Проектирование ПО

Лекция 15: Google Case Study

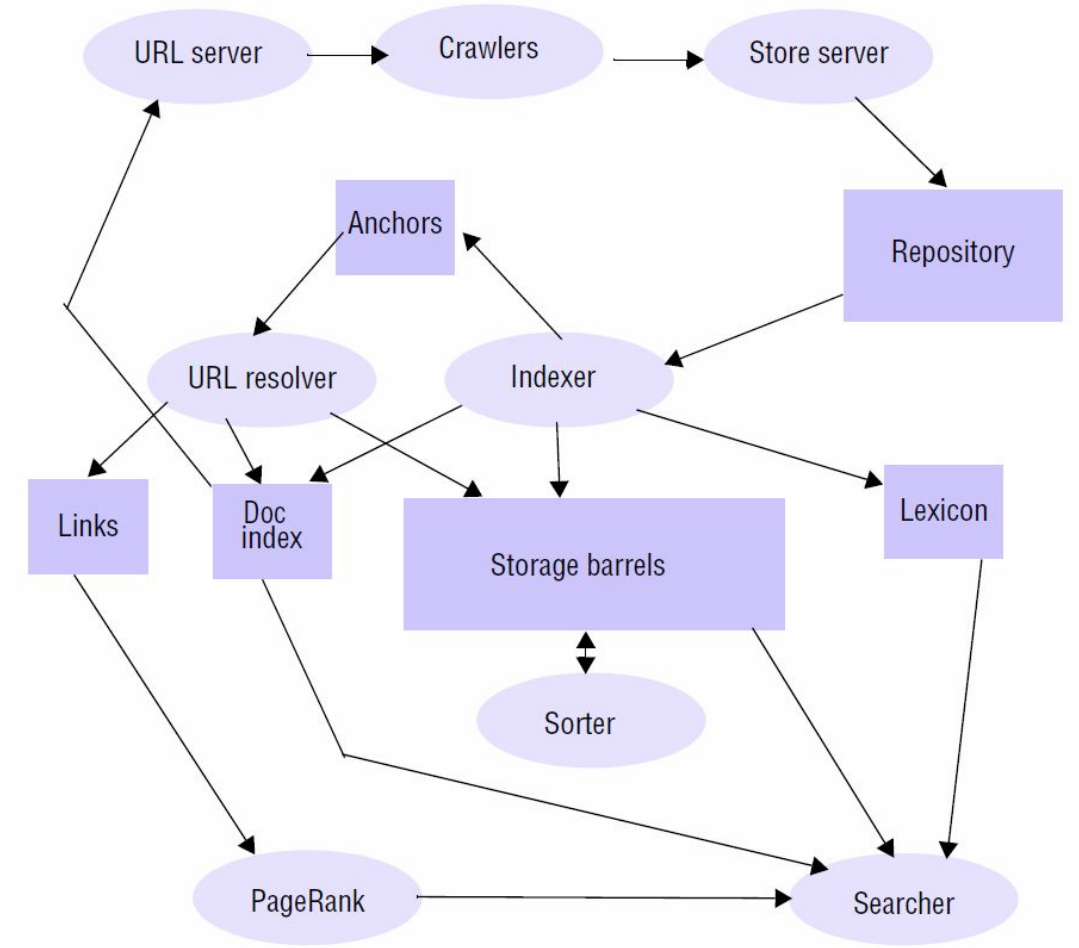
Тимофей Брыксин
timofey.bryksin@gmail.com

Google

- поиск
- SaaS
  - Gmail
  - Google Docs
  - Google Calendar
  - ...
- PaaS
  - Google App Engine

