Apache Ignite Transactions Architecture

Версия 0.4
19.12.2017

Данный документ посвящен текущей реализации механизма транзакций в Apache Ignite.

Мы:

1. Очень кратко вспомним основные моменты архитектуры Ignite
2. Поговорим о протоколе 2х фазной фиксации транзакций
3. Узнаем какие режимы блокировки поддерживает Ignite
4. Рассмотрим как выполняется транзакция
5. Как обрабатываются ситуации, когда "теряются" узлы на разных этапах транзакции

Для начала рассмотрим определение и основные свойства Ignite.

1. Introduction to Apache Ignite

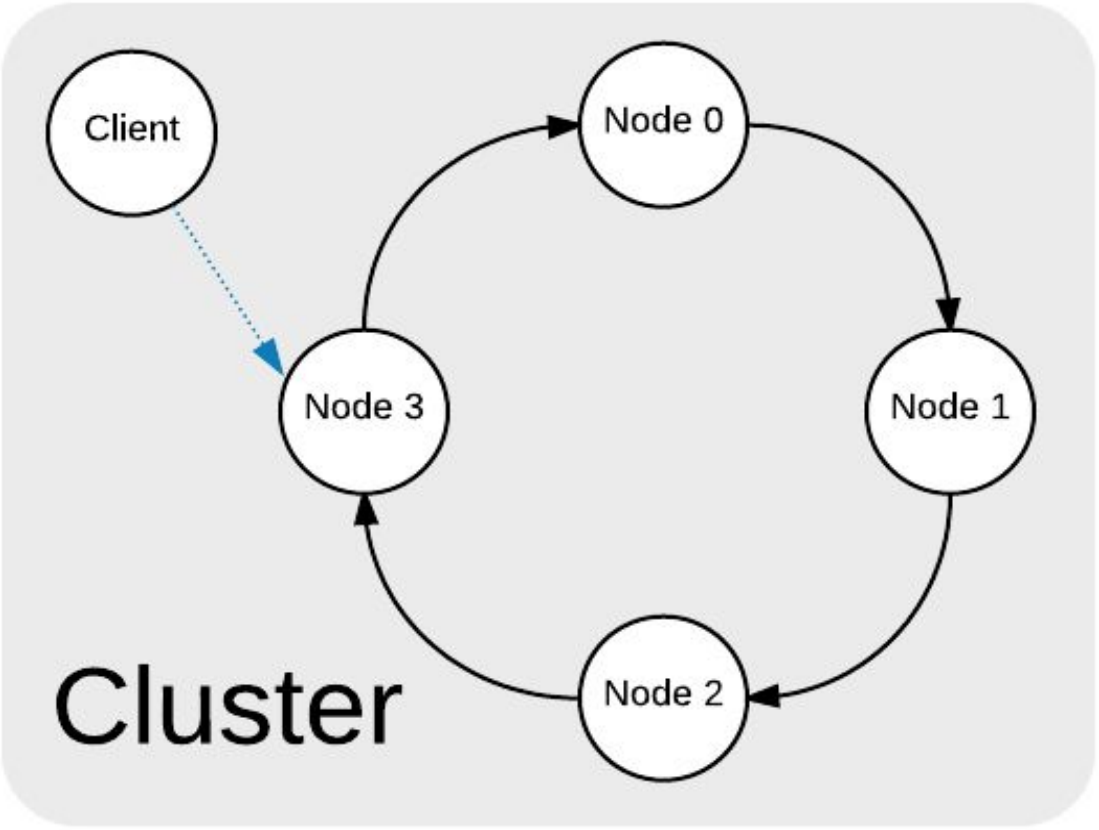
1.1. Apache Ignite

Актуальным для Ignite на текущий момент является следующее определение: "Ignite - это распределенная, memory-centric СУБД".
Как распределенная система, она получила пары ключ-значение для хранения данных. Все алгоритмы ее разрабатывались и оптимизировались под параллельные вычисления. Как СУБД она позволяет переживать переживать перезагрузку, и предоставляет поддержку распределенных индексов и SQL запросов по всему набору данных в оперативной памяти и на диске.

Более полно Ignite, это система с возможностью:

1. создавать эластичный, горизонтально масштабируемый Data Grid с масштабированием на тысячи узлов, с распределением данных через аффинити функцию для уменьшения избыточного перемещения данных при добавлении или удалении узлов.
2. использовать созданный Data Grid для выполнения параллельных вычислений в Compute Grid. Причем код выполняется непосредственно на узлах, где содержатся необходимые данные.
3. использовать Durable Memory для хранения данных не только в памяти, но и на диске.
4. выполнять распределенные SQL ANSI-99 запросы по всему набору данных, находящихся как в памяти, так и на диске любого из узлов.
5. строить, обучать, тестировать Machine Learning алгоритмы и модели, специально оптимизированные для работы на совместно распределенных данных.

1.2. Cluster/Cache/Partitions



Кластер Ignite представляет собой множество объединенных в кольцо серверных и, подключенных к ним, клиентских узлов.
И серверные и клиентские узлы могут выполнять абсолютно все задачи Ignite API, стартовать транзакции, участвовать в вычислениях и загрузках.
Основное отличие между ними состоит в том, что на клиентских узлах не хранятся данные и они требуют наличия серверного узла для подключения к кластеру.

Для пользователя основная ценность кластера заключается в возможности хранить данные и выполнять на них вычисления.
Данные хранятся в виде пар ключ - значение, а способом объединения таких пар с общими настройками выступают кеши, которые объединяются в группу кешей.

При хранении в кеше большого объема данных, он разделяется на несколько частей партиций - partition.

С целью обеспечения сохранности данных, кроме основной (Primary) копии создаются ее резервные (Backup) копии.
Каждый узел в кластере (и серверный, и клиентский) "знает" о размещении всех Primary и Backup копий партиции. Эта информация собирается координатором кластера и рассылается в Partition Map Exchange сообщении всем узлам.
Для пользователя вся работа с кешем (и put, и get операции) выполняется только через Primary partition (за исключением случаев, когда явно указано выполнение операции чтения с Backup partition).
Если Primary раздел не доступен (например узел перегружается), транзакция должна дождаться смены топологии (т.к. один из узлов вышел из обслуживания) и определения нового узла с Primary партицией. После этого транзакция попробует выполнить операцию еще один раз.

