда гарантируют согласованность данных.

Однако необходимо отметить, что за счет тонкой настройки конфигурации многие NoSQL-системы имеют возможность наряду с обеспечением гибких гарантий AP также предоставить более строгие гарантии согласованности CP, когда это необходимо.

Подход к построению распределенных систем BASE

Открытие теоремы CAP и понимание того, что в распределенной системе невозможно одновременно обеспечить согласованность, доступность и устойчивость к разделению, послужило катализатором для разработки нового подхода, который попытался найти баланс между этими требованиями, исходя из специфических нужд приложения и бизнес-требований.

Во второй половине 2000-х гг. был сформулирован подход к построению распределённых систем, в которых требования целостности и доступности выполнены не в полной мере, названый акронимом BASE (англ., Basically Available, Soft-state, Eventually consistent) – базовая доступность, неустойчивое состояние, согласованность в конечном счёте. Подход BASE применяется не только к системам управления базами данных, но и к развитию распределенных веб-сервисов и облачных приложений, где требуется обработка больших объемов данных и где постоянно актуальные и точные данные не являются критичными. Однако, как и СУБД, приложения, использующие подход BASE, разрабатываются с учетом возможности существования временной неконсистентности данных и должны быть готовы к обработке таких ситуаций. В общем, подход BASE может быть применим к любой системе или приложению, которое работает с распределенными данными и которое может жертвовать немедленной единообразностью и согласованностью в пользу доступности и ответов в реальном времени.

Базовая доступность означает, что в основном система всегда будет работать, независимо от того, произошли ли какие-либо сбои или нет. Если некоторые узлы перестанут работать, это затронет только небольшую часть системы. Большинство функций все еще будет доступно для использования. Это как если бы вышел из строя один компьютер в огромном офисе – да, это может вызвать проблемы для некоторых задач, но в целом работа офиса может продолжаться со многими другими компьютерами.

Неустойчивое состояние или *мягкие данные* означает, что данные, которые система сохраняет и использует, могут меняться со временем, даже если никакой новой информации не поступает. Это можно сравнить с автоматическим обновлением данных по

мере их старения. Например, в системах, где учитывается время, некоторые операции могут быть запланированы на исполнение через определенный период времени, что приводит к изменению состояния системы. При наличии нескольких узлов, выполняющих различные задачи и операции, может потребоваться время, чтобы все они были синхронизированы.

Согласованность в конечном счете подразумевает, что в течение какого-то времени данные в разных узлах могут выглядеть немного иначе, но со временем они станут обновлены и синхронизированы, в результате чего все узлы будут иметь одну и ту же информацию. Это как, например, когда люди спорят в интенсивной дискуссии – вначале у всех могут быть разные мнения, но по мере обсуждения и обмена информацией все приходят к общему заключению.

BASE позволяет системам быть очень надежными и масштабируемыми, однако ценой отказа от постоянной, жесткой согласованности данных. Это может быть приемлемым компромиссом во многих ситуациях, когда быстрый отклик на запросы пользователя является более высоким приоритетом, чем полная согласованность в каждый момент времени. Время, которое требуется для достижения конечной согласованности, может сильно варьироваться и зависит от многих факторов, включая степень распределения системы, объем данных, задержка сети, эффективность алгоритмов синхронизации и так далее. В некоторых системах несогласованность может длиться всего несколько миллисекунд или секунд, в то время как в других, особенно в очень больших и сложных, это может занять минуты или даже часы. Важно отметить, что системы, использующие подход BASE, обычно разработаны так, чтобы минимизировать возможные проблемы, вызванные временной несогласованностью.

Важно отметить, что выбор между подходами ACID и BASE в конечном итоге зависит от потребностей конкретного приложения или системы. Оба подхода имеют свои преимущества и недостатки, и применимость каждого из них зависит от особенностей использования.

