Цитаты из книги: «Kafka в действии»
Авторы: Дилан Скотт, Виктор Гамов, Дейв Клейн

О Kafka

Apache Kafka – это платформа потоковой передачи, которую можно использовать для быстрой обработки большого количества событий.

Ее можно вставить в середину рабочего процесса, и она будет служить единым интерфейсом к данным вместо многочисленных API и баз данных.

Линейное чтение и запись – вот основная сфера, где *Kafka* показывает себя с лучшей стороны и обеспечивает максимально быстрое перемещение данных.

Одна из ключевых особенностей *Kafka* – использование кеша страниц операционной системы. Избегая кеширования в куче JVM, брокеры могут предотвратить некоторые проблемы, свойственные большим кучам (например, длительные или частые паузы на сборку мусора).

Когда поступают новые сообщения, высока вероятность, что пришедшие последними представляют больший интерес для потребителей, и, соответственно, предпочтительнее обслуживать их из кеша. Обслуживание из кеша страниц, а не с диска в большинстве случаев происходит намного быстрее. В некоторых особых случаях увеличение объема ОЗУ помогает большей части рабочей нагрузки попасть в кеш страниц.

Брокер (broker)

Брокер – это серверный компонент *Kafka*. Быть брокером *Kafka* означает иметь возможность **координировать свои действия с другими брокерами**, а также взаимодействовать с ZooKeeper (теперь KRaft).

ISR (in-sync replicas) – синхронизированные реплики.

Только один брокер в кластере действует как контроллер. Роль контроллера заключается в управлении кластером. Контроллер также выполняет другие административные действия, такие как переназначение разделов.

Когда мы производим последовательное обновление кластера, останавливая и запуская по одному брокеру за раз, то обновлять контроллер лучше всего последним. Иначе может потребоваться остановить и запустить контроллер несколько раз.

Одна из особенностей *Kafka*, на которую следует обратить внимание, заключается в том, что по умолчанию реплики не пытаются восстановить свою работоспособность самостоятельно. Если брокер, хранящий одну из реплик раздела, потерпел сбой, то *Kafka* (в настоящее время) не создаст новую копию.

Любые разделы, созданные перед добавлением нового брокера, попрежнему будут храниться в брокерах, существовавших на момент их создания. Если новый брокер создавался только для обслуживания новых тем, то больше ничего делать не нужно.

Важным свойством конфигурации брокера для последовательного перезапуска является `controlled.shutdown.enable`. Установка его в значение `true` позволяет передать лидерство в разделе до остановки брокера.

Тема (topic)

Чтобы отправить сообщение, необходимо иметь место, куда оно будет отправлено. Таким местом в *Kafka* является **тема**.

Одна из копий раздела (реплика) будет называться ведущей, или лидером. Роль лидера предполагает получение обновлений от внешних клиентов, в то время как простые реплики получают обновления только от соответствующего лидера.

Темы состоят из блоков, называемых разделами. Другими словами, одна тема может состоять из одного или нескольких разделов. Что касается фактической реализации *Kafka*, то она по большей части работает именно с разделами.

Клиент также имеет возможность выбирать раздел для записи данных, настроив уникальный класс распределения по разделам.

Реплика с одним разделом существует только в одном брокере и не может разбиваться между брокерами.

Если вы стремитесь к надежности и хотите иметь три копии данных, то сама тема – это не одна сущность (или один файл), которая копируется; в действительности трижды копируются различные разделы.

Лидер будет одной из копий раздела, а остальные разделы будут подписчиками лидера и обновлять свою информацию, получая ее от своего лидера (ведущей реплики) раздела.

В New York Times, например, используется всего один раздел размером менее 100 Гб.

Проектирование темы – это двухэтапный процесс. На первом этапе определяется перечень событий и необходимость их распределения по разным темам. На втором этапе внимание сосредотачивается на каждой конкретной теме и выборе количества разделов для каждой из них. Важно помнить, что деление на разделы – это вопрос организации каждой темы, а не кластера. Мы можем установить количество разделов по умолчанию перед созданием темы, но в большинстве случаев необходимо учитывать, как будет использоваться тема и какие данные она будет хранить.

