Аннотация

аннотация

объем аннотации < 1500 символов, отразить цели и задачи работы, полученные результаты, рекомендации, предложенные на основании данной работы

Оглавление

Введение

Существует спрос на инструменты проверки согласованности транзакций, так как базы данных не обеспечивают тот уровень согласованности, на который претендуют. Почему? Чем они могут быть полезны?

- Пользователю(разработчику приложения, хранящего свои данные в базе данных) хочется научиться понимать про ту или иную базу данных, насколько она соответствует документации (это необходимо для того, чтобы выбрать базу, которая больше всего подходить для потребностей пользователя базы данных);
- Удобный инструмент для тестирования согласованности может существенно помочь на этапе разработки распределенных систем, возможно, стать одним из этапов CI/CD процесса;
- Jepsen проводит свои исследования независимо и в соответствии с их этической политикой. Это вносит большой вклад в сообщество, помогает в развитии и совершенствовании распределенных систем.

Большинство распределенных систем стремятся к достижению баланса между временем выполнения операций и согласованностью. Один из инструментов для проверки гарантии согласованности - это инструмент хаос-тестирования Jepsen. Хаос-тестирование — это тестирование путем внесения в систему незапланированных сбоев [5]. Наблюдая за поведением системы, можно понять, как сделать распределенную систему более надежной. Хаос тестирование это важная часть тестирования, потому что помогает выявить состояния гонки (race condition), которые сложно иначе обнаружить в процессе разработки.

2.1 Цели работы

- Получить опыт работы с выбранным инструментом проверки свойств транзакций (Jepsen);

- Изучить выполненные данным инструментом исследования различных баз данных;
- Исследовать с помощью выбранного инструмента реальную базу данных (Azure Cosmos DB), которая еще не была проанализирована;
- Сравнить уровень согласованности, заявленный в документации, и уровень, установленный с помощью тестов.

2.2 Основные понятия

Введем основные понятия, необходимые в дальнейшем.

Параллельная система — это система, состоящая из независимых компонент, которые могут выполнять некоторые операции одновременно.

Распределенная система — это тип параллельных систем, который представляет собой систему с несколькими независимыми компонентами, которые расположены на разных узлах в компьютерной сети. Эти узлы способны обмениваться данными, а также они координируют свои действия так, чтобы для конечного пользователя распределенная система работала как единая согласованная система. Система имеет логическое стостояние, которое меняется с течением времени.

Хаос-тестирование [5] — это тестирование путем внесения в распределенную систему незапланированных сбоев.

Процесс — это логически однопоточная программа, которая способна выполнять некоторые операции.

Операция — переход из одного состояние в другое.

Атомарная операция [6] — операция в общей области памяти, которая завершается за один шаг относительно других потоков, имеющих доступ к этой области памяти. Во время выполнения такой операции над переменной ни один поток не может наблюдать изменение наполовину завершенным. Неатомарные операции не дают такой гарантии.

Параллелизм — это свойство системы, которое означает, что несколько процессов могут выполняться в одно и то же время.

Сбой — это состояние процесса, в котором тот не может вызывать никаких операций.Если операция по какой-то причине не была завершена и не имеет времени завершения, то ее следует рассматривать одновременно с каждой операцией, которая будет вызвана после.Это нарушает требования на однопоточность процесса,поэтому мы говорим о состоянии сбоя процесса.

История — совокупность операций и их параллельной структуры. В этой работе мы будем рассматривать историю с точки зрения Jepsen. То есть история будет представлена в виде упорядоченного списка операций вызова и завершения.

Модель согласованности — набор гарантий, используемый в той или иной распределенной системе, для обеспечения согласованности данных.

Транзакция — некоторый конечный набор операций, переводящий данные из одного согласованного состояния в другое. Либо будет выполнена каждая операция из набора, либо ни одной. Основные свойства транзакций - атомарность, согласованность, изолированность и прочность (ACID).

2.3 План работы

В *Главе 2* будут рассмотрена основная информация о базах данных и о моделях согласованности для них. А также будут рассмотрены проблемы, которые могут возникать в процессе эксплуатации распределенных систем.

В *Главе 3* будет представлен анализ инструмента Jepsen для тестирования согласованности распределенных систем. Также будет проведен обзор исследований разных баз данных, выполненный с помощью Jepsen. Кроме того, будет рассмотрен инструмент Jepsen, который позволяет выявить аномалии в транзакционных историях (Elle).Этот инструмент для проверки транзакционных историй будет использован в дальнейшем в нашем эксперименте.

Глава 4 включит в себя подробное описание базы данных Azure Cosmos DB. Будет описан эксперимент,который был проведен над ней, чтобы выяснить,какие гарантии согласованности предоставлены и соответствуют ли они документации.

Синхронизация в базах данных

3.1 Базы данных

База данных это некоторое хранилище данных. Чаще всего в данной работе мы будем говорить о распределенных базах данных, то есть таких базах данных, составные части которых могут размещаться на различных узлах в компьютерной сети. Основное требование к базам данных - это поддерживание какого-то внутреннего консистентного состояния данных. Иными словами, согласованность и целостность, непротиворечивость данных в каждый момент времени. Различные базы данных добиваются этого разными способами. Реляционные базы данных предоставляют механизм транзакций, который гарантирует согласованность данных. Чтобы ускорить работу с данными, базы данных используют некоторый механизм блокировок, который теоретически гарантирует консистентное состояние.Однако, как мы увидим далее, это не всегда правда.