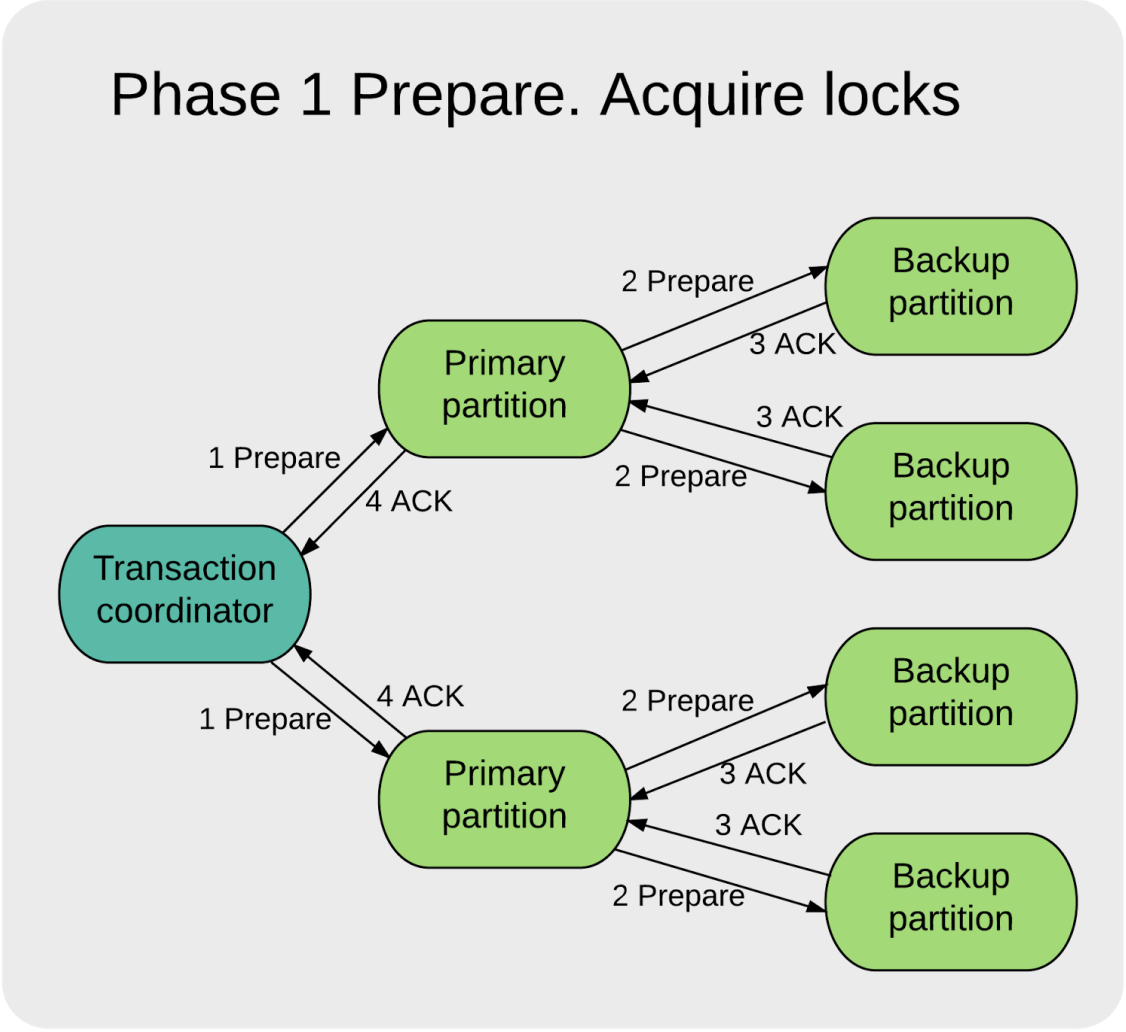
2. Transactions

2.1. Two-Phase-Commit Protocol

В распределенных системах одна транзакция может затрагивать данные расположенные на разных узлах, что накладывает (для соблюдения принципов atomic и consistency) дополнительные требования. Необходимо отслеживать и корректно обрабатывать ситуации, когда изменения будут применены только частью узлов. Или, когда часть узлов выйдет из обслуживания.