При выборе количества разделов для темы следует учесть, что в настоящее время не поддерживается возможность уменьшения их числа.

Установить параметр `auto.create.topics.enable` в значение `false`. Это гарантирует, что темы будут создаваться только явно, а не по желанию производителя.

В настоящее время данные в темах *Kafka* по умолчанию хранятся семь дней, однако мы легко можем настроить этот период по времени или размеру данных.

Сообщения

Доставка сообщений в *Kafka* может осуществляться как минимум тремя способами:

- не менее одного раза (at-least-once) – сообщение будет отправляется потребителям до тех пор, пока те не подтвердят его получение;
- не более одного раза (at-most-once) – сообщение отправляется только один раз и в случае сбоя не отправляется повторно;
- точно один раз (exactly-once) – потребитель гарантированно получит сообщение ровно один раз.

Если порядок сообщений имеет значение, то, помимо установки числа повторных попыток в параметре retries (например, 3), мы должны также установить параметр max.in.flight.requests.per.connection равным 1 и параметр acks (количество брокеров, которые возвращают подтверждение) равным all. На наш взгляд, это один из самых надежных способов гарантировать поступление сообщений от нашего производителя в требуемом порядке.

Клиенты

Производитель

Производитель – это компонент, отправляющий сообщения в темы *Kafka*.
Производителей по умолчанию не существует.

Поскольку производители могут передавать данные только в ведущую реплику того раздела, которому они назначены, метаданные помогают определить, какому брокеру послать сообщение, так как пользователь может указать только имя темы без любых других подробностей.

Потребитель

Потребитель – это инструмент для получения сообщений из *Kafka*.

Клиент-потребитель – это программа, которая подписывается на интересующую ее тему или несколько тем.

Ведущая реплика потребителя может отличаться от ведущей реплики (лидера) производителя из-за смены лидерства с течением времени; однако концептуально они схожи.

Тема определяет логическое имя того, что интересует потребителей, но читать сообщения они будут из ведущих реплик назначенных им разделов.

Но как потребители узнают к какому разделу подключаться? И не только раздел, но и ведущую реплику этого раздела? **Для каждой группы потребителей конкретный брокер принимает на себя роль координатора группы**. Клиент-потребитель

соединяется с этим координатором и получает от него назначенный раздел вместе с другими деталями, необходимыми для чтения сообщений.

Некоторые потребители не получат ничего, если потребителей больше, чем разделов, например когда имеется четыре потребителя и только три раздела.

Каждая ведущая реплика раздела обрабатывается только одним потребителем.

Один из ключевых недостатков конфигурации с большим количеством разделов является увеличение сквозной задержки в разделах. Если в вашем приложении счет идет на миллисекунды, то может случиться так, что при отправке сообщения вы не дождетесь в отведенный интервал времени, пока раздел будет скопирован между брокерами.

Потребители, не являющиеся частью одной и той же группы, не координируют информацию о смещении.

Важно решить, нужен ли новый идентификатор группы – работают ли потребители как части одного приложения, или это отдельные логические потоки.

Все потребители, использующие один тот же идентификатор group.id, будут считаться работающими вместе, совместно использующими разделы и смещения в теме и составляющими одно логическое приложение.

В некоторых системах потребители не оставляют на месте то, что они прочитали. Они извлекают сообщение, которое после подтверждения получения удаляется из очереди. Такая организация хорошо подходит для одиночных сообщений, что должны обрабатываться только одним приложением. Некоторые системы используют темы, в которых публикуют сообщения для всех подписчиков. И часто подписчики не получают опубликованных сообщений, потому что отсутствовали в списке получателей на момент публикации.

Ключевые координаты, позволяющие клиентам-потребителям работать вместе, представляют собой уникальное сочетание идентификатора группы, темы и номера раздела.

Как правило, только один потребитель в группе может читать один раздел. Другими словами, раздел может читаться многими потребителями, но только одним потребителем из каждой группы в каждый момент времени.

При сбое одного из потребителей его разделы, которые он читал, переназначаются другим потребителям в той же группе. Существующий потребитель начинает читать разделы, откуда читал потребитель, выпавший из группы.

heartbeat.interval.ms, определяющий интервал проверки связи с координатором группы.