Некоторые базы данных реализуют различные модели согласованности,позволяющие регулировать гарантии, предоставляемые базой данных. Рассмотрим наиболее часто встречаемые модели согласованности, реализуемые базами данных.

3.2 Модели согласованности [1]

3.2.1 Отношения между моделями согласованности

3.2.2 Строгая сериализуемость (англ. *Strict Serializability*)

Строгая сериализуемость это модель, которая означает, что операции произошли в некотором порядке, совместимом с порядком этих операций в реальном времени. Например, если операция A завершается до начала операции B, это означает, что A предшествует B в порядке сериализации.

Итак, строгая сериализуемость гарантирует, что операции выполняются атомарно, то есть подоперации из одной транзакции не чередуются с подоперациями другой транзакции. Строгая сериализуемость не может гарантировать полной или частичной доступности. Это означает, что в случае разрыва в сети некоторые или все узлы в сети не смогут добиться прогресса. Строгая сериализуемость подразумевает сериализуемость и линеаризуемость. Вы можете думать о строгой сериализуемости как об общем порядке сериализуемости транзакционных многообъектных операций плюс ограничения линеаризуемости в реальном времени. В качестве альтернативы можно представить строгую сериализованную базу данных как ли-

неаризованный объект, в котором состояние объекта - это вся база данных. Ограничение линеаризуемости в реальном времени: транзакция A предшествует транзакции B, если A завершается до начала B.

3.2.3 Сериализуемость(англ. *Serializability*)

Сериализуемость - это транзакционная модель, в которой операции(транзакции) могут включать в себя несколько примитивных подопераций, выполняемых по порядку. Сериализуемость гарантирует, что операции выполняются атомарно: подоперации транзакции не чередуются с подоперациями из других транзакций.

Неформально сериализуемость означает, что транзакции произошли в некотором общем порядке. Это мульти объектное свойство: операции могут действовать на несколько объектов в системе, и к системе в целом. Сериализуемость не дает гарантии доступности. Сериализуемость подразумевает "повторяемое чтение "изоляцию моментальных снимков"и т. д. Однако она не накладывает никаких ограничений в реальном времени (здесь и далее это означает, что если процесс A завершает запись w, то процесс B начинает чтение r, не гарантируется, что r наблюдает w). Кроме того, процесс может наблюдать запись, а затем не наблюдать эту же запись в последующей транзакции. Фактически, процесс может не наблюдать свои собственные предыдущие записи, если эти записи происходили в разных транзакциях.

Более формально, сериализуемое выполнение определяется как одновременное выполнение операций SQL-транзакций, которое производит тот же эффект, что и некоторое последовательное выполнение тех же SQL-транзакций. Последовательное выполнение - это такое выполнение, при котором каждая SQL-транзакция выполняется полностью до начала следующей SQL-транзакции. Также можно говорить, что сериализуемость это read committed (чтение фиксированных данных), но с дополнительным ограничением на фантомное чтение. В другой формулировке сериализуемость определяется как отсутствие 4 явлений:

- грязной записи,
- грязного чтения,
- неповторяющееся чтение,
- фантомное чтение.

Сериализуемость это также комбинация трех свойств: Внутренняя согласованность (здесь и далее это означает, что в рамках транзакции чтения наблюдают за последними записями

этой транзакции (если таковые имеются)). Внешняя согласованность (здесь и далее: чтение без предшествующей записи в транзакции T1 должно учитывать состояние, записанное транзакцией T0, так что T0 виден T1, и никакая более поздняя транзакция не записана в этот объект). Полная видимость: отношение видимости должно быть полным порядком.

3.2.4 Повторяемое чтение(англ. *Repeatable Read*)

Эта модель согласованности тесно связана с сериализуемостью, но допускает фантомное чтение. Повторяемое чтение-это транзакционная модель(здесь и далее это означает, что операции (транзакции) могут включать в себя несколько примитивных подопераций, выполняемых по порядку). Это также мульти объектное свойство(здесь и далее это означает. что операции могут действовать на несколько объектов в системе). Нет гарантии полной доступности. Повторяемое чтение включает в себя стабильность курсора, чтение фиксированных данных и т.д. Также повторяемое чтение не накладывает никаких ограничений в реальном времени. Повторяемое чтение не требует упорядочивания процессов между транзакциями (здесь и далее это означает, что процесс может наблюдать запись, а затем не наблюдать эту же запись в последующей транзакции. Фактически, процесс может не наблюдать свои собственные предыдущие записи, если эти записи происходили в разных транзакциях). Можно определять повторяемое чтение как read committed (чтение фиксированных данных), но с дополнительным ограничением на неповторяющееся чтение. Иными словами, для повторяемого чтения запрещены: грязная запись, грязное чтение, неповторяющееся чтение, но допускается фантомное чтение.

