

Сравнительный анализ хранилищ данных

Коринский и Царев...
Плоды многолетних размышлений
v. 2.0 @ #DevPoint2 2010

О чём этот доклад?

- Существует пропасть между разработчиками и DBA
- Отсутствуют популярные материалы по данной теме
- Доклад является обзорным
- Авторы - не DBA!

Зачем нужны транзакции?

- Примеры:
 - Оплата счёта
 - Размещение поста в блоге
 - Покупка билета

Транзакции

- Транзакция - <здесь по идее нужно определение>
- Наивно: последовательность действий, со следующими свойствами:
 - Atomic
 - Consistence
 - Isolated
 - durability

Транзакции

- Atomic: Последовательность действий применяется целиком, либо не применяется вообще
- Consistence: Действия оставляют базу в непротиворечивом состоянии.
- Isolated: транзакции работают независимо
- Durability: после применения не может потеряться

Партицирование

- Партицирование - деление данных на части между различными файлами, разделами диска или машинами в кластере.
- Примеры:
 - Социальные сети хранят тексты постов и картинки пользователей на различных серверах
 - Информация о клиентах мобильного оператора распределена по регионам
 - Сервер для обработки пользователя выбирается по первой букве имени

Аспекты партицирования

- Виды: способы деления данных на части
- Назначение: какие проблемы решает
- Примеры: использование в реальных проектах
- Environment: наличие и условия решений
- Транзакции: как партицирование влияет на транзакции

Функциональная декомпозиция

- Суть: Данные бьются на части в соответствии с семантикой.
- Возможные критерии:
 - Тип данных (текст, картинки, видео)
 - Изменчивость данных (статический контент или динамический?)
 - Назначение данных (личные данные пользователей, публичные сообщения)

Горизонтальное партицирование

- Суть: Множество однотипных данных разделяется на (равные) части по некоторому критерию
- Критерии:
 - Географическая рассредоточенность (в каждом регионе своя база)
 - Надежность сервиса (обычные пользователи/платные аккаунты)
 - Дата поступления данных (удобство поддержки, снижение нагрузки)
 - Хеш-функция: отображение большего множества в меньшее (примеры далее)

Горизонтальное партиционирование

- Примеры хеш-функций:
 - Числа: остаток от деления на десять (десять возможных значений)
 - Логин пользователя: первая буква ника (мощность алфавита - множество возможных значений)
 - "Идеальная": разбивает множество на равные части (таблица значений функции)

Вертикальное партиционирование

- Суть: Однотипные данные разделяются группируются в части по атрибутам
- Критерии:
 - Востребованность (часть колонок нужна редко)
 - Масштабируемость (равномерно распределяем данные)

Column/Row-oriented system

- Суть: данные организуются либо по строкам, либо по колонкам
- Row-oriented:
 - Все атрибуты каждой записи лежат рядом
 - Записи лежат в файле последовательно
- Column-oriented:
 - Для каждого атрибута отдельный файл
 - При чтении необходимо делать соединение

Enviroment

- Операционные системы
- Файловые системы
- Сеть
- Аппаратура
- Поддержка на уровне хранилища

Транзакции и партицирование

- Транзакции вносят свои проблемы:
 - atomic: синхронность применения (консистентность)
 - consistence: издержки на синхронизацию
 - isolated: всё хорошо
 - durability: что там с отказоустойчивостью?

Репликация

- Репликация - способ синхронизации (приведение в соответствие) частей данных
- Виды репликации:
 - master-slave
 - master-master

Master-Slave репликация

- Все изменения применяются на master
- Slave-узлы получают обновления с master
- Slave-узлов может быть несколько

Назначение

- Резервные копии - плохая идея
- Распределение нагрузки
- Таргетирование запросов

Таргетирование запросов

- Различные хранилища на master/slave:
 - master - OLTP, slave – OLAP
 - master - SQL, slave - документо-ориентированная база
 - Скорей всего репликацию придётся писать руками

Таргетирование запросов

- Различная схема данных:
 - master содержит просто таблицы для записи
 - slave имеет дополнительные индексы
 - slave имеют более узкие типы данных

Таргетирование запросов

- Различная схема данных:
 - master содержит просто таблицы для записи
 - slave имеет дополнительные индексы
 - slave имеют более узкие типы данных

Master-Master репликация

- Суть: Изменения производятся одновременно на нескольких master
- Защищаемся от выхода из строя master-узла
- Снижение нагрузки на master-узел
- Снижение нагрузки от slave-узлов

Транзакции и репликации

- Транзакции вносят свои проблемы
 - master-slave: транзакции на slave запаздывают
 - master-master: как применять транзакции?

Consistency

- Система всегда выдает корректные, непротиворечивые, ответы.
- Данные не могут потеряться после завершения записи.
- Не может быть так, что один и тот же запрос к данным (выборка данных, поиск, и т.д.) в зависимости от узла выдавал различные данные.