Свойство partition.assignment.strategy определяет стратегию назначения разделов каждому потребителю. Доступны стратегии *Range*, *RoundRobin*, *Sticky* и *CooperativeSticky*.

В каких случаях следует использовать синхронную, а в каких – асинхронную фиксацию? Имейте в виду, что при синхронной фиксации задержка выше, так как приходится ждать завершения блокирующего вызова. Эта задержка может быть оправдана, если в число ваших требований входит необходимость обеспечения согласованности данных.

В *Kafka* нет поиска сообщений по ключам, зато есть поиск по определенному смещению

С точки зрения потребителя, каждый раздел – это неизменяемый журнал с сообщениями. Он должен только расти и добавлять сообщения в наше хранилище данных.

Кластеризация

Kafka обеспечивает отказоустойчивость и надежность, но что-то должно обеспечивать координацию, и этим «чем-то» является ZooKeeper (теперь KRaft).

Как вы уже видели, наш кластер для *Kafka* включает несколько брокеров (серверов). Чтобы брокеры действовали как одно согласованное приложение, они должны не только общаться друг с другом, но и достигать согласия. Согласование того, какой из них является ведущей репликой раздела – один из примеров практического применения ZooKeeper (теперь KRaft) в экосистеме *Kafka*.

Журнал

Хранением данных в журнале можно управлять по возрасту или размеру с помощью свойств конфигурации.

Поскольку *Kafka* хранит данные на диске, поддержка воспроизведения сообщений в случае сбоя приложения также является частью набора функций *Kafka*.

Получение данных и организация их доступности для пользователей – две большие и сложные задачи. Их можно решить, организовав хранение исходных данных и реализовав процедуры восстановление в случае сбоев.

Было бы полезно провести параллель между повторным чтением темы *Kafka* и идеей журнала упреждающей записи (*Write-Ahead Log*, *WAL*). Используя журнал *WAL*, можно сказать, какое значение имел тот или иной параметр раньше и как его значение менялось с течением времени, потому что изменения значений записываются в журнал до того, как они будут применены. Журналы *WAL* широко используются в системах баз данных и помогают им восстанавливаться, если в ходе выполнения транзакции произошел сбой. Проследив за событиями от начала до конца, можно увидеть, как меняются данные от начального значения к текущему.

Kafka создавалась с прицелом на высокую производительность, она использует файлы `.index` и `.timeindex` для хранения соответствий между смещением сообщения и физической позицией внутри индексного файла.

Другое

Термин serde означает serializer/deserializer (сериализатор/десериализатор) – класс, реализующий и сериализацию, и десериализацию. В практике, однако, интерфейсы сериализации и десериализации реализуются разными классами.

Как отключить ограничение времени хранения? Для этого достаточно установить параметры log.retention.bytes и log.retention.ms в значение -1.

Многие используют популярный термин, существующий в области преобразования данных – ETL (extract, transform, load – извлечение, преобразование, загрузка).

Сбор изменений в данных (Change Data Capture, CDC). Как следует из названия, можно выявлять изменения в данных и реагировать на эти изменения.

Каждое действие может масштабироваться независимо, не ограничивая другие действия. CQRS.

Перехватчики, определяемые нами, дают возможность добавить логику в производителя, потребителя или в оба клиента, которая включается в рабочий процесс, перехватывает события и добавляет дополнительные данные до того, как события продолжат перемещение по своему обычному пути. Изменения в клиентах зависят от конфигурации и помогают отделить нашу конкретную логику от клиентов.

Шифрование не означает, что другие не смогут увидеть ваши сообщения, шифрование лишь препятствует получению их содержимого в исходном виде.

SSL используется как обобщенное название, хотя в настоящее время существует более новая версия протокола – *TLS*.

Аутентификация – это процесс доказательства, что пользователь или приложение действительно является тем, за кого себя выдает.

Авторизация, с другой стороны, определяет круг привилегий пользователя – какие действия ему разрешено выполнять.

Одним из способов поддержки авторизации являются списки управления доступом (*Access Control List*, *ACL*).