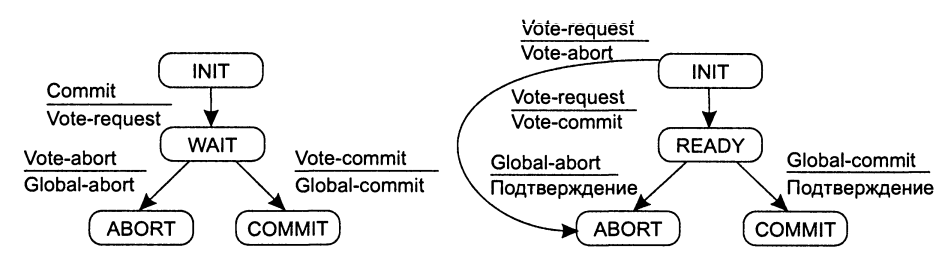
Лекция 12

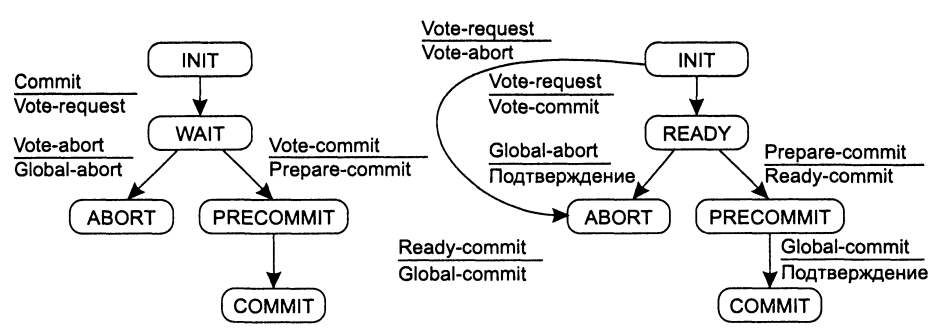
РИС, III курс, ИСиТ

**Распределенное подтверждение**

1. **Постановка задачи:** распределенная транзакция (двухфазная блокировка); как организовать распределенный COMMIT (distributed commit – распределенное подтверждение)? на одном из узлов РИС, может быть отказ, откатиться должны операции на всех узлах; как организовать распределенный ROLLBACK (distributed rollback – распределенный откат)?
2. **Координатор** –один из узлов распределенной системы, участвующих в распределенном подтверждении и управляющий этим процессом; как правило, координатором является инициатор распределенного подтверждения (узел, выполняющий распределенную транзакцию).
3. **Протокол однофазного подтверждения**: координатор выдал commit и сообщил всем участникам распределенной транзакции, что надо выполнить commit; если участник не может выполнить локальный commit, он не имеет механизма сообщить об этом.
4. **Протокол двухфазного подтверждения (2PC):**
5. координатор перед выполнением локального COMMIT, рассылает всем участникам сообщение **vote\_request** и переходит в состояние ожидания;
6. участник получив **vote\_request** проверяет может ли он выполнить COMMIT, если да, то возвращает **vote\_commit** координатору, если не может возвращает **vote\_abort**;
7. координатор собирает ответы участников, если все **vote\_commit**, то рассылает всем участникам **global\_commit**; если есть хотя бы один участник ответил **vote\_abort,** то рассылает всем участникам **global\_ rollback**;
8. каждый из проголосовавших участников ждет итогового сообщения координатора; если **global\_commit**, то выполняет COMMIT, если **global\_rollback**, то выполняет ROLLBACK (ABORT).
9. Основное преимущество 2PC: участник может попробовать выполнить действие предусмотренное распределенной транзакцией. Например, участник может проверить соответствие операции констрейнтам, если соответствует, то **vote\_commit**, иначе **vote\_abort.**