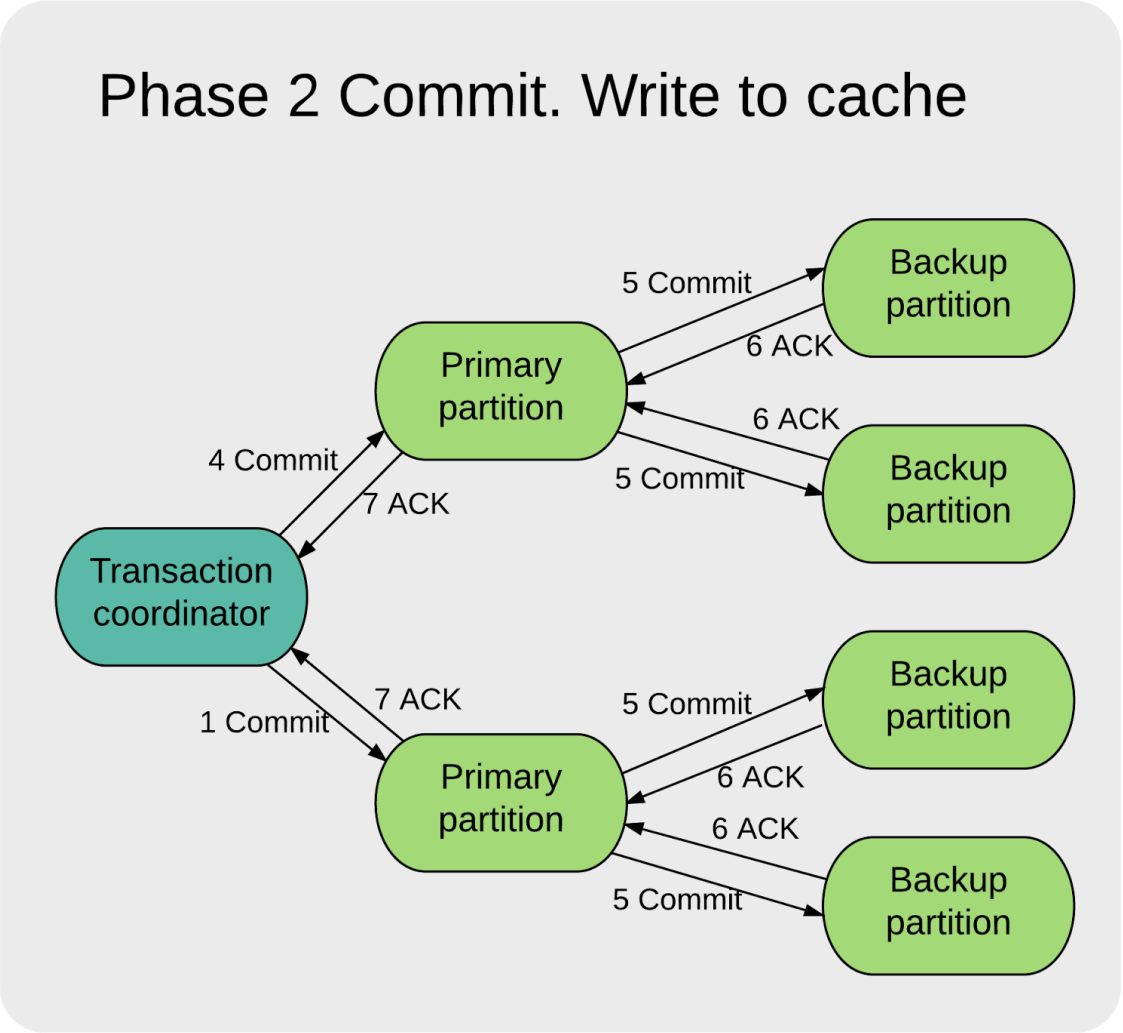
Традиционным алгоритмом для решения таких задач выступает алгоритм 2х-фазной фиксации изменений. В оракле такие транзакции называются распределенными (distributed), они выполняются в нескольких базах, связанных через dblink.

Протокол 2х-фазного коммита, как и следует из его названия, выполняется в два шага.



1. Координатор транзакции, NearNode в терминологии Ignite отправляет всем Primary узлам, участвующим в транзакции, "Prepare" сообщение.
Получив такое сообщение, узел синхронно пытается "захватить" все необходимые блокировки и, в случае успеха, отправляет сообщение, подтверждающее свое участие в транзакции.

2. Координатор, получив подтверждение от всех участников, отправляет "Commit".
После этого узел "фиксирует" транзакцию.

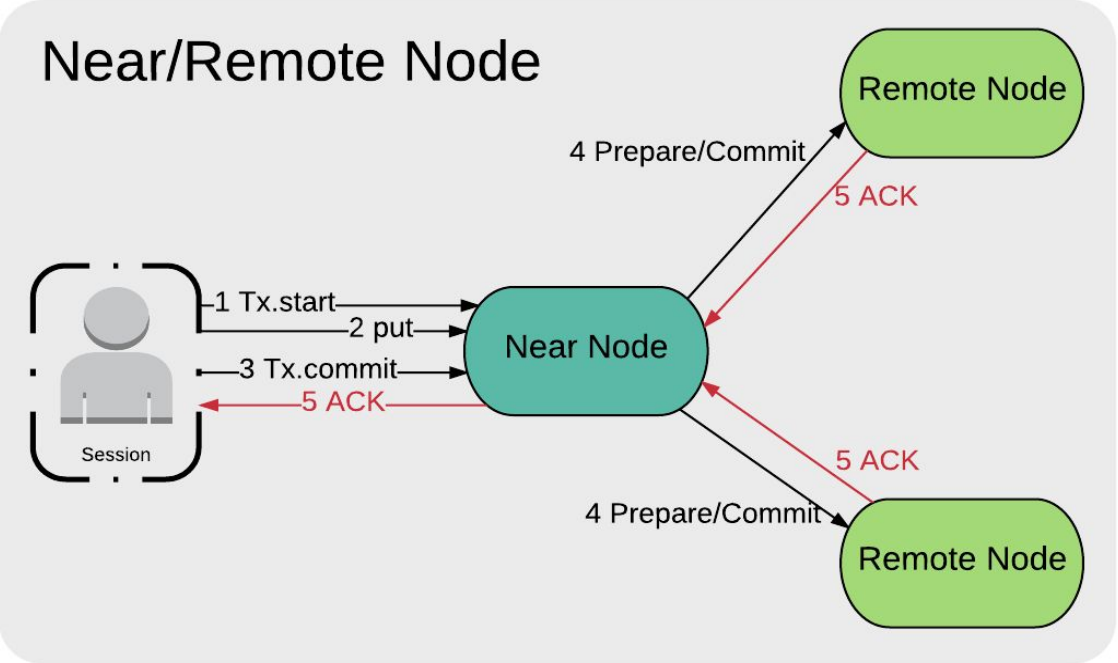


Такой метод гарантирует, что все участвующие узлы одновременно сохранят или отметят изменения транзакции.

В дальнейшем, мы рассмотрим как Ignite поступает в случаях "отказа" на разных шагах подтверждения.

2.2. NearNode/Remote/DHT

Если рассматривать систему с точки зрения транзакции, то необходимо ввести еще одну важную сущность - NearNode.
В алгоритме 2х фазной фиксации для каждой транзакции должен существовать узел, который будет выступать в роли ее "координатора". Он и будет называться NearNode. С точки зрения Ignite - это узел, где транзакция была инициирована и был вызван tx.start. Такой узел "отслеживает" состояние транзакции, затронутые ею ключи, версию топологии старта транзакции, узлы участвующие в транзакции и другие атрибуты контекста транзакции.



В противоположность Near, участвующие в транзакции partitions называются Remote. Они хранят "свою" часть кеша. Физически каждый узел размещает у себя часть DHT (Distributed hash table) и, через афинити функцию, любой участник транзакции может определить ее партиции и узлы. Надо заметить, что DHT хранит бакеты до уровня партиций, а дальше используется B+ tree.