3.2.5 Изоляция моментальных снимков(англ. *Snapshot Isolation*)

Изменения транзакции видны только этой транзакции до момента фиксации, когда все изменения становятся видимыми атомарно. Если транзакция T1 изменила объект x, а другая транзакция T2 совершила запись в x после начала моментального снимка T1 и до фиксации T1, то T1 должна прерваться. Изоляция моментальных снимков - это транзакционная модель. Это также мульти объектное свойство. Нет гарантии полной доступности. В отличие от сериализуемости, которая обеспечивает полный порядок транзакций, изоляция моментальных снимков вызывает только частичный порядок: подоперации в одной транзакции могут чередоваться с подоперациями из других транзакций. Наиболее заметными явлениями, допускаемыми изоляцией моментальных снимков, являются перекосы записи, которые позволяют транзакциям считывать перекрывающееся состояние, изменять непе-

ресекающиеся наборы объектов, а затем фиксировать; и аномалия транзакций только для чтения, включающая частично непересекающиеся наборы записи. Изоляция моментальных снимков подразумевает read commited (чтение фиксированны данных). Однако не накладывается никаких ограничений в реальном времени и не требует упорядочивания процессов между транзакциями.

В терминах абстрактного алгоритма можно говорить о данной модели согласованности так: каждая транзакция считывает данные из моментального снимка зафиксированных данных на момент начала транзакции, называемого ее меткой начала отсчета. Это время может быть в любое время до первого чтения транзакции. Транзакция никогда не блокируется при попытке чтения до тех пор, пока данные моментального снимка из его метки начала отсчета могут быть сохранены. Записи транзакции (обновления, вставки и удаления) также будут отражены в этом моментальном снимке, чтобы быть прочитанными снова, если транзакция обращается к данным во второй раз. Обновления другими транзакциями, активными после метки начала отсчета транзакции, невидимы для транзакции. Когда транзакция T1 готова к фиксации, она получает метку времени фиксации, которая больше любой существующей метки начала отсчета или другой метки времени фиксации. Транзакция успешно фиксируется только в том случае, если ни одна другая транзакция T2 с меткой времени фиксации в интервале выполнения T1 [время начала отсчета, время фиксации] не записала данные, которые также записал T1. В противном случае T1 прервется. Это предотвращает потерю обновлений. Когда T1 фиксирует, его изменения становятся видимыми для всех транзакций, метки начала отсчета которых больше, чем метка времени фиксации T1. В другой формулировке изоляции моментальных снимков определяется как комбинация четырех свойств:

- внутренняя согласованность
- внешняя согласованность
- Префикс: транзакции становятся видимыми для всех узлов в одном и том же порядке
- Отсутствие конфликта: если две транзакции записываются в один и тот же объект, одна должна быть видна другой.

3.2.6 Стабильность курсора(англ. *Cursor Stability*)

Стабильность курсора - это модель согласованности, которая усиливает read committed(чтение фиксированных данных), предотвращая потерю обновлений. Введем понятие курсора, кото-

рый относится к определенному объекту, доступ к которому осуществляется транзакцией. Транзакции могут иметь несколько курсоров. Когда транзакция считывает объект с помощью курсора, этот объект не может быть изменен какой-либо другой транзакцией до тех пор, пока курсор не будет отпущен или транзакция не будет зафиксирована. Это предотвращает потерю обновлений, когда транзакция T1 считывает, изменяет и записывает обратно объект x, но другая транзакция T2 также обновляет x после того, как T1 прочитал x, что приводит к фактической потере обновления T2. Стабильность курсора - это транзакционная модель. Это также мульти объектное свойство. Нет гарантии полной доступности. Стабильность курсора не требует упорядочивания процессов между транзакциями. Не накладывает никаких ограничений в реальном времени. Так как стабильность курсора это более строгое требование, чем read committed, также запрещены грязное чтение и грязная запись, но допустимо неповторяющееся чтение и фантомное чтение.

Формализация Адьи определяет этот уровень изоляции как два запрещенных явления: G-cursor и G1.

3.2.7 Монотонное атомарное представление(англ. *Monotonic Atomic View*)

Монотонное атомарное представление - это модель согласованности, которая усиливает read committed, препятствуя транзакциям наблюдать некоторые, но не все, эффекты ранее зафиксированной транзакции. Он выражает атомарное ограничение ACID, что все (или ни один) эффекты транзакции должны иметь место. Как только запись из транзакции T1 наблюдается транзакцией T2, то все эффекты T1 должны быть видны T2. Есть гарантия полной доступности. Монотонное атомарное представление не требует упорядочивания процессов между транзакциями. Не накладывает никаких ограничений в реальном времени. Монотонное атомарное представление сильнее, чем read committed, поэтому также запрещены грязное чтение и грязная запись, но допустимо неповторяющееся чтение и фантомное чтение. Как только некоторые эффекты транзакции Ti наблюдаются другой транзакцией Tj, после этого все эффекты Ti наблюдаются Tj. То есть, если транзакция Tj считывает версию объекта, записанную транзакцией Ti, то более позднее чтение Tj не может вернуть значение, более поздняя версия которого установлена Ti. С точки зрения формализации Адьи монотонное атомарное представление запрещает G1b (промежуточное чтение).

