Базы Данных

Семинар 13

Описание семинара: In-Memory DataBase

In-Memory DataBase

Объемы баз данных и сложность запросов к ним всегда росли быстрее, чем скорость их обработки. Поэтому лучшие умы человечества много лет думали о том, что произойдет, когда оперативной памяти станет столько, что можно будет всю базу данных взять и поместить в кэш оперативной памяти.

В последние годы логический момент для этого, казалось бы, настал. Стоимость оперативной памяти падала, падала, и упала совсем. Еще в начале века казалось, что 256 МБ памяти для сервера — это нормально, и даже много. Сегодня нас не удивишь параметром 256 ГБ оперативной памяти на сервере начального уровня, а с промышленными серверами вообще настал полный коммунизм, любой благородный дон может набрать хоть терабайт оперативной памяти на сервере, если захочет.

Но дело не только в этом — появились новые технологии индексирования данных, новые технологии компрессии данных — OLTP-компрессия, компрессия неструктурированных данных (LOB Compression). В Oracle Database 11g, например, появилась технология Result Cache, которая позволяет кэшировать не просто строки таблиц или индексы, но и сами результаты запросов и подзапросов.

То есть, с одной стороны, наконец-то можно использовать оперативную память по ее прямому назначению, но с другой стороны — не так все просто. Чем больше кэш, тем больше накладные расходы на его сопровождение, включая процессорное время. Вы ставите больше памяти, увеличиваете объем кэша, а система работает медленнее, и это, в общем-то, логично, потому что алгоритмы управление памятью, разработанные в Раннем средневековье нашими прапрадедушками, попросту не годятся для эпохи Возрождения, и все тут. Что же делать?

Давайте вспомним о том, что существует, по сути дела, две категории баз данных: строчные базы данных, которые и в буферном кэше в оперативной памяти, и на диске хранят информацию в строчном виде — Oracle Database, Microsoft SQL Server, IBM DB/2, MySQL и т.д.; и колоночные СУБД, в которых информация хранится по столбцам, и которые большого распространения в индустрии, к сожалению, не нашли. Строчные базы данных хорошо обрабатывают OLTP-операции, а вот для обработки аналитики больше подходят, вы будете смеяться, колоночные базы данных — зато DML-операции для них проблема, ну вы поняли, почему. Промышленность, как вы знаете, пошла по пути строчных баз данных, на которые в виде компромисса навешиваются аналитические возможности.

Как это обычно бывает, у термина In-Memory DataBase нет устойчивого русского перевода, это что-то вроде «базы данных в оперативной памяти».

Мы все помним известное сокращение ACID, обозначающее требования к транзакционной системе: атомарность, согласованность, изолированность, надежность. Но в мире больших данных, где постоянно растут требования к объему и скорости обрабатываемых данных, часто жертвуют одним из этих требований, чтобы обеспечить необходимую производительность. In-memory базы данных – типичный пример такого подхода.

Базы данных in-memory хранятся в оперативной памяти сервера. Очевидно, что это обеспечивает большие преимущества в скорости, поскольку операции с данными в памяти выполняются меньшим числом инструкций процессора, а время доступа к данным операциями типа seek вообще несравнимо – оперативная память глобально выигрывает у жесткого диска.

В терминах ACID базы данных in-memory конечно жертвуют надежностью, ведь при обесточивании или перезагрузке в них пропадают все данные. Существуют ряд методов, которые позволяют снизить этот риск:

- **Снэпшоты** (снимки) базы данных в разные моменты времени – фактически эквивалентны бэкапу базы данных на жесткий диск. Недостатком метода является гарантированная потеря части «свежих данных» при каких-то проблемах.

- **Логи транзакций** записывают все транзакции на жесткий диск и позволяют восстановить данные после перезагрузки.

- **Использование NVRAM или NVDIMM** – модулей памяти, которые сохраняют свое состояние при пропадании питания, хотя конечно NVDIMM и технология Flash-памяти – это уже какое-то гибридное и безусловно более медленное решение.

Redis

Redis – быстрое хранилище в памяти с открытым исходным кодом для структур данных «ключ-значение». Redis поставляется с набором разнообразных структур данных в памяти, что упрощает создание различных специальных приложений.

Благодаря высокой скорости и простоте Redis часто используется для мобильных и интернет-приложений, игр, рекламных платформ, «Интернета вещей», т. е. в тех случаях, когда необходима максимально возможная производительность.