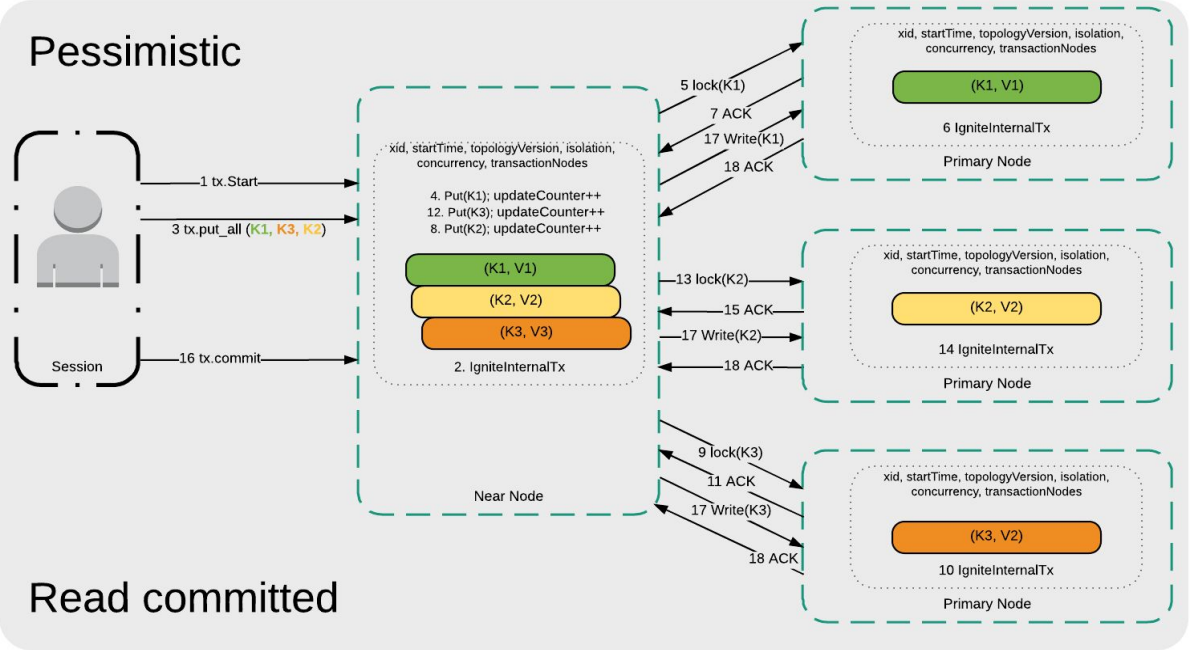
3. Lock mode

3.1. Pessimistic

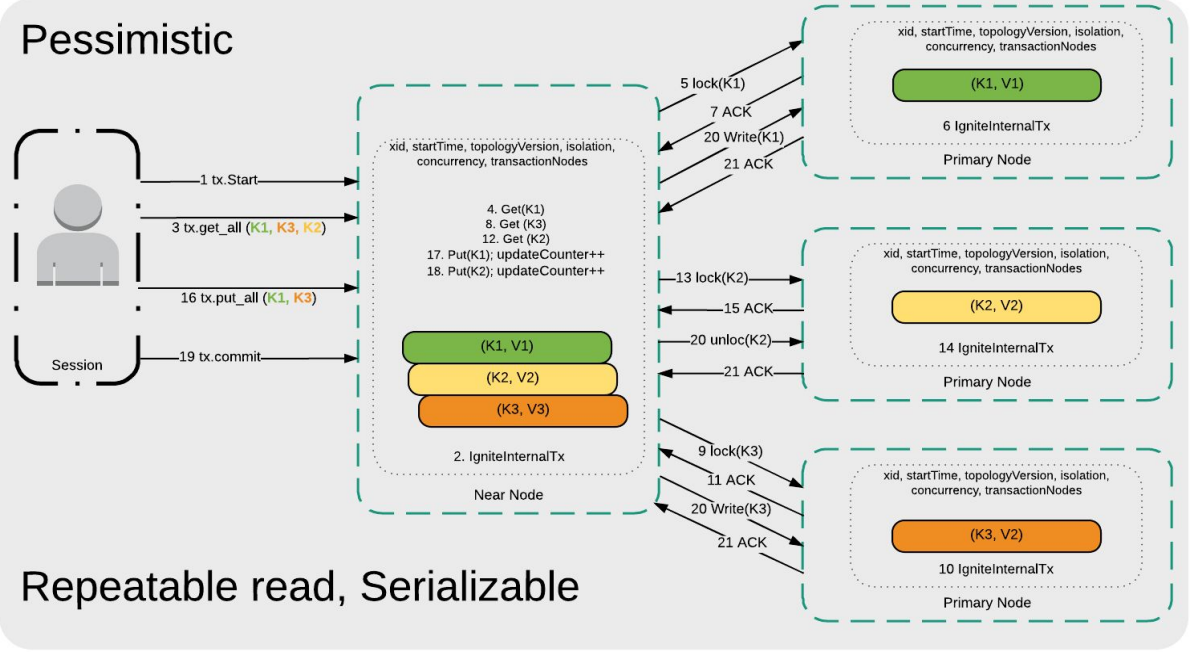
В многопользовательских системах разные пользователи могут одновременно модифицировать одни и те же данные.
Для корректного разрешения таких ситуаций существует два основных подхода - оптимистическая и пессимистическая блокировка.

При пессимистической блокировке система вначале выставляет блокировку на данные, которые она планирует изменить, затем вычисляет новые значения и, гарантированно, изменяет данные.

Так же, для момента получения блокировок играет роль и уровень изоляции транзакции.
Для начала рассмотрим pessimistic режимы.
Для уровня Read committed блокировки приобретаются перед операцией записи (put, put_all).



В режимах Repeatable read и Serializable блокировки приобретаются перед любой операцией (и чтения и записи).