3.2.8 Чтение фиксированных данных(англ. *Read Committed*)

Модель согласованности, которая усиливает read uncommitted, предотвращая “грязное чтение” (транзакциям запрещено наблюдать за записями других транзакций, которые не фиксируются). Тем не менее, в процессе работы одной транзакции другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая транзакция будет работать с другим набором данных.

Это транзакционная модель: операции (обычно называемые «транзакциями») могут включать несколько примитивных подопераций, выполняемых по порядку. Это также multi-object свойство: операции могут действовать на несколько объектов в системе.

Read Committed не требует отдельного процесса между транзакциями. Процесс может наблюдать запись, а затем не наблюдать эту же запись в последующей транзакции. Фактически, процесс может не отслеживать свои предыдущие записи, если эти записи происходили в разных транзакциях. Реализация Read Committed может основываться на одном из двух подходов: блокировании или версионности.

3.2.9 Чтение незафиксированных данных(англ. *Read Uncommitted*)

Модель согласованности, которая запрещает “грязную” запись, когда две транзакции одновременно изменяют один и тот же объект перед фиксацией. Зато позволяет другие аномалии (грязное чтение, фантомное чтение, неповторяющееся чтение).

Это транзакционная модель: операции (обычно называемые «транзакциями») могут включать несколько примитивных под операций, выполняемых по порядку. Это также multi-object свойство: операции могут действовать на несколько объектов в системе.

Модель гарантирует отсутствие потерянных обновлений. Если несколько параллельных транзакций пытаются изменять одну и ту же строку таблицы, то в окончательном варианте строка будет иметь значение, определенное всем набором успешно выполненных транзакций. При этом возможно считывание не только логически несогласованных данных, но и данных, изменения которых еще не зафиксированы. Типичный способ реализации данного уровня изоляции — блокировка данных на время выполнения команды изменения, что гарантирует, что команды изменения одних и тех же строк, запущенные параллельно, фактически выполняются последовательно, и ни одно из изменений не потеряется. Транзакции, выполняющие только чтение, при данном уровне изоляции никогда не блокируются.

3.2.10 Линеаризуемость(англ. *Linearizability*)

Линеаризуемость - это одна из самых сильных моделей однообъектной согласованности и подразумевает, что каждая операция выполняется атомарно, в некотором порядке, совместимом с порядком этих операций в реальном времени. Нет гарантии какой-либо доступности. Линеаризуемость-это однообъектная модель, но объем “объекта” варьируется. Некоторые системы обеспечивают линеаризуемость отдельных ключей в хранилище ключ-значение; другие могут обеспечивать линеаризуемость операций над несколькими ключами в таблице или несколькими таблицами в базе данных, но не между различными таблицами или базами данных соответственно. Более формально, история исполнения H такова, что существует эквивалентная последовательная история S, а частичный порядок операций в реальном времени в H согласуется с общим порядком S и сохраняет однопоточную семантику объектов. Иными словами, должны выполняться сразу три свойства:
1) единый порядок (существует некоторый общий порядок операций);
2) привязка к реальному времени;
3) подчинение однопоточным законам типа данных связанного объекта.

3.2.11 Последовательная согласованность(англ. *Sequential Consistency*)

Это сильное свойство безопасности для параллельных систем. Неформально последовательная согласованность подразумевает, что операции происходят в некотором общем порядке и что этот порядок согласуется с порядком операций на каждом отдельном процессе. Не гарантируется полная или частичная доступность. Процесс в последовательно согласованной системе может быть далеко впереди или позади других процессов. Например, может быть прочитано устаревшее состояние. Однако, как только процесс A наблюдает некоторую операцию из процесса B, он никогда не может наблюдать состояние, предшествующее B. Можно представить последовательную согласованность как три свойства: единый порядок (существует некоторый общий порядок операций), PRAM и порядок должен соответствовать семантике типа данных. Более формально, когда мы говорим о последовательной согласованности, считается, что результат любого исполнения такой же, как если бы операции всех процессоров выполнялись в некотором последовательном порядке, и операции каждого отдельного процессора отображаются в этой последовательности в порядке, определенном его программой.

3.2.12 Причинная согласованность(англ. *Causal Consistency*)

Это модель согласованности говорит о том, что причинно-связанные операции должны появляться в одном и том же порядке во всех процессах, хотя порядок может меняться для причинно-независимых операций. Гарантируется частичная доступность.

Причинная память проистекает из определения отношения "происходит до"(happens-before), которое фиксирует понятие потенциальной причинности, связывая операцию с предыдущими операциями того же процесса и с операциями над другими процессами, последствия которых могли быть видны благодаря сообщениям, которыми обменивались эти процессы.
Happens before
Пусть есть поток X и поток Y (не обязательно отличающийся от потока X). И пусть есть операции A (выполняющаяся в потоке X) и B (выполняющаяся в потоке Y).

B таком случае, A happens-before B означает, что все изменения, выполненные потоком X до момента операции A и изменения, которые повлекла эта операция, видны потоку Y в момент выполнения операции B и после выполнения этой операции.