Несколько фактов о ключах:

- Ключи в Redis — бинарно-безопасные (binary safe) строки.

- Слишком длинные ключи — плохая идея, не только из-за занимаемой памяти, но так же и в связи с увеличением времени поиска определенного ключа в множестве в связи с дорогостоящим сравнением.

- Хорошая идея — придерживаться схемы при построении ключей: «object-type:id:field».

Типы данных Redis

- **Строки** (strings). Базовый тип данных Redis. Строки в Redis бинарно-безопасны, *могут использоваться так же как числа,* ограничены размером 512 Мб.

- **Списки** (lists). Классические списки строк, упорядоченные в порядке вставки, которая возможна как со стороны головы, так и со стороны хвоста списка. Максимальное количество элементов — 232 — 1.

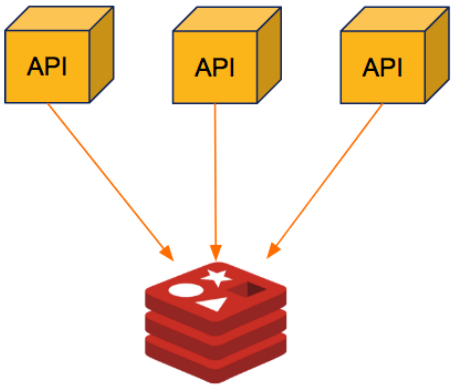
- **Множества** (sets). Множества строк в математическом понимании: не упорядочены, поддерживают операции вставки, проверки вхождения элемента, пересечения и разницы множеств. Максимальное количество элементов — 232 — 1.

- **Хеш-таблицы** (hashes). Классические хеш-таблицы или ассоциативные массивы. Максимальное количество пар «ключ-значение» — 232 — 1.

- **Упорядоченные множества** (sorted sets). Упорядоченное множество отличается от обычного тем, что его элементы упорядочены по особому параметру «score».

Топология Redis

Один Redis-инстанс



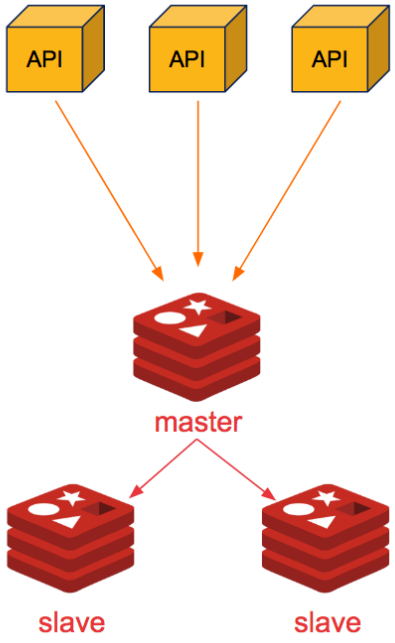
Самый классический случай.

Простой в запуске и настройке.

Ограничен ресурсами хоста, на котором запущен — его процессором и памятью.

В случае выхода из строя хоста или самого сервера Redis — разумеется, упадут все зависящие от него службы, т.е. никакого обеспечения надёжности нет.

Master-slave репликация



Один мастер, к которому подключены несколько слейвов, на которые выполняется полная репликация всех данных.

Данные обновляются на мастере, после чего он отправляет информацию об обновлениях на слейвы.

Слейвы общаются только с мастером (но могут иметь собственные слейвы).

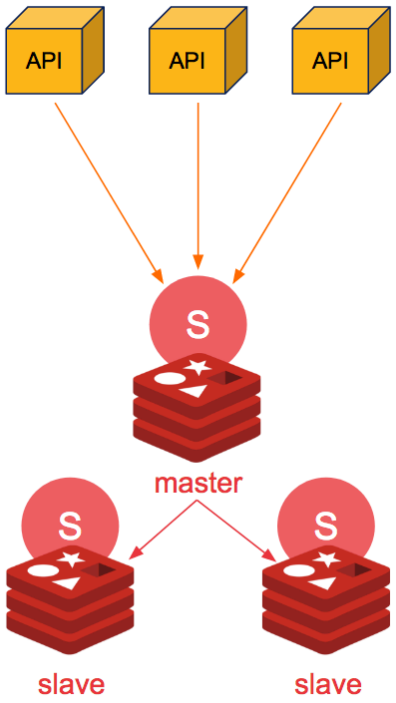
Слейвы являются read-only нодами — никаких изменений на них не выполняется (если явно не задано другое, см. во второй части).