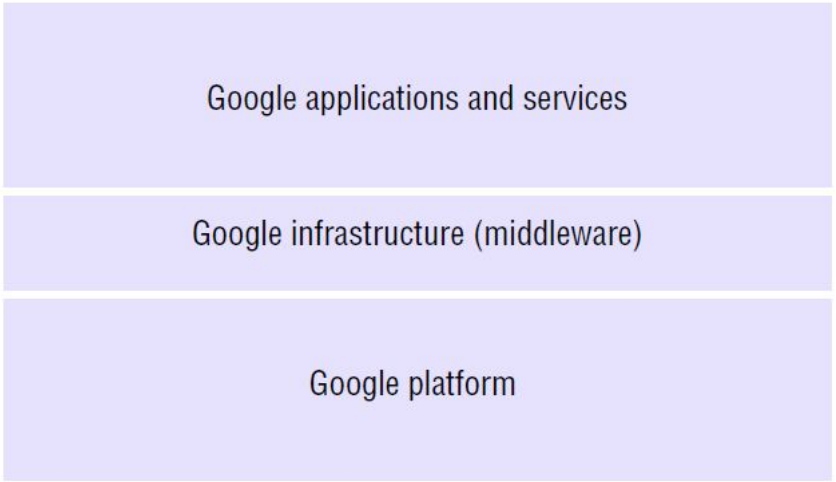
Поиск

- поисковые роботы
- построение индекса
- ранжирование



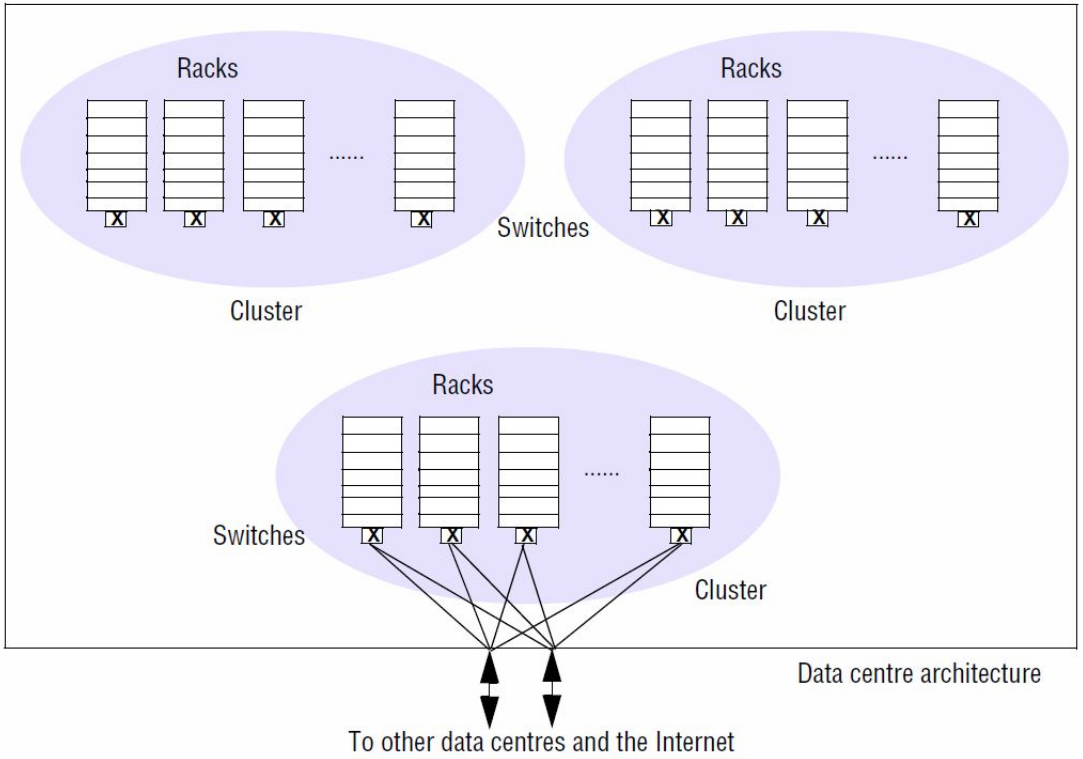
Требования к системе

- масштабируемость
  - больше данных
  - больше запросов в минуту
  - лучше результаты запросов
- надёжность
  - поиск
  - сервисы
- производительность
  - меньше задержка
  - больше результатов
  - совокупный эффект
- открытость