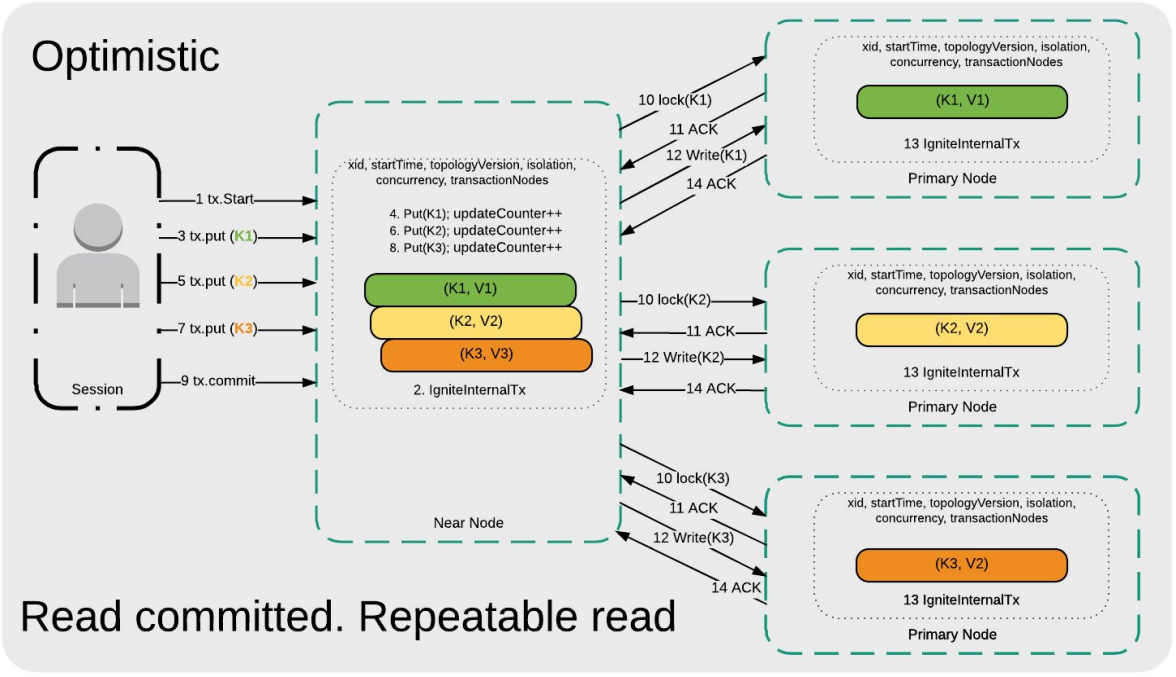
В pessimistic режиме данные остаются заблокированными до окончания транзакции. Для сокращения времени удержания блокировок может использоваться optimistic режим.

3.2. Optimistic

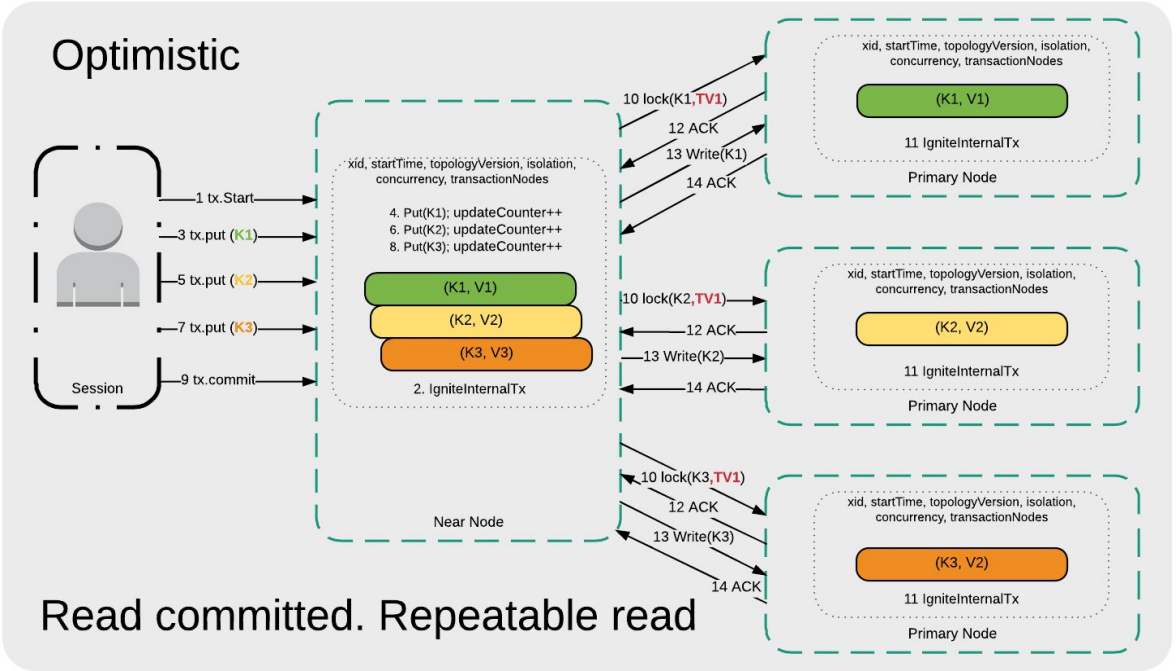
В этом режиме, вначале вычисляется новое значение, затем проверяется, что за время вычислений не произошло конкурирующих изменений. Если это действительно

так, данные блокируются и модифицируются. Если было обновление, необходимо провести вычисление еще раз.

Для уровня изоляции Read committed и Repeatable read блокировки получаются в момент выполнении фазы "prepare", при этом проверка, что версия не изменилась с начала транзакции не выполняется.