В случае выхода из строя одной из нод — все данные останутся доступными с других улов.

Прост в настройке, но операции записи ограничены доступными мастеру ресурсами.

В случае выхода из строя мастера — требует ручного вмешательства для переопределения нового мастера из одного из слейвов, кроме того — клиенты должны точно знать на какую ноду выполнять запросы чтения и записи.

Redis Sentinel



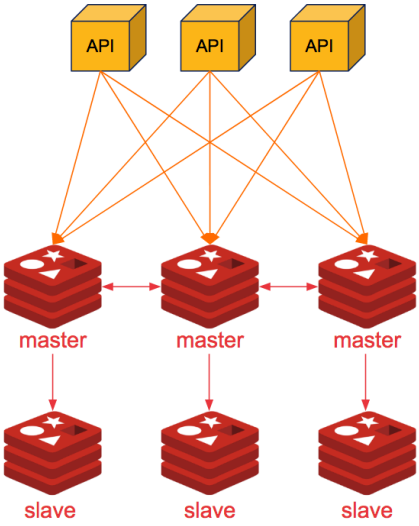
Уже описывался выше, но ещё несколько слов.

Как и в случае с репликацицей Redis — имеет одну Master Sentinel ноду, которая имеет приоритет при принятии решения о том, какую ноду Redis использовать в роли Master.

Т.е, в случае с одним Мастером Redis и двумя слейвами, и если Master Sentinel работает на том же хосте, где Мастер Redis-а, и этот хост умирает — Sentinel определяет свой новый мастер-инстанс, затем два оставшихся Sentinel-инстанса принимают решение о том, какой из оставшихся двух Redis-нод станет новым мастером. При этом у мастер-инстанса Sentinel будет приоритет при принятии решения.

Стоит учитывать, что не все клиенты умеют работать с моделью Redis Sentinel, список клиентов — тут>>>.

Redis Cluster



И самый полнофункциональный вариант — Redis Cluster.

Несколько мастер-инстансов, у каждого один или более слейвов (до 1000).

Выполняет все задачи по шардингу, репликации, failover, синхронизации данных.

Требует как минимум 6 нод Reedis-а: три для мастеров, и три для слейвов.

Умеет перенаправлять запросы от клиентов на нужный мастер или слейв — но это требует поддержки кластера самими клиентами Redis.

Какие есть преимущества:

- **Высочайшая производительность.** Все данные Redis находятся в оперативной памяти сервера, в отличие от большинства систем управления базами данных, в которых данные хранятся на жестких дисках или твердотельных накопителях (SSD). Размещаемые в памяти базы данных, такие как Redis, не требуют доступа к дисковым накопителям, что позволяет избежать потерь времени на поиск. Доступ к данным можно организовать при помощи более простых алгоритмов, в которых используется меньшее количество инструкций ЦПУ. Для выполнения типовых операций требуется менее миллисекунды.

- **Структуры данных в памяти.** Redis позволяет пользователям хранить ключи, привязанные к различным типам данных. Основной тип данных – это строка, которая может состоять из текстовых или двоичных данных размером до 512 МБ. Redis также поддерживает списки строк (List of Strings), упорядоченные в порядке вставки; множества неупорядоченных

строк (Sets of unordered Strings); упорядоченные по результату множества (Sorted Sets); хеш-таблицы (Hashes), содержащие список полей и значений, и тип данных HyperLogLogs для подсчета уникальных элементов в наборе данных. С помощью Redis в памяти можно хранить практически любые типы данных.

- **Универсальность и простота использования.** Redis поставляется с несколькими инструментами, ускоряющими и облегчающими разработку и эксплуатацию, включая шаблон «издатель-подписчик», для публикации сообщений в логических каналах, доставляемых подписчикам (хорошо подходит для чатов и систем обмена сообщениями); TTL: ключи могут иметь определенное время жизни (TTL), после чего они самостоятельно уничтожаются, это используется, чтобы избежать заполнения базы данных ненужными данными; атомарные счетчики, которые обеспечивают непротиворечивость результатов в ситуации состязания; а также Lua, мощный, но несложный язык скриптов.