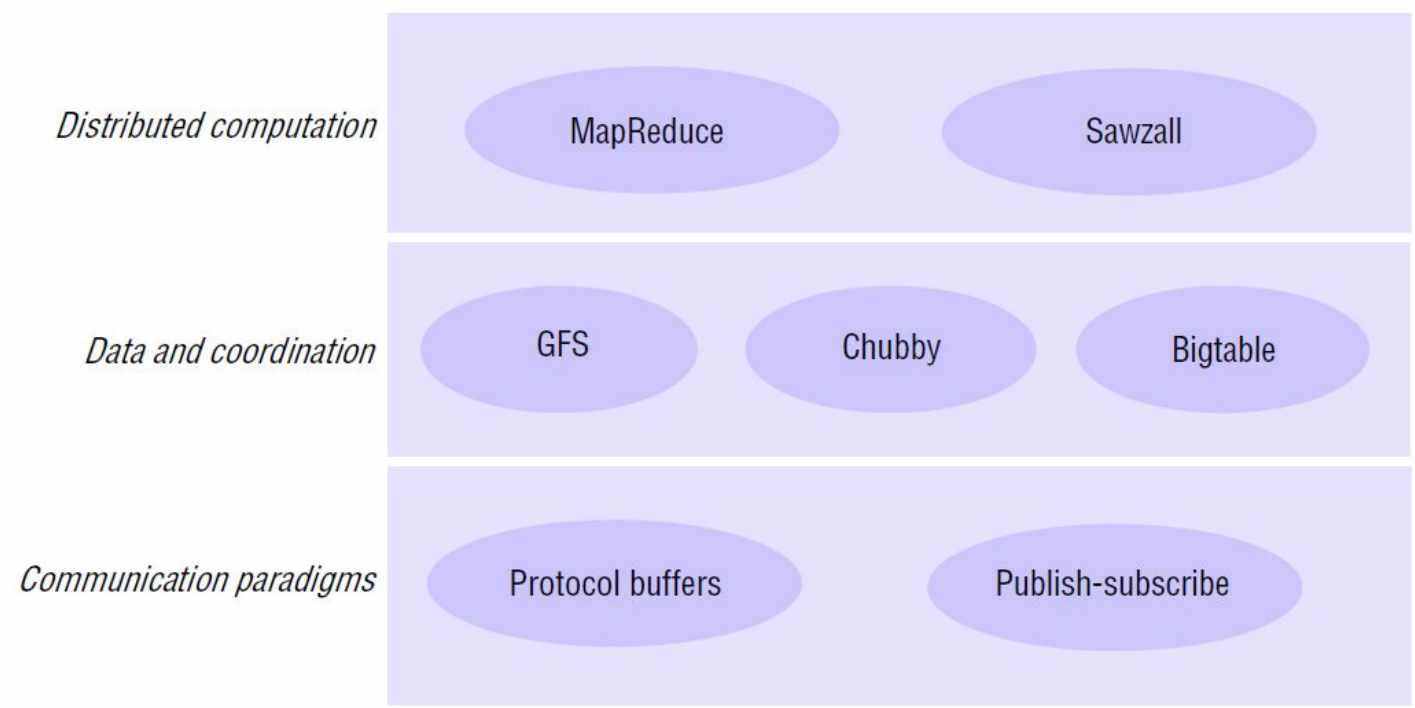


Физическая модель

- “обычные” компьютеры
  - linux
- стойки
  - 40-80 компьютеров
  - 1Gbps ethernet свитч
- кластеры
  - ~30 стоек
  - два свитча

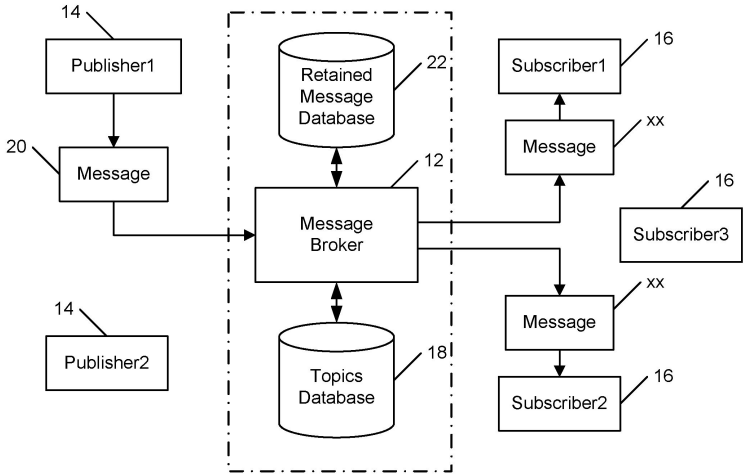


Инфраструктура



Коммуникационный слой

- protobuf + gRPC
  - один параметр запроса, один ответ
  - бинарный формат
  - HTTP/2
- publish-subscribe (Google Cloud Pub/Sub?)
  - легковесный механизм
  - topic-based каналы + фильтрация
  - иерархия брокеров
  - надёжность доставки
    - дублирование брокеров
    - QoS

