Сравнительный анализ хранилищ данных

Коринский и Царев...

Плоды многолетних размышлений

v. 1.0 @ #ADD 2010

Введение

Содержание

- Постановка проблемы
- Что такое распределенные хранилища
- Обзор доступных хранилищ

Друзья в социальной сети

Задача: получить список друзей.

- Facebook - 500M пользователей - 5M rps
- ВКонтакте - 100M пользователей - 1M rps
- Мой Мир - 50M пользователей - 0.5M rps
- Toy net - 10 пользователей - <1 rps

Матрица смежности

таблица:

- оси таблицы – пользователи;
- ячейки таблицы – связь.

- ВКонтакте/Facebook: один бит
- Мой Мир: шесть байт
- Toy net: один бит

Список смежных вершин

каждый элемент:

- пара (UID, UID), UID – 4 байта):
  - Facebook
  - Вконтакте
  - Toy net
- кортеж (UID, UID, Info – 6 байт):
  - Мой Мир

Генерация списка друзей

каждый элемент:

- Facebook/ВКонтакте - 8 байт.
- Мой Мир - 14 байт.
- Toy net - 1 байт?

Генерация списка друзей

Матрица смежности:

- Сканируем строку таблицы
- Проверяем каждый столбец строки
- Каждая ячейка сообщает отношения

Генерация списка друзей (просто)

Список смежных вершин:

- Каждый элемент - (Кто?, С кем?, Что?)
- Фильтруем список по "Кто?"
- Усеченный список содержит всех друзей

Генерация списка друзей (сложно)

Список смежных вершин:

- Каждый элемент - (Кто?, С кем?, Что?, Следующий)
- Находим первый элемент с нужным Кто?
- Вытягиваем весь список

Масштабы: memory

Матрица смежности:

- Facebook: 28421 Петабайт
- ВКонтакте: 1136 Петабайт
- Мой Мир: 13642 Петабайт
- Toy net: 16 байт

Масштабы: memory

Список смежных вершин (сложно):

- Facebook: 69 Гигабайт
- ВКонтакте: 14 Гигабайт
- Мой мир: 97 гигабайт
- Toy net: 200 байт

Масштабы: latency

DDR3 ECC2 (DDR3-1066E):

- 8 гигабит в секунду максимум
- 8 гигабайт - $1500-2000
- Страница памяти - 4 килобайта
- 2 миллиона страниц в секунду

Масштабы: latency

Матрица смежности:

- Facebook: 16 rps
- ВКонтакте: 80 rps
- Мой мир: 40 rps
- Toy net: 2 000 000 rps

Масштабы: latency

Список смежных вершин (сложно):

- 150 друзей у каждого в среднем
- 150 страниц на генерацию списка
- Итог: 14 000 rps

Масштабы: memory/latency

Список смежных вершин (просто)

- Нет смысла рассматривать, full scan

Оптимизация матрици смежности

- Матрица смежности сильно разряжена
- Можно производить адаптивную упаковку
- Реализация станет сложней

Оптимизация списка смежных вершин

- Список смежных вершин можно группировать по "Кто?"
- Возрастёт потребление памяти
- Реализация станет сложней

Видео: постановка задачи

- Существуют различные кодеки
- Разные характеристики:
  - Размер ролика
  - Потребляемая при конвертации память
  - Количество вычислений при конвертации
  - Качество записи

Видео: решение

- Несжатое видео
- H.264

Видео: "Экстремумы"

- CPU-limited: H.264 требует много вычислений.
- Memory-limited: Несжатое видео требует много памяти для хранения
- Latency-limited: Несжатое видео требует много памяти для хранения
- Latency-limited: H.264 требуети время на конвертацию

Видео: выводы

- В задаче с видео тоже есть "экстремумы"
- Каждый их решений имеет свои минусы
- Три точки экстремума: CPU, Memory, Latency.

Основные оси

- CPU
- Memory
- Latency
- Сложность

Производные оси

- Бюджет проекта
- Сроки
- Опыт разработчиков
- Аппаратура
- Имеющиеся инструменты

Выводы

- Основная задача HighLoad - поиск локального оптимума.
- Иногда задача не разрешима в имеющихся условиях
- Наша цель - познакомить слушателей с ключевыми факторами

Виды партицирования

- Функциональная декомпозиция
- Горизонтальное партицирование
- Вертикальное партицирование

Manual partitioning

- Разделяем данные на уровне приложения
- Требует дополнительной работы
- Теряется транзакционность

Automatic partitioning

- Тяжело поддержать на уровне хранилища
- Приходится указывать критерии партицирования
- Универсальные критерии не работают

Иные аспекты партицирования

- На каждом узле выбирается:
  - Аппаратура
  - Хранилище
  - Операционная система
- В случае HDD хранилища: файловая система

Репликация

- Master-Master: все узлы равноправны
- Master-Slave: главные и ведомые узлы

offline транзакции

Для клиента транзакция завершается после исполнения на текущем узле.

online транзакции

Для клиента транзакция завершается после исполнения на всех узлах.

Consistency

- Система всегда выдает корректные, непротиворечивые, ответы.
- Не может быть так, что после записи данные потерялись.
- 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 выдавать всегда выдавать ответы - независимо от аппаратных сбоев отдельных узлов.

Partition tolerance

- Система продолжает работать корректно при недоступности части узлов.