3.2.13 Записи следуют за чтениями(англ. *Writes Follow Reads*)

Данная модель согласованности гарантирует, что если процесс считывает значение v, которое пришло из записи w1, а затем выполняет запись w2, то запись w2 должна быть видна после w1. Как только вы что-то прочитали, вы не можете изменить прошлое этого чтения. Гарантируется полная доступность.

3.2.14 Конвейерная Оперативная Память(англ. *PRAM, Pipeline Random Access Memory*)

PRAM пытается ослабить существующие когерентные модели памяти, чтобы получить лучший параллелизм (и, следовательно, производительность). Эта модель гарантирует, что любая пара записей, выполненных одним процессом, наблюдалась везде в том порядке, в котором процесс их выполнил; однако записи из разных процессов могут наблюдаться в разных порядках.

PRAM эквивалентен монотонному чтению, монотонной записи и чтению своих записей (смотреть далее). Гарантируется частичная доступность. Более формально, ПРАМ удовлетворяется, если порядок сеанса (порядок операций над каждым процессом) является подмножеством порядка видимости (какие операции видны данной операции).

3.2.15 Монотонные чтения(англ. *Monotonic Reads*)

Эта модель согласованности гарантирует, что если процесс выполняет чтение r1, а затем r2, то r2 не может наблюдать состояние до записи, которая была отражена в r1. Не применяется к операциям, выполняемым различными процессами, а только к операциям, выполняемым одним и тем же процессом. Монотонное чтение может быть полностью доступно (все узлы могут совершать прогресс даже во время разрыва в сети)

Более формально, можно говорить о монотонном чтении в терминах порядка сеанса (порядка операций, выполняемых одним и тем же процессом) и порядка видимости (какие записи видны для каких чтений). Для всех операций a, b и c, где b и c являются чтением, если a виден b, и b выполняется до и тем же процессом, что и c, то a должен быть виден c.

3.2.16 Монотонные записи(англ. *Monotonic Writes*)

Эта модель согласованности гарантирует, что если процесс выполняет запись w1, а затем w2, то все процессы наблюдают w1 до w2. Не применяется к операциям, выполняемым различными процессами, а только к операциям, выполняемым одним и тем же процессом. Есть полная доступность: даже во время разрыва сети все узлы могут совершать прогресс.

3.2.17 Чтение своих записей(англ. *Read Your Writes*)

Этот уровень изоляции транзакций требует: если процесс выполняет запись w, то этот же процесс выполняет последующее чтение r, и тогда r должен наблюдать эффекты w. Важно, что это модель не применяется к операциям, выполняемым различными процессами. Например, нет никакой гарантии, что если процесс 1 успешно запишет значение, то процесс 2 впоследствии будет наблюдать эту запись.

Есть гарантия частичной доступности: если возникает разрыв сети, каждый узел может добиться прогресса (если клиент никогда не меняет сервер, с которым он взаимодействует). Более формально: для любой записи и любого чтения, если запись происходит прямо перед чтением в данном сеансе (процессе), то запись должна быть видна для чтения. Другими словами, порядок сеанса (ограниченный только записью -> чтением) является подмножеством порядка видимости.

3.3 Нарушения согласованности

... какие проблемы бывают с согласованностью (только те, которые были упомянуты в секции выше) DSG(H)
ребра анти зависимости (anti-dependency edges)

3.3.1 грязная запись

3.3.2 грязное чтение

3.3.3 неповторяющееся чтение

3.3.4 фантомное чтение

3.3.5 G-cursor

G-cursor(x): граф направленной сериализации, ограниченный одним объектом x, содержит анти-зависимый цикл и по крайней мере одно ребро, зависимое от записи.

3.3.6 G1

G1: включает в себя три запряженных явления:
G1a (прерванное чтение): транзакция наблюдает объект, измененный прерванной транзакцией
G1b (промежуточное чтение): транзакция наблюдает объект, измененный транзакцией, которая не была окончательной модификацией этого объекта этой транзакцией. Иными словами, транзакции должны завершиться, прежде чем мы сможем их прочитать,
G1c (циклический информационный поток): направленный граф сериализации транзакций содержит направленный цикл, полностью состоящий из ребер зависимостей. Иными словами, что если транзакция T1 подвержена влиянию T2, то T2 не может быть подвержена влиянию T1.

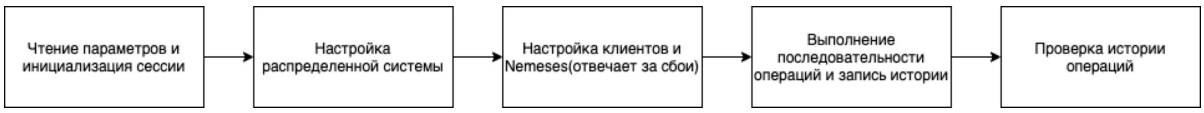
3.3.7 G-item

аномалия, которую нашли тесты

Методология проверки согласованности распределенных систем

Итак, в предудущей главе мы рассмотрели некоторые нарушения и аномалии,которые нарушают гарантии согласованности распределенной системы.Несмотря на то, что они встречаются довольно часто,их сложно обнаружить. В этой главе мы рассмотрим инструмент,разработанный специально для проверки того, соответствует ли распределенная система своим гарантиям согласованности.