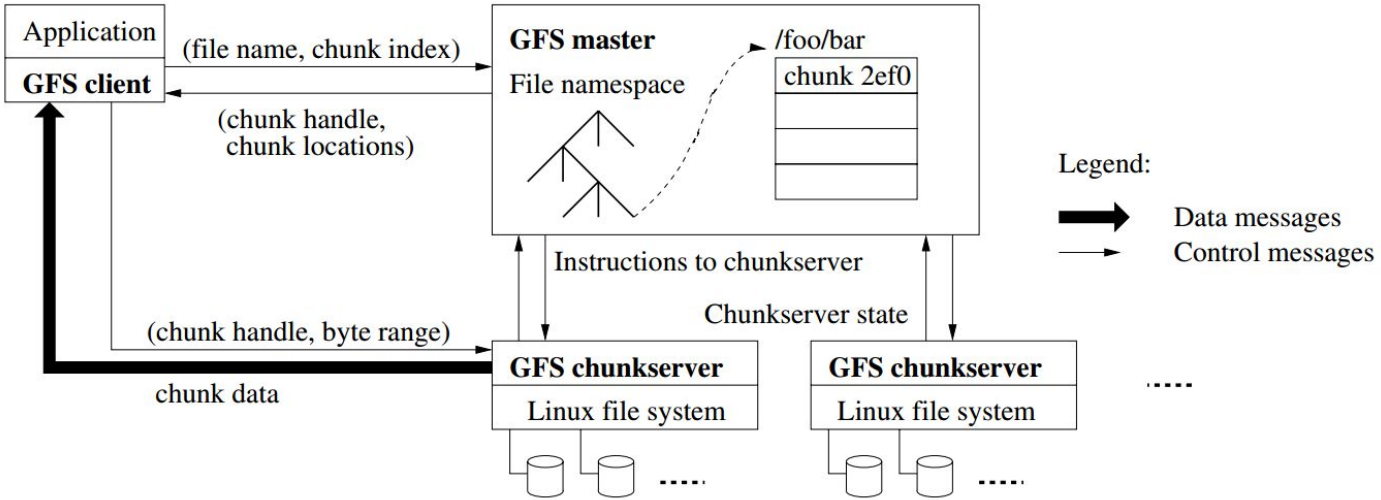


Google File System (v1)

- требования
  - надёжная работа на описанной физической архитектуре
  - эффективная работа с большими файлами
  - эффективная работа с последовательностями операций записи или чтения
  - файлы часто дописываются в конец, но редко модифицируются
  - поддержка конкурентного доступа
  - масштабируемость, надёжность, производительность, …
- иерархичная структура
- идентификация файла путём
- POSIX-like API
  - create, delete, open, close, read, write
  - snapshop, record append

GFS: особенности реализации

- сегменты (chunks) по 64Mb
- репликация сегментов (на 3 сервера)
- разделение управления и работы с данными
- отсутствие кэширования на серверах



GFS: метаданные

- менее 64 байтов метаданных на фрагмент
- хранятся в ОП мастера
  - пространства имён файлов и сегментов
  - отображение файлов на сегменты
  - расположение сегментов на серверах
- лог операций
  - на диске мастера + репликация
- опрос серверов фрагментов
  - HeartBeat сообщения
- фоновые процессы на мастере
  - сборка мусора
  - перераспределение фрагментов

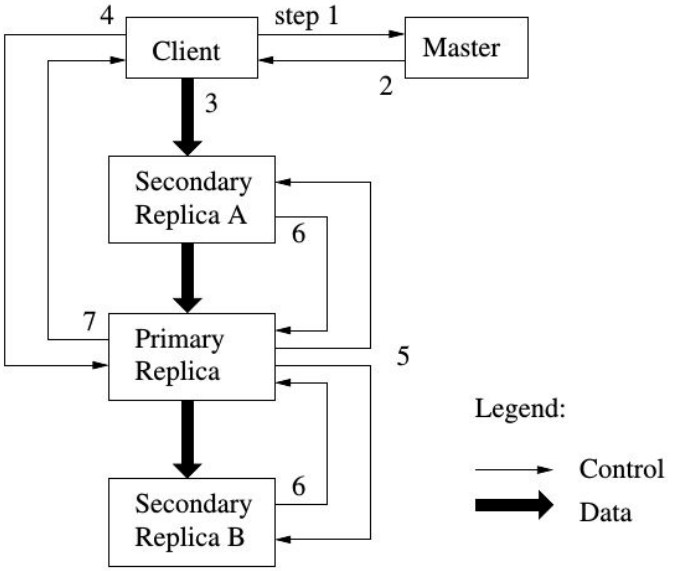
GFS: поддержка целостности

- операции над пространством имён файлов делает мастер
  - целостность обеспечивается логом операций
- состояния consistent и defined для содержимого файлов
- атомарный record append с at-least-once семантикой
- целостность гарантируется
  - одинаковым порядком применения операций на всех репликах
  - номерами версий фрагментов
- кэширование vs устаревшие данные
- некорркетные данные
  - контрольные суммы
  - явные ошибки клиентам
- ограничения на клиентов

	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

GFS: вариант пассивной репликации

- назначение основного репликатора
  - 60 сек. + продления
  - определяет линейный порядок операций
- передача данных всем репликаторам
- подтверждение получения, запрос на запись
  - определение основным репликатором порядка, применение изменений
- применение изменений бэкапами
  - в том же порядке
- получение основным подтверждений
  - отказ при отсутствии хотя бы одного подтверждения
  - повтор изменений (шаги 3-7)