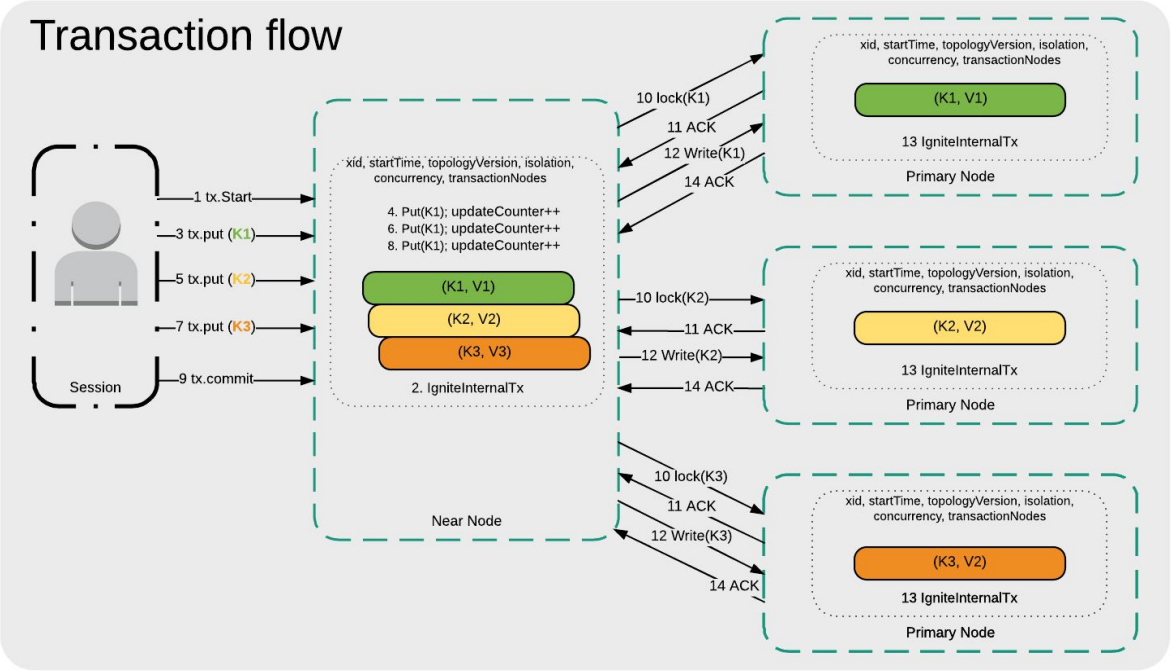


При уровне Serializable, блокировки так же создаются в на фазе "prepare", но проверка неизменности версии выполняется.



4. Transaction flow

Теперь попробуем рассмотреть весь жизненный цикл транзакции в Ignite. Для простоты предположим, что топология кластера стабильна, и не меняется.



Началом транзакции является операция ее старта tx.start. При этом, на координаторе транзакции (NearNode), создается структура для управления контекстом транзакции (интерфейс IgniteInternalTx).

При этом, для транзакции:

- генерируется уникальный идентификатор транзакции,
- запоминается время начала транзакции,
- актуальная версия топологии,
- уровень изоляции и конкуренции, определенный для транзакции, и т.д.

Статус транзакции устанавливается в “активный”.

Дальше алгоритм, в зависимости от уровня изоляции и конкуренции транзакции, может отличаться.
Например, для транзакций с уровнем pessimistic, транзакция сразу проверяет возможность получить блокировку по ключу, а при optimistic, это происходит в момент commit.

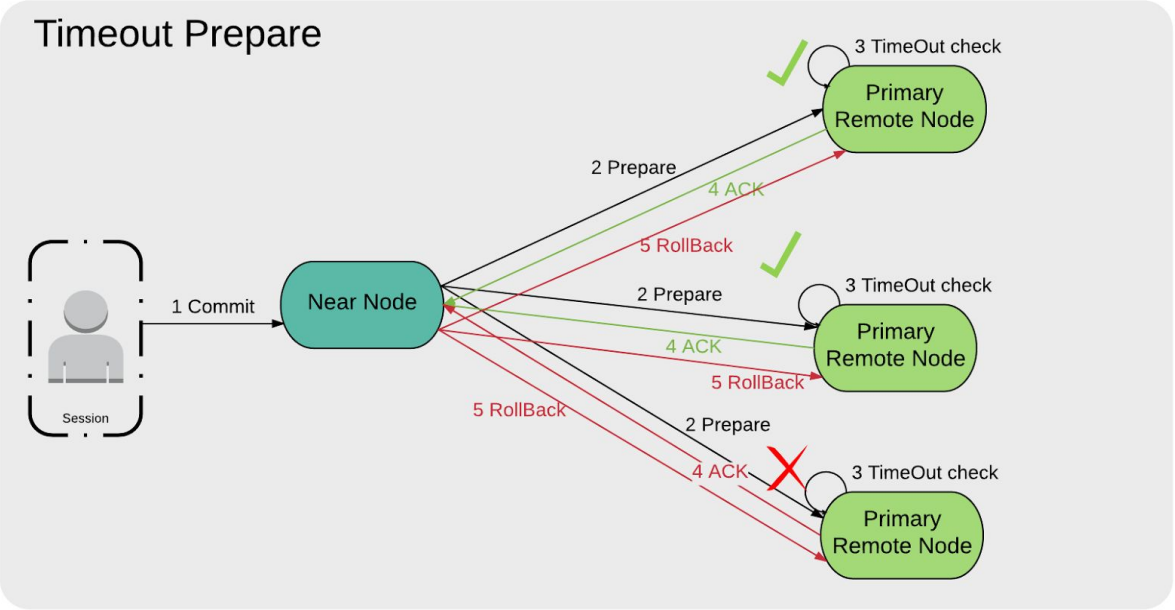
Все выполняемые get операции (в режиме REPEATABLE READ) приводят к чтению данных из кластера и сохранению ключей и значений в контексте транзакции в java heap NearNode.
Все выполняемые put операции также приводят к кешированию ключей и данных в контексте, тоже в java heap NearNode.