4.1 Jepsen



Jepsen это библиотека для функционального языка Clojure.

Jepsen проверяет систему, генерируя случайные последовательности операции (например, чтение, запись, cas) в распределенной системе, записывая метку времени и продолжительность каждой операции, а также создавая модель системы в памяти. А затем он пытается доказать, имеет ли история событий смысл с учетом модели. Jepsen также может генерировать множество сбоев в распределенной системе, например, проблемы с сетью, уничтожение компонентов, а также генерацию случайной нагрузки.

При запуске теста jepsen сначала подключается по ssh к каждому узлу, загрузит, распакует и настроит на них распределенную систему. После запустятся клиентские и nemesis процессы. Во время теста в jepsen есть два типа процессов: один - клиент, который будет выполнять различные операции с системой (с интервалом и частотой, заданных в генераторе), а другой - nemesis, который будет вносить сбои и разрушения и выполнять восстановление системы. После завершения операций jepsen будет использовать checker (после jepsen 2.0

это будет elle), чтобы проверить правильность истории операций с определенными моделями согласованности.

4.1.1 Jepsen and docker

В данном исследовании мы будем запускать кластер Jepsen на одном компьютере, используя docker compose. Это упрощает и стандартизирует выполнение тестов. Репозиторий Jepsen предоставляет базовые настройки для запуска тестов в докере. Он поддерживает 3 вида контейнеров:

jepsen-control: управляет другими узлами, настройкой и удалением, генерирует данные и сбои;

jepsen-nX: один из узлов в кластере(по умолчанию таких узлов 5);

jepsen-node: используется агентом для создания сбоев (nemesis).

4.1.2 Возможные сбои

сетевые разделы, изолирующие отдельные узлы; а также разделяющие кластер на мажоритарные и миноритарные компоненты приостановка или разрушение случайного подмножества узлов сбой часов

4.1.3 Как анализировать результаты

При каждом запуске jepsen создает новый каталог в store/ директории, и мы можем увидеть последние результаты в папке store/latest. Там лежат несколько файлов. history.txt содержит операции, которые выполнял тест. jepsen.log копия консоли для этого запуска, jepsen.log - журнал всех операций, выполненных jepsen к тестируемой системе, и наконец test.fressian это необработанные данные теста, включающие полную историю операций, timeline.html - это html документ, который показывает удобную временную шкалу операций. Эта шкала очень полезный инструмент для понимания порядка операций в тесте и выявления причин несогласованности результатов теста. Синий цвет указывает на то, что операция прошла успешно, красный - на неудачную операцию (состояние системы не изменилось), а оранжевый - на неопределенную операцию.

4.2 Elle

Elle — это инструмент для анализа транзакций. Он способен автоматически выводить график логического порядка транзакций и искать циклы в этом графике для выявления нарушений согласованности. Например, если транзакции образуют логический цикл, то невозможно сказать, какая транзакция произошла до и после, а значит, нарушается гарантия линеаризуемости. Дополнительно проверяется наличие прерванных и промежуточных считываний и другие нарушения.

Elle не является полным: он может не идентифицировать аномалии, которые присутствовали в тестируемой системе. Это следствие двух факторов:

- Elle проверяет истории, наблюдаемые в реальных базах данных, где результаты транзакций могут остаться незамеченными, а информация о времени может быть не такой точной, как хотелось бы.
- Проверка сериализуемости является NP-полной; Elle намеренно ограничивает свои выводы теми, которые можно решить за линейное (или лог-линейное) время.

Список возможных аномалий:

- G0 (грязная запись),
- G1a (прерванное чтение),
- G1b (промежуточное чтение),
- G1c (cyclic information flow, циклический поток информации),
- G-single (read skew, перекос чтения) ,
- G2-item (anti-dependency cycle, цикл защиты от зависимостей).

Инструмент также умеет проверять согласованность внутри одной транзакции: то есть можно проверить, что транзакции считывают значения, соответствующие их собственным предыдущим записям, нет дублирующихся элементов и неожиданных элементов (например, элементов, которые никогда не были записаны).

4.3 Обзор исследований, выполненных Jepsen

Jepsen использовался для тестирования многих распределенных систем. Я ознакомилась с результатами некоторых из них.

4.3.1 Etcd 3.4.3 [2] [3]

При изучении jepsen инструмента и его возможностей мы повторим эксперимент из туториала для тестирования ETCD(распределенное хранилище пар ключ-значение).