Availability

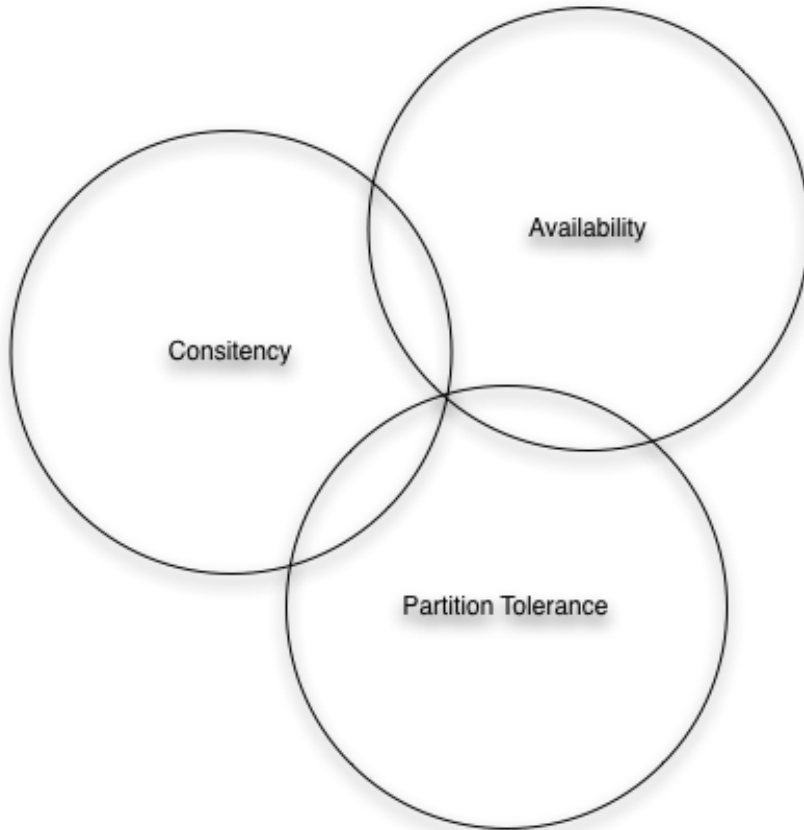
- Система обязана *всегда* выдавать ответы - независимо от аппаратных сбоев отдельных узлов.

Partition tolerance

- Система продолжает работать *корректно* при недоступности части узлов.

Проблема CAP теоремы

- Только два из трех



Про что говорим

- Voldemort, Riak, cassandra, MySQL cluster
- MongoDB, Hbase, memcache/memcached, berkleydb
- PostgreSQL, MySQL, Oracle, TimesTen

+Consistency +Availability -Partition tolerance

- PostgreSQL
- MySQL
- Oracle
- TimesTen

PostgreSQL

- Очень много разных индексов
- PostGIS
- Репликация внешними средствами
- Сложна

MySQL

- Привычная база
- Есть репликация
- Простота поддержки

Oracle

- Умеет все
- Дорого

TimesTen

- Быстрая in memory база
- Простота
- Цена

PostgreSQL vs MySQL vs Oracle vs TimesTen

- есть деньги oracle/timesten
- хватает – postgres
- иначе mysql

+Availability +Partition tolerance -Consistency

- Voldemort
- Riak
- Cassandra
- MySQL cluster

Voldemort

- Написан на Java, REST
- Key-value
- LinkedIn
- разные backend

Riak

- Написан на Erlang, REST
- Key-value
- разные backend

Riak vs Voldemort

- **Одинаковы**

Cassandra

- Написана на Java, Thrift api
- Column-oriented
- Разные датацентры

MySQL cluster

- Написана на C
- Честный SQL
- in memory

Riak/Voldemort vs Cassandra vs MySQL cluster

- СОВМЕСТИМОСТЬ MySQL
- map reduce: Riak

+Consistency +Partiotion tolerance -Availability

- MongoDB
- Hbase
- memcache/memcached
- berkleydb

MongoDB

- Написана на C++, BSON
- Key-value
- github, sourceforge, bit.ly, disqus

Hbase

- Написана на Java, REST
- column-oriented

memcache/memcached

- Написаны на С, Простой протокол
- Key-value
- используется везде

berkeleydb

- Написана на C, C-API
- Key-value
- СТОИТ ДЕНЕГ

MongoDB vs Hbase vs memcached vs berkeleydb

- кеш memcached
- hbase как замена bigtable
- встраиваемая berkeleydb

CAP Solution

- Жертвуем Consistency
- Разбиваем систему на части
- Система в разные моменты времени находится в разных позициях CAP теоремы

Счастья нет

- У каждой задачи свои требования
- У каждого хранилища свои сильные и слабые стороны
- Хранилище под задачу, а не задачу под хранилище
- Одного хранилища мало
- Правильное решение - всегда компромис

Any question?

Oleg Tsarev

oleg.tsarev@percona.com

<http://percona.com/>

Kirill A. Korinskiy

kkorinsky@rooxteam.com

<http://www.roox.ru/>



PERCONA
Performance Consulting Experts