Завершая обзор распределенных БД и распределенных вычислений необходимо отметить, что на данный момент, к 2024

году, ситуация с распределенными системами значительно изменилась по сравнению с предыдущим периодом. Эпоха преимущественно централизованных систем и одноузловых баз данных сменилась эрой масштабируемых и географически распределенных систем. Если в 1990-х годах распределенные вычисления только начинали применяться в промышленности, то теперь они стали обычной практикой. Распределенные системы стали общепринятыми как в промышленности, так и в академических кругах. В эксплуатации находится множество таких систем, и обширный опыт их использования накоплен. К тому же, увеличилось применение распределенных технологий, таких как облачные вычисления и блокчейн. Прогнозируется дальнейшее расширение использования и развития распределенных систем и технологий в различных отраслях и регионах.

Соответствующие знания и опыт также стали более доступными благодаря обучающим материалам, курсам, исследованиям, а также благодаря активному развитию и распространению таких технологий во всем мире.

Параллельная обработка данных

Параллельная обработка данных – это использование множества вычислительных ресурсов, таких как процессоры, память, диски с целью одновременной обработки больших объемов данных. Параллельность позволяет ускорить процесс получения результата и решать более сложные задачи.

Развитие архитектур вычислительных систем, переход от одно-процессорных к многопроцессорным системам с одной стороны, возрастающий объем данных, требующий новых подходов к обработке информации с другой стороны, стали решающими факторами появления систем параллельных вычислений.

Термин *Параллельность* относится к способу обработки данных системой управления базой данных В СУБД, поддерживающей параллелизм, одна и та же задача (множество запросов или один большой запрос) может быть разбита на подзадачи, которые затем запускаются одновременно на разных процессорах или узлах. Вместо того, чтобы обрабатывать миллионы записей силами

одного, пусть и очень мощного процессора, можно распределить эту задачу между несколькими процессорами и получить выигрыш в скорости за счет одновременного выполнения. Чем больше процессов или узлов используется на вычислительной платформе для параллельного выполнения запросов, тем выше становится производительность программных систем с БД.

Успешное выполнение параллельной обработки требует тщательного планирования и оптимизации. Важно правильно разбить задачи и следить за координацией работы между процессорами или серверами, чтобы предотвратить ошибки и неэффективное использование ресурсов. Успешно примененная параллельность в СУБД может значительно увеличить производительность и скорость обработки запросов, что особенно важно при работе с большими объемами данных.

Первой коммерческой СУБД с параллельной обработкой данных стала реляционная СУБД Teradata, разработанная в 1979 году компанией с одноименным названием. Teradata выполняет параллельную загрузку, архивирование и обработку огромных объемов данных – более 400 Тб в одной области. Основное преимущество Teradata заключается в её масштабируемости и способности обрабатывать упорядоченные данные огромных размеров. Teradata – это и параллельная, и распределенная СУБД. Она распределенная, потому что использует множество узлов для хранения данных, и она параллельная, потому что может выполнять множество операций одновременно. Термин «параллельная» часто используется для описания СУБД Teradata именно из-за ее способности выполнять множество задач одновременно для увеличения производительности и быстроты обработки запросо..

Языки реляционных БД, основанных на теории множеств, поддерживают широкие возможности для поддержки внутриоперационного параллелизма. Разница в написании SQL-запросов между однопроцессорной и параллельной СУБД может быть не так очевидной на уровне синтаксиса SQL-запросов, но затрагивает способ выполнения и оптимизации этих запросов.

Важные свойства параллельных СУБД

СУБД, поддерживающие параллельное выполнение запросов, реализуются три вида параллелизма, которые требуются приложениям с интенсивной обработкой больших объёмов данных.

Межзапросный параллелизм предполагает одновременное выполнение множества запросов, относящихся к разным транзакциям.

Внутризапросный параллелизм осуществляет одновременное выполнение нескольких операций, относящихся к одному и тому же запросу (например, сразу несколько операций выборки).

Внутриоперационный параллелизм означает параллельное выполнение одной операции в виде набора составляющих её субопераций (например, при выполнении запроса на выборку данных из таблицы, операции считывания данных с диска, их обработки и вывода результата могут выполняться параллельно).

Параллельная СУБД может быть реализована с помощью *архитектуры с разделением данных* или *архитектуры с разделением процессоров*. В первом случае, данные разбиваются на части и распределяются по различным процессорам. Во втором случае, все данные доступны каждому процессору и задачи, использующие эти данные, распределяются между процессорами.