Дизайн тестов

Для тестирования etcd командой jepsen были разработаны следующие тесты:

register тест: тест выполняет случайные операции чтения, записи и сравнения (compare-and-set) по отдельным ключам. Оценка этих истории операций с помощью средства проверки линеаризуемости Knossos.

set тест: для того, чтобы обнаружить чтение устаревший данных, был разработан тест, который использовал cas операцию для считывания набора целых чисел по одному ключу и добавление значения к этому набору. Мы одновременно читаем множество значений на протяжении всего теста. В конце теста ищем случаи, когда элемент, который теоретически должен быть в наборе, не появляется при чтении. Эти случаи использовались для количественного измерения устаревших чтений и потерянных обновлений.

append тест: для проверки строгой сериализуемости был разработан тест, в котором транзакции параллельно читают и добавляют в списки уникальные целые числа. Каждый список храниться по уникальному ключу. Мы выполняем append каждой транзакции, читая каждый ключ, который должен быть изменен, в одной транзакции, затем пишем эти ключи и выполняем чтения во второй транзакции, что гарантирует что никакие записанные ключи не изменились с первого чтения. В конце строится график зависимостей между транзакциями на основе приоритета в реальном времени и отношений между чтениями и добавлениями, ищутся циклы в этом графике.

lock тест: etcd предоставляет возможность блокировки. Тест оценивает надежность функционала блокировки двумя способами: 1) генерация случайных запросов на блокировку и разблокировку. После история операций проверяется с помощью Knossos, чтобы увидеть, являются ли блокировки линеаризуемыми; 2) для получения количественного представления о том, как часто блокировки могут отказывать, тест использует блокировку для обеспечения взаимного исключения обновления набора значений (те обновление в критической секции) и искали (lost updates) потерянные обновления этого набора.

watch тест: чтобы убедиться, что каждое обновление ключей происходит в правильном порядке, тест создает один ключ и вслепую устанавливать для него уникальные целочисленные значения в течении теста. Тем временем клиенты параллельно наблюдают за этим ключом в течении нескольких секунд за раз. И каждый раз, когда клиент запускал наблюдение, оно

начиналось с того места, где в последний раз закончилось. В конце теста было проверено, что каждый клиент наблюдал точно такую же последовательность обновлений ключа.

Результаты тестирования

В документации etcd утверждает, что операции сериализуемые и линеаризуемые. И etcd действительно соответствует данным требованиям, даже если в процесс были добавлены паузы, сбои, сдвиги часов и сетевые разделы. А при выполнении чтения с установленным флагом serializable возможно устаревшее чтение, как и предполагает документация. Также было выявлено, что блокировки небезопасны: несколько клиентов могут удерживать одновременно одну и ту же блокировку. Хотя стоит заметить, что Jepsen может доказать только лишь наличие ошибок, а не их отсутствие.

Для того, чтобы поближе познакомиться с Jepsen и Clojure и в дальнейшем понять, как выполнить с помощью Jepsen свое собственное исследование, мы, следуя туториалу,реализовали set и register тест.

Для того, чтобы развернуть jepsen кластер, мы использовали docker compose. Основные используемые библиотеки - Jepsen и Verschlimmbesserung (библиотека для взаимодействия с etcd). **Этапы подготовки тестов**

- настройка автоматической установки тестируемой системы на рабочие узлы (В Jepsen это делается путем указания узлу управления(control node) загружать двоичные файлы с URL-адреса с помощью встроенной функции Jepsen install-archive);
- написание клиента для взаимодействия с распределенной системой (как к ней подключиться, как выполнять операции над системой).
- добавление сбоев в систему с помощью специального клиента jepsen.nemesis
- анализ истории операций с помощью модели

В результате запуска этих тестов никаких аномалий не было обнаружено.

4.3.2 MongoDB 4.2.6

??? пока не уверена, стоит ли описывать результаты проверки

4.3.3 PostgreSQL 12.3

??? пока не уверена, стоит ли описывать результаты проверки

Исследование согласованности Azure Cosmos DB

5.1 Azure Cosmos DB

информация о базе данных

5.2 Уровни согласованности [4]

Одной из особенностей Azure Cosmos DB является то, что она позволяет выбрать один из 5 уровней согласованности, который будет удовлетворять нужным критериям.

5.2.1 Строгая (англ. *strong*)

Данные синхронно реплицируются на все реплики в режиме реального времени. Гарантирует линеаризацию. Это означает, что порядок операций сохраняется, и считывания гарантированно возвращают самую последнюю версию элемента в базе данных. Клиент всегда получает последние изменения в данных по запросу. Никогда не будет видима незафиксированная или частично измененная запись. Самая низкая производительность и доступность.

5.2.2 Ограниченное устаревание(англ. *bounded staleness*)

Данные реплицируются асинхронно с заданным окном устаревания, определяемым либо количеством записей, либо периодом времени. Запрос на чтение может отставать либо на определенное количество операций записи, либо на заранее определенный период времени. Однако при чтении гарантируется соблюдение последовательности данных. По мере приближения окна устаревания репликация данных запускается в учетной записи базы данных, заставляющей базу данных обновлять новые записи с момента последнего изменения.

Низкая доступность из-за задержки синхронизации разных регионов. Более хорошая производительность.

5.2.3 Сеанс(англ. *session*)

Это уровень согласованности по умолчанию. Это обеспечивает сильную согласованность для сеанса приложения с одним и тем же токеном сеанса. Это означает, что все, что написано сеансом, также вернет последнюю версию для чтения из того же сеанса. Доступность данных относительно высока и более низкой задержкой и более высокой пропускной способностью, чем bounded staleness. Данные из других сеансов поступают в правильном порядке, просто не гарантируется, что они будут актуальными.