****

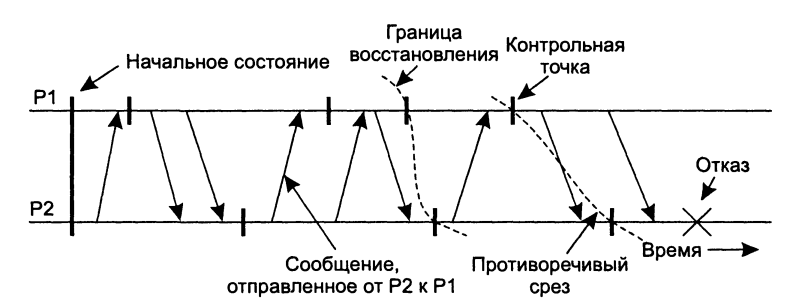
1. **Проблема 1:** участник выполнил операцию и ждет от координатора **vote\_request**, а его нет. Устанавливается timeout, после которого отсылается координатору **vote\_abort** и выполняется локальный ROLLBACK.
2. **Проблема 2:**  координатор не получает ответов **vote\_commit/vote\_abort** от всех участников. Устанавливается timeout, после которого отсылается всем участникам **global\_rollback** и выполняется локальный ROLLBACK (ABORT).
3. **Проблема 3:**  участник не получает ответ **global\_commit/global\_rollback** от координатора. Устанавливается timeout, после которого осуществляется опрос других участников (если они есть) если, кто-то получил **global\_commit**, то выполняется локальный COMMIT, иначе ROLLBACK (ABORT).
4. **Процесс восстановления участника:** если на узле был сбой после отправки координатору **vote\_commit**, то после восстановления работоспособности узел должен решить, что делать COMMIT/ROLLACK(ABORT); один из вариантов связаться с другим участником или координатором выяснить; для полноценного процесса восстановления требуется сохранение информации в журналах.
5. **Проблема 4:** координатор всем отравил **vote\_request** и сам сломался, участники никак не смогут определить, что им дальше делать; обычно ждут восстановление координатора. Поэтому 2PC – протокол блокирующего подтверждения.
6. **Не путать с двухфазным блокированием данных при распределенной транзакции.**
7. **Протокол трехфазного подтверждения (3PC)**: используется редко. Суть протокола: состояние координатора и любого из участников удовлетворяет следующим 2м условиям:
8. нет такого состояния, при котором возможен прямой переход в состояния COMMIT или ROLLBACK (ABORT);
9. нет такого состояния, в котором невозможно принять итоговое решение, но возможен переход в COMMIT.
10. **Протокол трехфазного подтверждения (3PC)**:
11. координатор рассылает всем участникам **vote\_request** и ждет ответы;
12. если хотя бы один из участников отвечает **vote\_abort**, координатор рассылает всем **global\_rollback**;
13. если все участники ответили **vote\_commit**, координатор рассылает всем **prepare\_commit**;
14. если все участники ответили **redy\_commit**, координатор рассылает всем **global\_commit**;
15. если участник получил **global\_commit**, он выполняет локальный COMMIT.



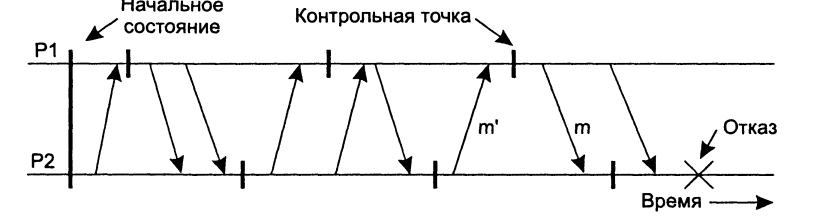
1. **Проблема 1:** координатор в состоянии PRECOMMIT не дождался ответа **redy\_commit** от участника, после timeout делает вывод об отказе участника, исключает его из группы и остальным высылает **global\_commit**; участник после обнаружения, что он вне группы, запускает процедуру включения в группу с восстановлением.
2. **Проблема 2:** участник может заблокироваться в состояниях REDY или PRECOMMIT; связывается с другим участником и выясняет состояние; если все участники находятся в PRECOMMIT выполняется локальный COMMIT.