- **Репликация и сохранность.** В Redis применяется архитектура «ведущий–подчиненный» и поддерживается асинхронная репликация, при которой данные могут реплицироваться на несколько подчиненных серверов. Это обеспечивает как улучшенные характеристики чтения (так как запросы могут быть распределены между серверами), так и восстановление при отключении основного сервера. Для обеспечения надежности Redis поддерживает как снимки состояния на момент времени (копирование наборов данных Redis на диски), так и создание файла только для добавления (AOF) для последовательной записи на диск всех изменений данных. Любой из этих методов обеспечивает быстрое восстановление данных Redis в случае сбоя.

- **Поддержка удобного языка разработки.** Для разработчиков под Redis доступны более ста клиентов с открытым исходным кодом. Поддерживаемые языки программирования включают Java, Python, PHP, C, C++, C#, JavaScript, Node.js, Ruby, R, Go и многие другие.

Примеры использования Redis:

- **Кэширование**. Если Redis поместить «перед» другой базой данных, создается высокопроизводительный кэш в памяти, который уменьшает задержку доступа, увеличивает пропускную способность и уменьшает нагрузку на реляционную базу данных или базу данных NoSQL.

- **Управление сессиями**. Redis отлично подходит для задач управления сессиями. Можно просто использовать Redis как быстрое хранилище пар «ключ-значение» с соответствующим временем жизни (TTL) ключей сессии для управления информацией сессии. Управление сессиями обычно требуется для интернет-приложений, включая игры, сайты электронной коммерции и платформы социальных сетей.

- **Таблицы лидеров в режиме реального времени.** С помощью структуры данных Redis Sorted Set элементы хранятся в списке, отсортированном по их результатам. Это позволяет легко создавать динамические таблицы лидеров, показывающие, кто побеждает в игре, или размещать наиболее понравившиеся сообщения, или использовать структуру в других случаях, когда необходимо показать, кто является лидером.

- **Ограничение интенсивности**. Redis может измерять и при необходимости ограничивать скорость наступления событий. Используя счетчик Redis, связанный с ключом API клиента, можно подсчитать количество запросов на доступ за определенный период времени и принять меры при превышении лимитов. Ограничители интенсивности обычно используются для установки лимита по количеству публикаций на форуме, лимитирования использования ресурсов и уменьшения влияния спамеров.

- **Очереди**. Структура данных список Redis позволяет легко создавать упрощенные постоянные очереди. Списки Redis List обеспечивают выполнение элементарных операций, а также возможности блокировки, поэтому они подходят для различных приложений, в которых требуется надежный брокер сообщений или циклический список.

- **Чат и обмен сообщениями.** Redis поддерживает стандарт «издатель-подписчик», а также сопоставления с шаблоном. Это позволяет использовать Redis для создания высокопроизводительных комнат чата, лент комментариев, работающих в режиме реального времени, и систем взаимодействия серверов. Стандарт «издатель-подписчик» также можно использовать для запуска действий на основе опубликованных событий.

Заключение

Резидентные СУБД отлично подходят для оперативной аналитики больших данных в режиме онлайн. Из-за их архитектурных особенностей, когда данные для распределенных вычислений хранятся в оперативной памяти кластерных узлов, а не считываются с жесткого диска, ключевыми преимуществами In-Memory СУБД считаются следующие:

- быстрота выполнения операций;

- эффективное сохранение зафиксированных данных, которые используются не часто, на жестком диске;

- высокая пропускная способность систем, критичных к производительности.

Обратной стороной этих достоинств являются следующие недостатки:

- однопоточность и эффективная утилизация только одного ядра ЦП, что не позволяет в полной мере воспользоваться возможностями современных многоядерных серверов. Поэтому при разработке решения на базе IMDB следует проектировать его с перспективой распараллеливания на десятки экземпляров (инстансов), что эффективно утилизировать hardware;

- энергозависимость и привязка к размеру оперативной памяти. Для решения этой проблемы используется запись с предварительным журналированием на энергонезависимом устройстве для обеспечения целостности базы данных при внезапной перезагрузке.

Поскольку существуют надежные и достаточно простые способы устранения указанных недостатков, неудивительно, что резидентные СУБД, такие как Tarantool, Arenadata Grid, Redis, Apache Ignite, InfinityDB, Memcached и прочие коммерческие и open-source продукты, очень востребованы в современных Big Data системах.