Параллельные СУБД обладают свойством линейного ускорения и линейной масштабируемости.

Линейное ускорение (англ., linear speedup) связано с наращиванием процессорной мощности и объема памяти при неизменном размере БД. При таком расширении системы реорганизация структуры базы данных, как правило, минимальна. Например, если сервер БД обрабатывает 100 запросов в минуту на одном процессоре, то при использовании двух процессоров он сможет обрабатывать 200 запросов в минуту, а с четырьмя процессорами – 400 запросов в минуту, при условии идеальной координации и отсутствия других узких мест в системе.

Линейная масштабируемость (англ., linear scaleup) обеспечивает сохранение производительности программной системы при увеличении размера БД за счёт одновременного пропорционального повышения процессорной мощности и объема памяти. Например, компания, которая использует облачную инфраструк-

туру для хранения и анализа данных анализирует 50 ГБ данных в день с использованием определенного количества ресурсов. При увеличении объема данных до 100 ГБ в день и пропорциональном увеличении облачных ресурсов, система сможет анализировать этот увеличенный объем в те же сроки.

«Линейный» в этих терминах обозначает прямую пропорциональность между ростом ресурсов/данных и производительностью. В обоих случаях ключевой идеей является масштабирование, но фокус различен: линейное ускорение касается скорости работы с постоянным объемом данных при увеличении мощности, тогда как линейная масштабируемость связана с возможностью обрабатывать больший объем данных за тот же период времени благодаря увеличению мощности и памяти.

Параллельные СУБД используют самые различные вспомогательные технологии, вносящие свой суммарный вклад в общую производительность. В настоящее время все крупные разработчики СУБД внедряют параллелизм в специализированные, а значит, более дорогостоящие, версии создаваемых ими продуктов.

Взаимосвязь параллельных и распределенных СУБД

Не существует четкого разграничения между параллельными и распределенными СУБД. Однако есть основные характеристики, которые помогают сравнивать и различить эти два типа систем.

Общее между параллельными и распределенными системами баз данных заключается в том, что обе категории предназначены для обработки больших объемов данных, и повышению производительности систем, удовлетворяя при этом повышающиеся требования к масштабируемости и отказоустойчивости в современном мире обработки данных.

Оба типа систем предлагают собственные способы увеличения производительности и надежности систем.

Параллельные СУБД – это системы, которые встречаются в архитектуре с многопроцессорностью и выполняют множество операций одновременно на разных процессорах. Они часто применяются для ускорения обработки больших объемов данных,

распределенных по всем процессорам на основе различных техник распараллеливания.

С другой стороны, распределенные СУБД обрабатывают данные, которые располагаются на разных физических местах. Они предоставляют возможность совместного использования данных и ресурсов с несколькими пользователями и системами. Схемы распределения данных, такие как горизонтальное и вертикальное разделение, а также техника репликации, обычно используются в таких системах для обеспечения эффективной и надежной обработки данных. Одним из ключевых преимуществ таких систем является повышенная доступность данных, поскольку они могут продолжать работать даже при сбое одного из узлов.

Взаимосвязь между параллельными и распределенными СУБД проявляется в их способности работать совместно для достижения этих целей. Параллельные СУБД обычно используются в составе больших распределенных СУБД, где они помогают обрабатывать данные, разделенные между различными узлами сети. Другими словами, параллельная обработка может происходить на каждом из отдельных узлов в распределенной системе баз данных, позволяя системе быстро обрабатывать большие объемы данных. Благодаря этому сочетанию можно значительно увеличить скорость обработки запросов и операций, даже когда данные физически разделены в разных локациях.

Примеры распределенно-параллельных систем управления базами данных включают в себя продукты от различных компаний, таких как СУБД IMS/VS Data Sharing от компании IBM, СУБД VAX DBMS и Rdb от компании DEC. Компания Oracle также предлагает свои решения, реализованные на основе такой архитектуры. Эти системы управления базами данных разрабатываются с учетом особенностей параллельной и распределенной обработки данных, предлагая высокую производительность и эффективность, а также улучшенную повторную использование ресурсов и отказоустойчивость.