GFS: итоги

- большой размер сегментов
  - эффективно для больших файлов, минимизирует метаданные
    - очень неэффективно для произвольного доступа небольшими участками
- централизованный мастер
  - мастер -- единая точка управления, проще реализация
    - единая точка отказа (с учётом репликации логов не так плохо)
- разделение потоков управления и данных
  - минимальное вовлечение мастера, эффективный обмен данными
    - усложнение логики клиента
- упрощённая модель согласованности
  - высокая эффективность, спроектированная с учётом специфики операций GFS
    - несогласованные данные, потенциально дублированные

Chubby

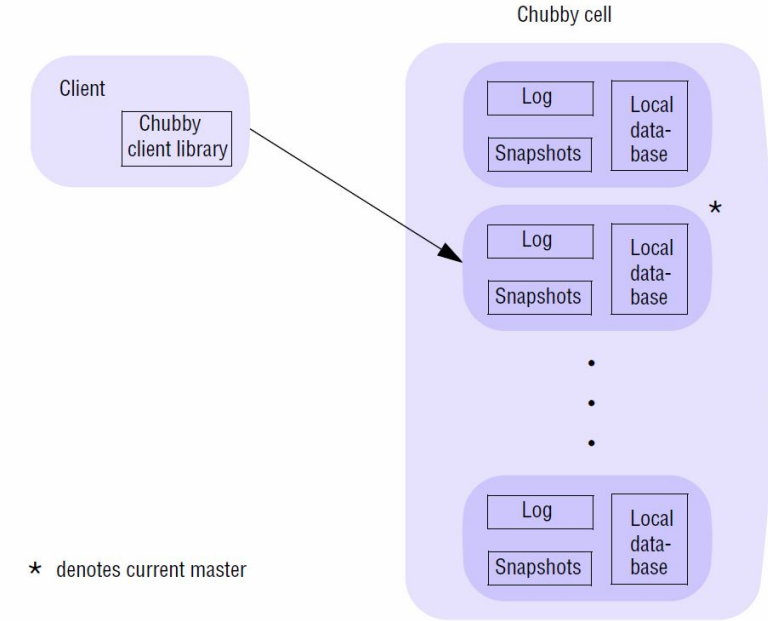
- coarse-grained распределённые блокировки
- ФС для небольших файлов
- механизм голосования для выбора основных репликаторов
- внутренний DNS

Chubby: интерфейс

- абстракция файловой системы
  - всё файлы
  - иерархическая структура
  - reader-writer блокировки
- /ls/cell/directory_name/.../file_name
  - /ls/local
- API
  - общее: Open, Close, Delete
  - файл: GetContentsAndStat, GetStat, ReadDir, SetContents, SetACL
  - блокировка: Acquire, TryAquire, Release
- события через callback’и
  - изменение содержимого, изменения в иерархии, изменения в блокировках и т.п.

Chubby: архитектура

- ячейка: выбираемый мастер + 4 бэкапа (в разных местах кластера)
  - поддержка нескольких десятков тысяч компьютеров
  - RPC
- целостность БД через Paxos (logs + snapshots)
  - репликация всей БД в GFS каждые несколько часов
- выбор нового репликатора
  - обновление DNS
  - обновление мастера
  - обновление списка серверов ячейки
  - получение новым репликатором данных
  - голосование на нового мастера



Chubby: взаимодействие с клиентами

- клиентская сессия
  - lease, продления, таймауты и т.п.
  - KeepAlive сообщения
  - grace period
- кэширование на клиенте
  - данные и метаданные
  - небольшие файлы, повторные запросы
  - аннулирование кэша клиентов сервером при запросах на обновление
    - поверх KeepAlive сообщений
    - ожидание нотификаций от всех перед изменениями