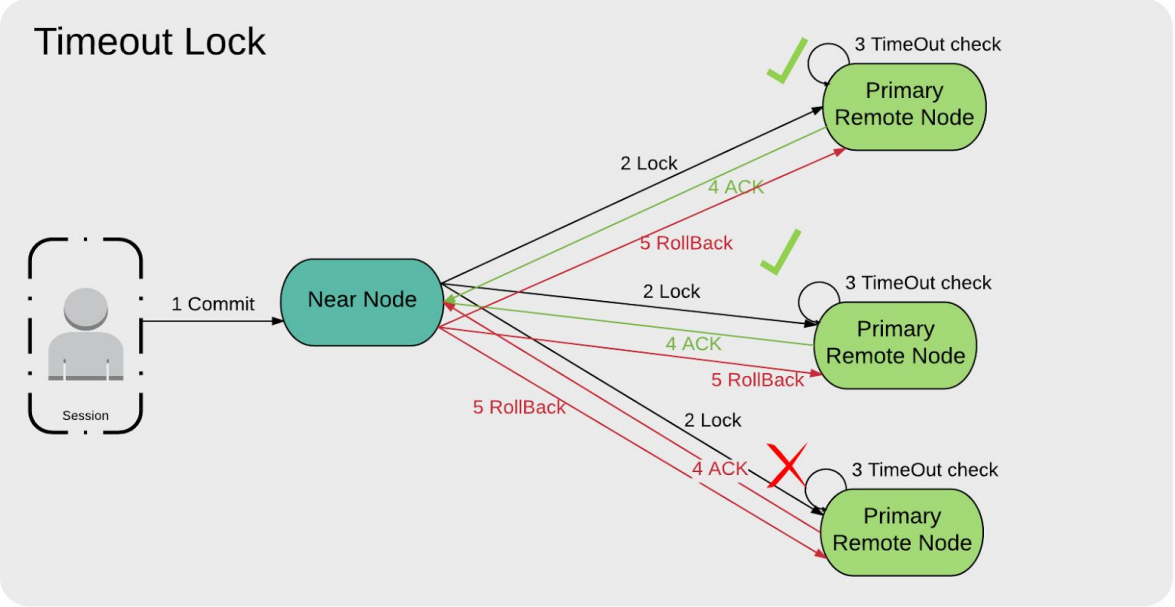
TX Rollback

Если пользователь отменяет транзакцию (выполняется tx.rollback), то, в режиме pessimistic необходимо “отпустить” полученные блокировки, и удалить контекст транзакции. В режиме “optimistic” блокировки приобретаются только в момент выполнения операции commit, поэтому, достаточно просто удалить контекст транзакции на NearNode.

TX Timeout

В настоящее время возможно опционально указание timeout для транзакций, что в случае если транзакция превысит указанное время выполнения, обеспечит ее отмену. При попытке получить блокировку на ключ (если транзакция “pessimistic”) или начать фазу “prepare” (если это “optimistic” транзакция, или все необходимые блокировки для “pessimistic” уже получены) на каждом primary узле транзакции выполняется проверка, что время выполнения транзакции не превысило заданное. Если время работы превысило заданное на любом из узлов, для транзакции выставляется флаг, что она не может быть сохранена, а может быть только отменена. Эта информация отправляется на NearNode. В случае если NearNode получила транзакцию с таким признаком, то, для нее, выполняется обычная процедура отмена.
Дополнительно, для уменьшения влияния “долгих транзакций”, планируется ввести обязательную проверку времени выполнения при старте процесса создания снимка данных, и принудительной отмены таких транзакций.





TX Commit

Если пользователь фиксирует транзакцию (tx.commit) то, на основании контекста транзакции, создается запрос на выполнение шага prepare.

При этом на каждый primary узел отправляется информация по новым или измененным ключам и, если это важно, информация о порядке получения блокировок.

Primary узлы

- проверяют, что версии топологии транзакции и узла совпадают;
- получают необходимые блокировки;
- создают DHT контекст для транзакции и сохраняют в него необходимые данные;
- в зависимости от режима в котором работает кеш, дожидаются или нет подтверждения об успешности шага prepare от Backup узлов;
- информируют координатор транзакции о готовности выполнить commit.

NearNode отправляет commit сообщение, дожидается подтверждения и переводит транзакцию в статус COMMITTED.

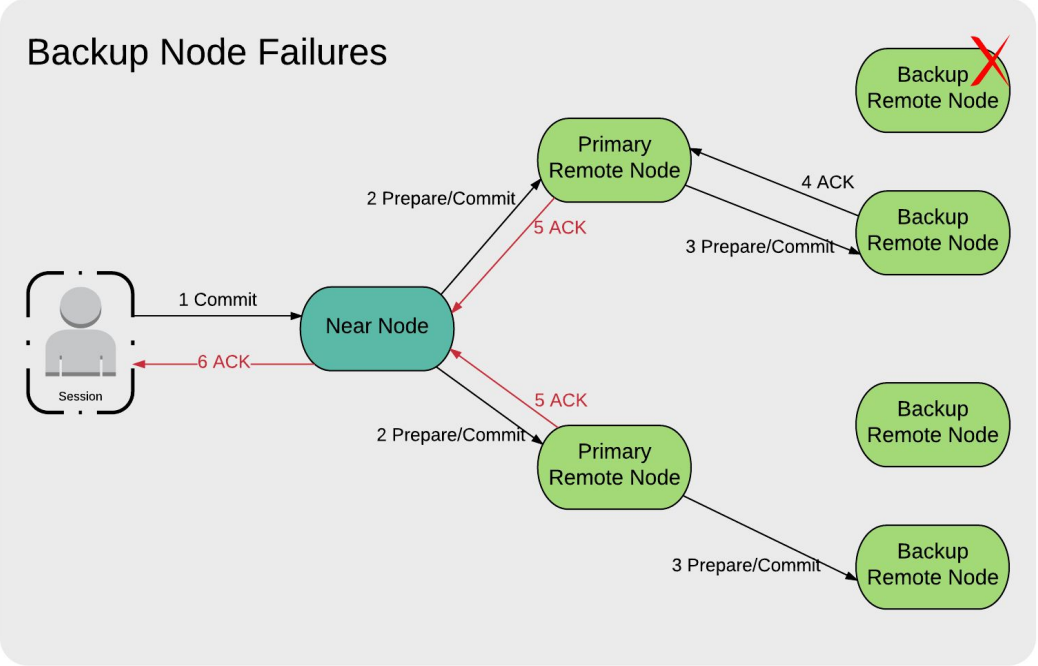
Еще раз важно обратить внимание, что все данные транзакции должны помещаться в heap памяти NearNode. В противном случае транзакция завершится ошибкой OutOfMemory.

Посмотрим теперь как обрабатываются не позитивные сценарии.