**Восстановление**

1. **Основа отказоустойчивости:** восстановление после ошибок.
2. **Два способа восстановления после ошибок:** обратное исправление, прямое исправление.
3. **Обратное исправление**: возвращение системы из текущего ошибочного состояния в предыдущее безошибочное состояние. Для этого периодически на узлах делаются контрольные точки, к которым можно откатиться. Точки необходимо синхронизировать (в начале семестра рассматривался алгоритм сохранения состояния всей распределенной системы). Пример: БД. Пример: связь – запрос на повторную отправку пакета. Этот метод применяется чаще. Обратное исправление – затратный механизм (контрольная точки). Не все операции являются обратимыми. Контрольные точки применяются совместно с протоколированием (протоколы – информация о сообщениях, откат и накат). Протоколы могут размещаться как на координаторе так и на участнике.
4. **Прямое исправление:** выполняется не откат, а накат, делается попытка перевести систему в новое согласованное состояние. Пример: связь – восстановление потерянного пакета из других (при блочном кодировании).
5. **Хранилища данных 3 вида:** оперативная память (после выключения или отказа процессора стирается), дисковая память (поломка головок); устойчивые хранилища.
6. **Устойчивые хранилища:**  RAID: зеркало, stripping, специальные диски для кодов исправления и пр.
7. **Контрольная точка:** непротиворечивый глобальный снимок распределенной системы (принципы построения рассматривались в начале курса), сохраненный в устойчивом хранилище.
8. **Граница восстановления:** последний распределенный непротиворечивый снимок состояния, который соответствует непротиворечивому срезу.
9. **Непротиворечивый срез**: множество локальных снимков состояния, образующих распределенный непротиворечивый снимок. Задача при восстановлении – найти такой срез. В локальных снимках не должны быть зафиксированы принятые сообщения, отправка которых не зафиксирована в других локальных снимках образующих глобальный снимок.



1. **Независимое создание контрольных точек:** локальные снимки создаются независимо друг от друга. Такой подход может привести к каскадному откату в самое начало – эффект домино.

****

1. **Координированное создание контрольных точек:** создание контрольных точек с помощью синхронизации всех процессов. Полученная контрольная точка является глобально непротиворечивой. Можно использовать алгоритм распределенного снимка, рассмотренный выше.
2. **Двухфазный протокол блокировки:** 1) координатор рассылает запрос checkpoint\_request; 2) узел, получивший checkpoint\_request, создает локальную контрольную точку, отправляет координатору checkpoint\_response, все дальнейшие выходные сообщения буферизирует; 3)координатор получив от всех узлов checkpoint\_response, отправляет всем узлам checkpoint\_done; 4) узел, получив checkpoint\_done, последовательно отправляет сообщения из очереди (восстановление).
3. **Инкрементные снимки состояния:** узел-координатор, имеет сохраненное локальной состояние; 1) рассылает checkpoint\_request только тем узлам, которым оправлял сообщения после последнего сохраненного состояния, все дальнейшие выходные сообщения буферизируются; 2) узлы, получившие checkpoint\_request, рассылают сообщения в узлы, в которые они оправляли сообщения после последнедего сохранения состояния, все входные сообщения буферизируются; 3) и т.д; 4) получив checkpoint\_response от всех узлов выполняет сохранение состояния и отправку checkpoint\_response; 5) координатор получив ото всех checkpoint\_response, оправляет им же checkpoint\_done для дальнейшего распространения и разблокировки.
4. **Что будет если два узла обменялись сообщениями после последней контрольной точки?**

1. **Протоколирование сообщений:** создание контрольных точек – затратная операция, технология сокращающая затраты – протоколирование сообщений. Протоколирование сообщений – запись в устойчивое хранилище всех сообщений между контрольными точками. Позволяет сократить количество контрольных точек за счет возможности повторить сообщения. Аналогия: полная резервная копия базы данных и журнал повтора (Oracle).
2. **Кусочно-детерминированная модель:** получение сообщений в общем случае не детерминированно (может быть несколько допустимых последовательностей событий в распределенной системе), если есть механизм, позволяющий правильно упорядочивать события, то их можно протоколировать, т.е. задавать событиям вполне определенный порядок. Протоколирование позволяет процесс восстановления сделать детерминированным, т.е. при каждом повторе восстановления будет получен одинаковый результат.