5.2.4 Постоянный префикс(англ. *consistent prefix*)

Эта модель согласованности аналогична bounded staleness, только без гарантии задержки. Реплики гарантируют согласованность и порядок записи, однако данные не всегда актуальны. Эта модель гарантирует, что пользователь никогда не увидит неупорядоченную запись. Высокая доступность и низкая задержка.

5.2.5 Случайная(англ. *eventual*)

Для операции чтения нет гарантии порядка данных, а также отсутствует гарантия того, сколько времени может потребоваться для репликации данных. Это слабая форма согласованности, так как могут быть считаны более старые значения чем те, которые были считаны раньше. Высокая доступность, самая высокая пропускная способность и низкая задержка.

5.3 Дизайн теста

Мы разработали тест с использованием библиотеки для тестирования распределенных систем Jepsen. Будем использовать его для оценки безопасности транзакций в Azure Cosmos DB.

Append тест

Генератор транзакций Jepsen генерировал транзакции, состоящие из последовательности операций. Длину последовательности операций можно настраивать, для запуска тестов использовалась длина 4. Каждая операция в последовательности — это либо чтение массива

значений по ключу id, либо добавление уникального целого числа в массив значений по ключу id. Далее, используя Elle для анализа транзакций, Jepsen выводил зависимости между транзакциями и искал циклы для выявления аномалий.

5.4 О реализации транзакций в Azure Cosmos DB

5.4.1 TransactionalBatch

??? РЕДАКТИРОВАТЬ TransactionalBatch (увеличение производительности; пакет транзакций описывает группу операций, которые должны либо успешно выполниться, либо завершиться сбоем с одним и тем же ключом секции в контейнере. Если все операции выполняются успешно в том порядке, в котором они описаны в транзакционной пакетной операции, транзакция будет зафиксирована. Однако при сбое любой операции выполняется откат всей транзакции.)

При тестировании транзакций с использованием TransactionalBatch Elle обнаружила следующие аномалии:

- внутренняя несогласованность (англ. *Internal Inconsistency*) — транзакция не соблюдает свои собственные предыдущие операции чтения и записи.
- несогласованный порядок версий (англ.*Inconsistent Version Orders*) — правила вывода предполагают циклический порядок обновления одного ключа.

Также, из отношений между аномалиями Elle[7] можно заключить, что из обнаружение в истории *nconsistent Version Orders* аномалии следует, что G1a аномалия там также присустствует.

Кроме того, на уровнях **Ограниченное устаревания** и **Случайная** была обнаружена *G-item* аномалия. Напомним, что обанаружение этой аномалии означает, что DSG(H) содержит направленный цикл с одним или несколькими ребрами анти зависимости [8].

Транзакции теряют подтвержденные записи. Кроме того, оказывается, что транзакции не изолированы. То есть,транзакции в такой реализации могли влиять на результаты других транзакций. А значит, TransactionalBatch в нашей задаче использовать нельзя.

5.4.2 Хранимые процедуры(англ. *Stored procedures*)

Azure Cosmos DB обеспечивает транзакционное выполнение JavaScript кода. При использовании API SQL в Azure Cosmos DB можно писать хранимые процедуры, триггеры и

определяемые пользователем функции (UDF) на языке JavaScript. Помимо того, что написанный JavaScript код будет выполняться атомарно, также данный способ реализации транзакций обещает хорошую производительность. Можно назвать следующие преимущества:

- *пакетная обработка* — это сократит затраты сетефого трафика и накладные расходы на хранение
- *предварительная компиляция* — хранимые процедуры, триггеры и определяемые пользователем функции неявно предварительно скомпилированы в формат байтового кода, чтобы избежать затрат на компиляцию во время каждого вызова скрипта. Благодаря предварительной компиляции скорость хранимой процедуры высокая, а занимаемая память небольшая.

Согласованность данных

Хранимые процедуры и триггеры всегда выполняются на основной реплике контейнера Azure Cosmos. Эта возможность гарантирует, что операции чтения в хранимых процедурах обеспечивают сильную согласованность.

Заключение

Выводы

Литература

[1] Consistency models. "https://jepsen.io/consistency".

[2] Kyle Kingsbury. etcd 3.4.3. "https://jepsen.io/analyses/etcd-3.4.3".

[3] Jepsen tutorial. "https://github.com/jepsen-io/jepsen/tree/main/doc/tutorial".

[4] Consistency levels in azure cosmos db. "https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels".

[5] Serdar Benderli. Chaos testing a distributed system with jepsen. "https://medium.com/appian-engineering/chaos-testing-a-distributed-system-with-jepsen-2ae4a8bdf4e5".

[6] Атомарные и неатомарные операции. "https://habr.com/ru/post/244881/".

[7] Kyle Kingsbury and P. Alvaro. Elle: Inferring isolation anomalies from experimental observations. *Proc. VLDB Endow.*, 14:268–280, 2020.

[8] Atul Adya, Barbara Liskov, and Patrick O'Neil. Generalized isolation level definitions. pages 67–78, 01 2000.