5. Failover/Recovery

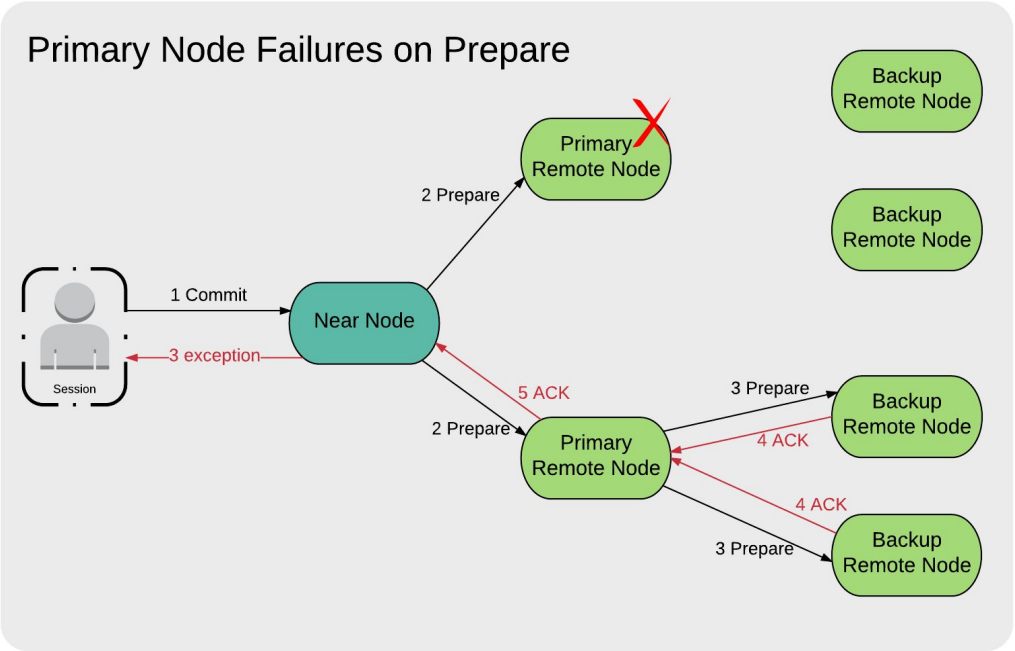
Для сохранения работоспособности кластера, особенно в системах с большим числом участников, необходимы механизмы предотвращения или исправления отказов любого из элементов.

Backup Node Failures



Самым простым случаем отказа является отказ Backup узла в ходе фазы "Prepare" или "Commit". В случае такого события выполнения специальных действий не требуется. Данные сохраняются на Primary и оставшихся Backup узлах. Далее, вне скоупа транзакции, для партиции создается новая резервная копия.

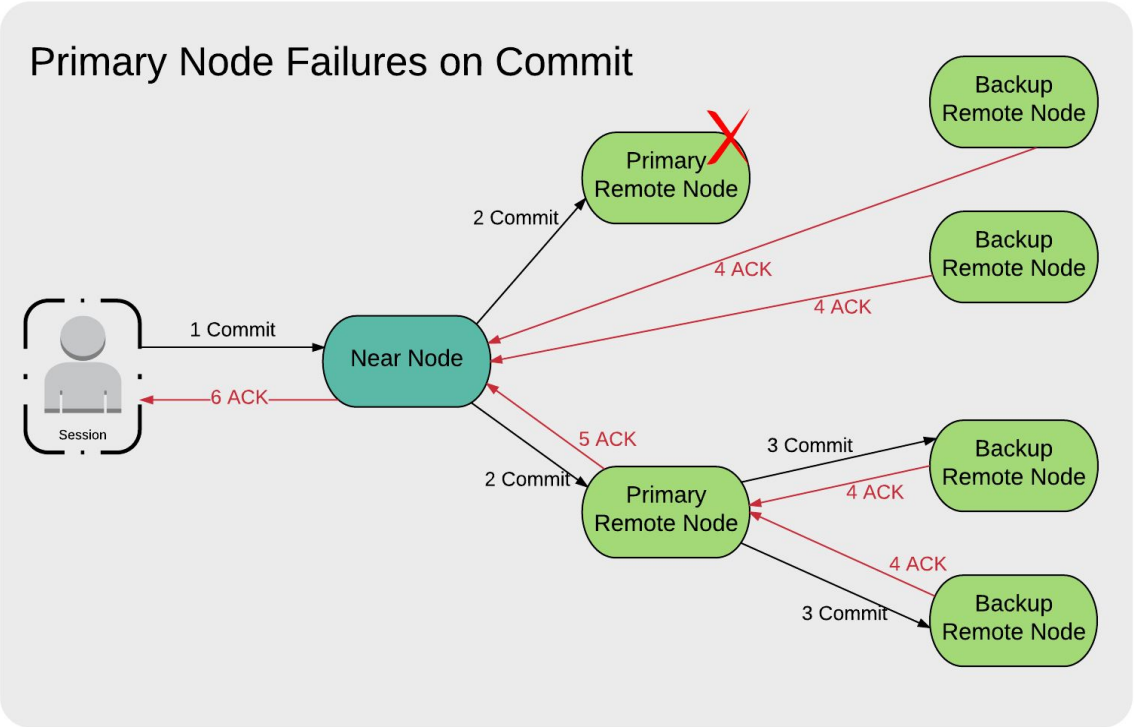
Primary Node Failures on Prepare



Для исправления "отказа" Primary узла требуется алгоритм более сложный.
Если отказ случился перед или во время "Prepare" стадии, и координатор транзакции не смог подключиться к Primary, то он инициирует исключение, и клиент должен явно принять дальнейшее решение. По отмене или повторной попытке сохранить результаты (но уже в новой топологии и, следовательно, новой транзакции).

Primary Node Failures on Commit

Если отказ случился после "Prepare" стадии, то координатор транзакции, определив недоступность Primary узла будет ожидать (NodeFailureDetection) ответа от Backup узлов.
Backup узлы, определив "пропажу" Primary, отправляют сообщение координатору транзакции (информация об инициаторе, и всех участниках транзакции включена в сообщение полученное на стадии prepare).
Топология кластера при этом не меняется, т.к. для смены необходимо прохождение сообщения PME, а оно будет ожидать окончания активных транзакций.
Координатор транзакции подтверждает ее, транзакция перестает быть активной, изменяется топология.



Coordinator Node Failures

При "отказе" координатора транзакций требуется больше действий. Каждый участник располагает информацией только по локальному статусу транзакции и, скорее всего этот статус будет различным на разных узлах. Возможно, что кто-то из участников уже получил "Commit" сообщение, или кто-либо не получил "Prepare". Поэтому, все Primary узлы отправляют сообщения с информаций о текущем статусе. Если один из узлов ответит, что сообщение "Prepare" к нему не поступало (или не ответит вообще), то транзакция отменяется на всех узлах.