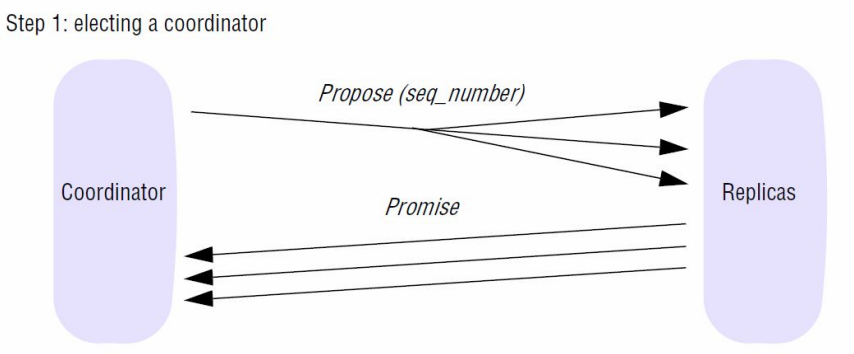
Chubby: Paxos

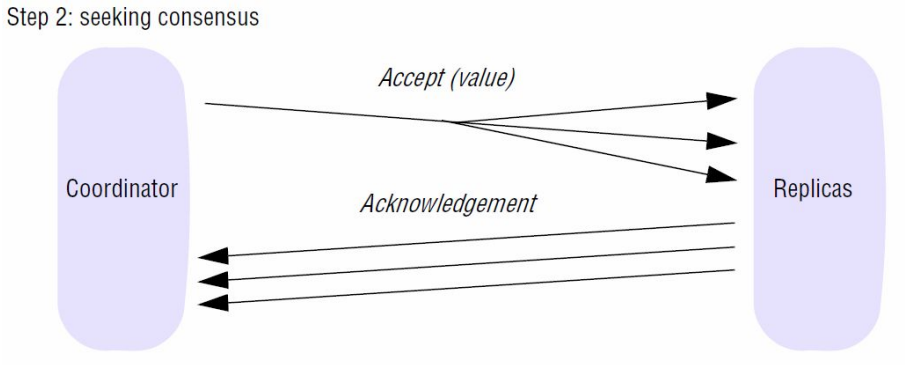
- семейство протоколов консенсуса для асинхронных систем
  - репликаторы работают каждый со своей скоростью
  - репликаторы могут исчезать и появляться
  - у репликаторов есть доступ к надёжному хранилищу данных
  - сообщения могут быть потеряны, задержаны, переупорядочены или дублировны
    - но если доставлены, то полностью корректны
- гарантирует корректность, но не живучесть

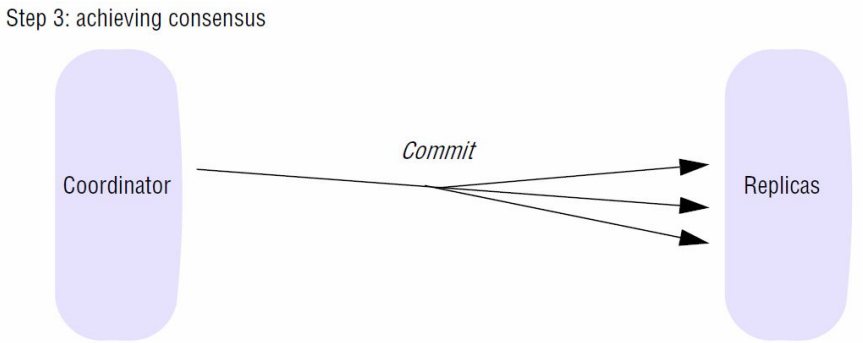
Paxos-L1 (Progress): If there exists a stable majority set of servers, then if a server in the set initiates an update, some member of the set eventually executes the update.

Paxos-L2 (Eventual Replication): If server s executes an update and there exists a set of servers containing s and r, and a time after which the set does not experience any communication or process failures, then r eventually executes the update.