Пример: +A +C -P

- Обычная СУБД

Пример: + C + P - A

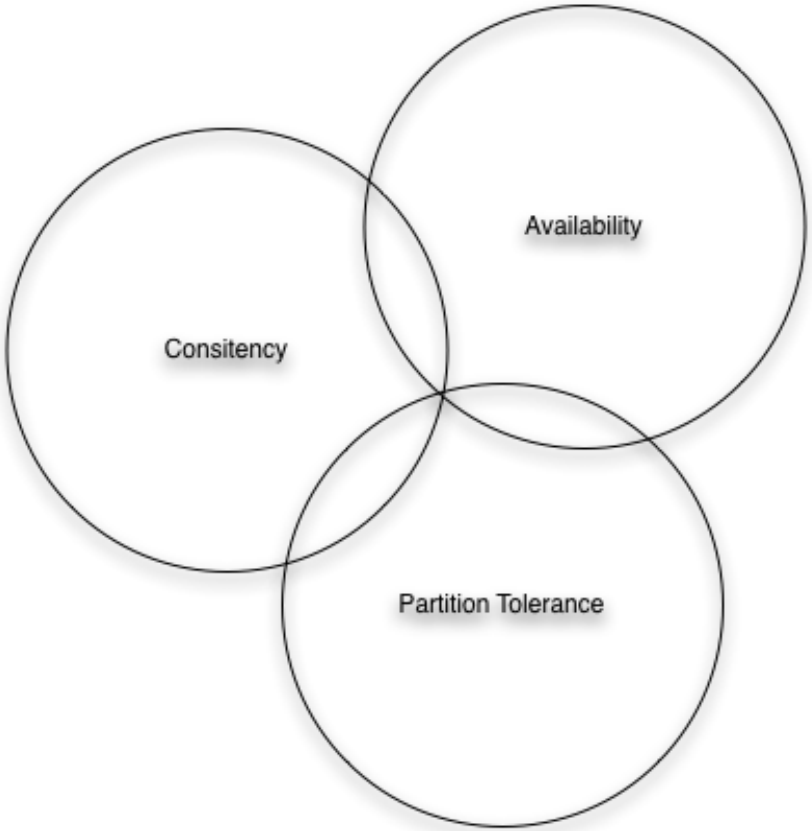
- Система с online транзакции

Пример: +A +P -C

- Система с offline транзакциями

Проблема CAP теоремы

- Только два из трех



CAP Solution

- Жертвуем Consistency
- Разбиваем систему на части
- CAP теорема работает в каждый момент времени

Interface: access

- Library: клиентская библиотека
- ODBC: стандартный интерфейс доступа
- Embedded: встраиваемая база
- Telnet: простой сетевой протокол

Interface: method

- API: интерфейс доступа
- DSL: domain-speicific-language
- SQL (суть DSL): structured query language

Interface: abstraction level

- Search: поиск и выборка значений по ключу
- Query (Search included): поиск, выборка и обработка значений

Interface: реализации

- PostgreSQL: lib/ODBC, SQL, Query
- MySQL: lib/ODBC, SQL, Query
- MySQL NDB-cluster: lib, SQL, Query
- Oracle: lib/ODBC, SQL, Query
- Oracle Timesten: ODBC, SQL, Query
- Riak: lib, API/DSL, Query
- MongoDB: lib, API/DSL, Query

Interface: реализации

- Hadoop: lib, API, Query
- Mnesia: embedded, DSL, Query
- memcached: telnet, DSL, Search
- memcachedb: telnet, DSL, Search
- berkleydb: embedded, API, Search
- voldemort: lib, API/DSL, Query
- cassandra: lib, API/DSL, Search

Storage

- RAM: все данные находятся в оперативной памяти
- Filesystem: система использует для хранения файлы
- Raw partition: система использует раздел диска

Storage: реализации

- PostgreSQL: FS, RAM
- MySQL: FS, RAM
- MySQL NDB-cluster: RAM
- Oracle: Raw, FS, RAM
- Oracle Timesten: RAM
- Riak: FS, RAM
- MongoDB: FS

Storage: реализации

- Hadoop: FS
- Mnesia: FS, RAM
- memcached: RAM
- memcachedb: FS
- berkleydb: FS, RAM
- Voldemort: FS, RAM
- Cassandra: FS

Persistent

- Полная (full): данные гарантированно не теряются
- Частичная (partial): может потеряться последние несколько транзакций
- Отсутствует (loose): данные не сохраняться вообще

Persistent: реализации

- PostgreSQL: full
- MySQL: full, partial
- MySQL NDB-cluster: partial, loose
- Oracle: full, partial, loose
- Oracle Timesten: partial, loose
- Riak: full, loose
- MongoDB: full

Persistent: реализации

- Hadoop: full
- Mnesia: full, loose
- memcached: loose
- memcachedb: full
- berkleydb: full, partial
- Voldemort: full, loose
- Cassandra: full

Динамическая типизация

- Гетерогенный доступ
- Единообразный доступ

Статическая типизация

- Целостность данных
- Высокая производительность
- Экономия памяти

Реализации

- PostgreSQL: Статическая
- MySQL: Статическая
- MySQL NDB-cluster: Статическая
- Oracle: Статическая
- Oracle Timesten: Статическая
- Riak: Динамическая

Реализации

- MongoDB: Динамическая
- Hadoop: Динамическая
- Mnesia: Динамическая
- memcached: Динамическая
- memcachedb: Динамическая
- berkleydb: Динамическая
- Voldemort: Динамическая
- Cassandra: Динамическая

Cost/license

- PostgreSQL: BSD-like
- MySQL: GPL 2.0/just call
- MySQL NDB-cluster: GPL 2.0/just call
- Oracle: $47,500
- Oracle Timesten: $41,500
- Riak: Apache 2.0
- MongoDB: AGPL 3.0

Cost/license

- Hadoop: Apache 2.0
- Mnesia: Erlang Public License
- memcached: BSD
- memcachedb: BSD
- berkleydb: Sleepycat License/$9,800
- Voldemort: Apache 2.0
- Cassandra: Apache 2.0

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/