Chubby: голосование в Paxos





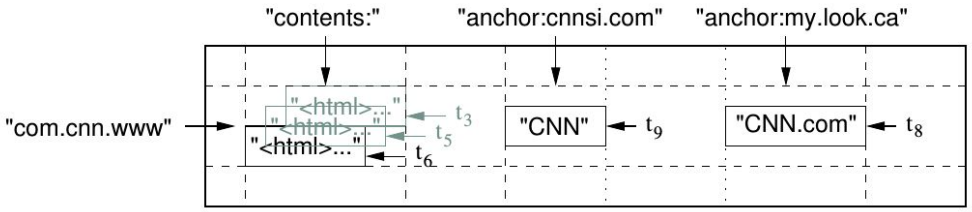


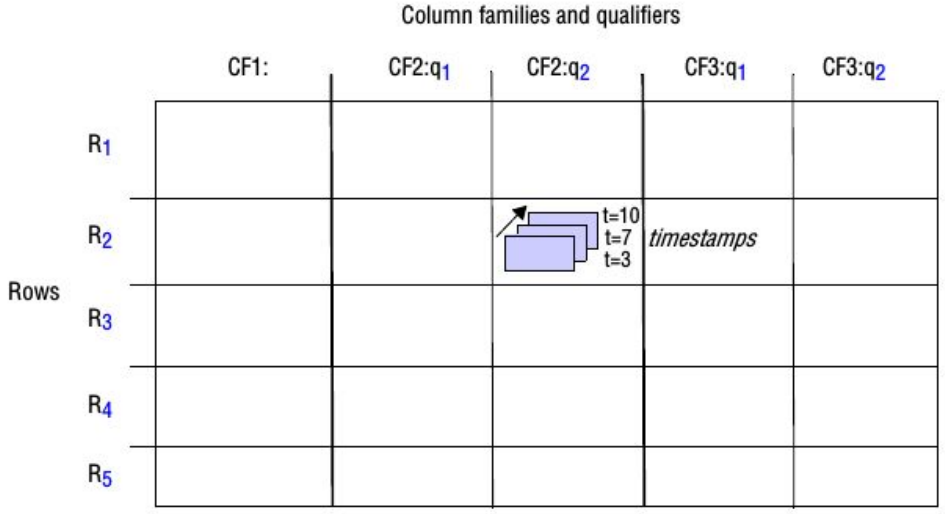
Chubby: итоги

- комбинация абстракций файлов и блокировок
  - многоцелевое использование
    - нужно чёткое понимание аспектов работы
- работа с файлами целиком
  - очень эффективно с малыми файлами
    - неэффективно с большими
- кэширование на клиенте со строгой согласованностью
  - детерминированное поведение
    - накладные расходы на поддержку согласованности

BigTable

- потенциально неограниченные структурированные данные
  - таблицы в десятки и сотни терабайт
  - табличная (не реляционная) модель
- трёхмерная структура
  - строка
    - до 64кб под имя
    - лексикографическая сортировка
  - столбец
    - группировка по типу
  - временная метка
    - время или версия
    - обратная сортировка
    - сборка мусора



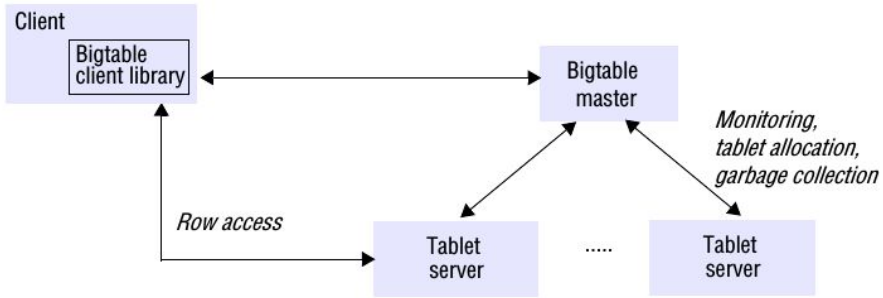


BigTable: подробности

- операции
  - создание и удаление таблиц, групп столбцов внутри таблиц
  - доступ к данным строк, изменение данных ячеек
    - атомарные операции над строками
    - транзакции между строками не разрешаются
  - итерирование по группам столбцов, включая регэкспы
  - управление метаданными и контроль доступа
- таблица разбивается на фрагменты (tablets) по 100-200Mb
- размещение фрагментов в GFS
  - эффективная балансировка
  - работа на тех же компьютерах, что и другие сервисы

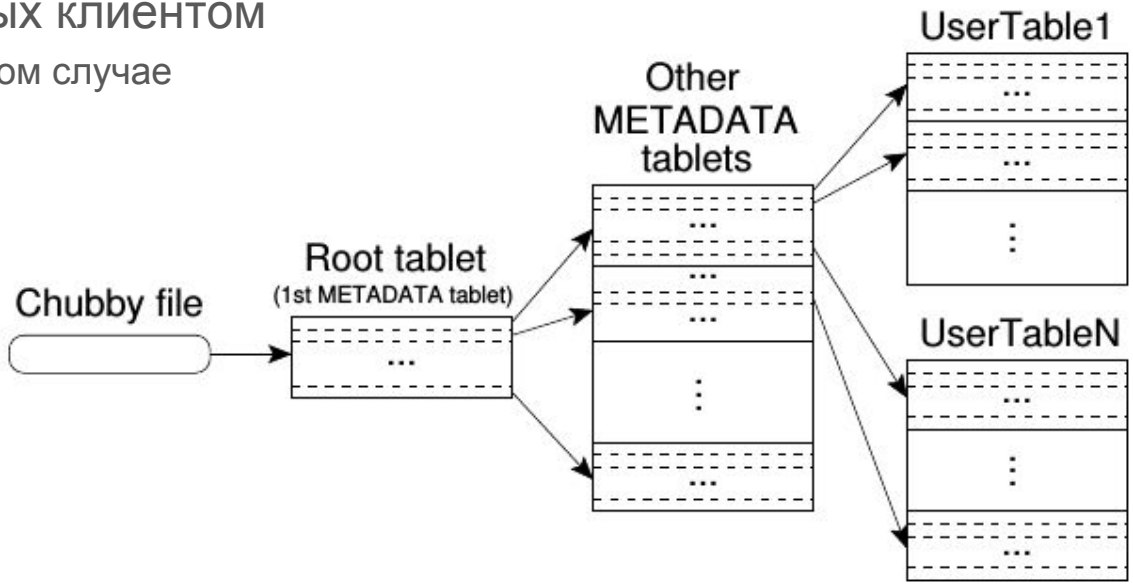
BigTable: архитектура

- кластер -- от десятков до сотен серверов
  - в среднем до тысячи фрагментов на кластер
  - динамическое управление
- система управления кластером
- единый мастер и куча реплик
  - разделение ответственности за логику и данные
  - мастер не отвечает за размещения фрагментов на физическую структуру GFS
    - многие клиенты вообще могут никогда не говорить с мастером



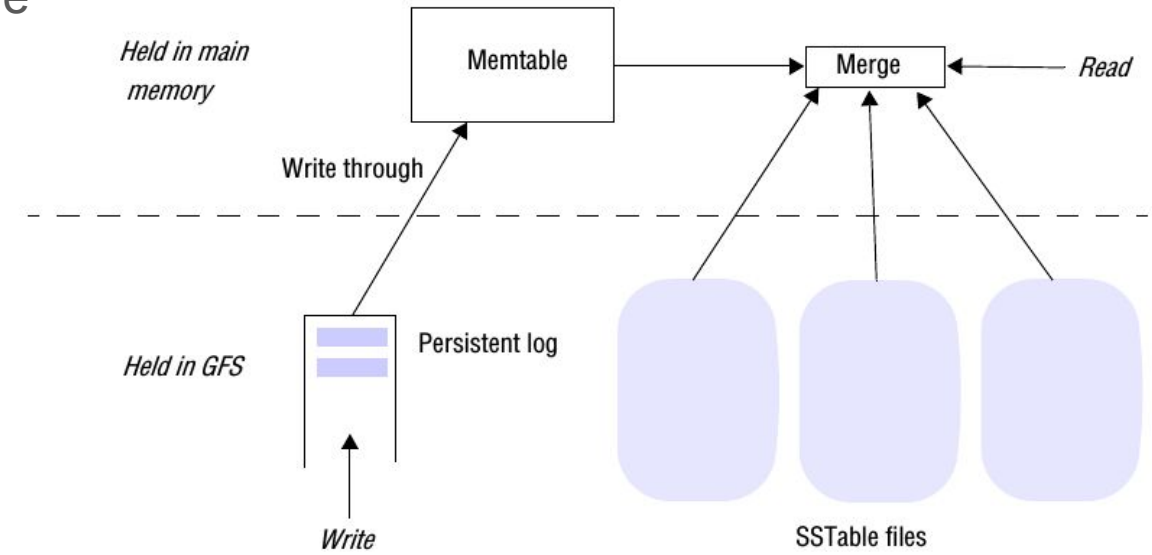
BigTable: схема индексации

- распределённая таблица метаданных
  - корневой фрагмент + фрагменты с метаданными
  - не более трёх уровней
- кэширование метаданных клиентом
  - 6 операций в самом плохом случае



BigTable: хранение фрагментов

- фрагменты как набор файлов в формате SSTable
  - snapshot фрагмента в виде упорядоченного неизменяемого ассоциативного массива
- индекс на основе B+ дерева
- лог операций в memtable
  - периодический сброс содержимого в SSTable
- сжатие SSTable
- фоновое уплотнение и слияние SSTable



BigTable: мониторинг

- директория в Chubby для мониторинга серверов
- блокировка как индикатор работы сервера
  - сервер создаёт файл и захватывает на него блокировку
    - при завершении снимает блокировку
    - если теряет блокировку, пытается переполучить заново
  - мастер узнаёт статус сервера, пытается получить его блокировку
    - если получает, удаляет файл, перераспределяет фрагменты
    - сервер (если живой) видит, что его файл удалили, и убивается
- мастер-блокировка в Chubby
  - потеря блокировки -> перезапуск мастера
  - гарантия единственного мастера в кластере
- мастер имеет список актуальных серверов
  - хранит и поддерживает распределение фрагментов по серверам

BigTable: немного статистики (2006)

- 388 кластеров, 24500 серверов фрагментов
- группа из 14 кластеров (8069 серверов): 1.2 млн запросов в секунду, 741 мбпс исходящего и 16 гбпс входящего RPC трафика

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

BigTable: итоги

- абстракция таблиц
  - эффективная поддержка структурированных данных
    - менее выразительный механизм, чем РСУБД
- централизованный мастер
  - мастер -- единая точка управления, проще реализация
    - единая точка отказа, потенциальный bottleneck
- разделение потоков управления и данных
  - разгрузка мастера, быстрый обмен данными
    - усложнение логики клиентов
- упор на мониторинге и балансировке нагрузки
  - поддержка большого количества параллельных клиентов
    - накладные расходы на поддержку глобальных состояний

MapReduce

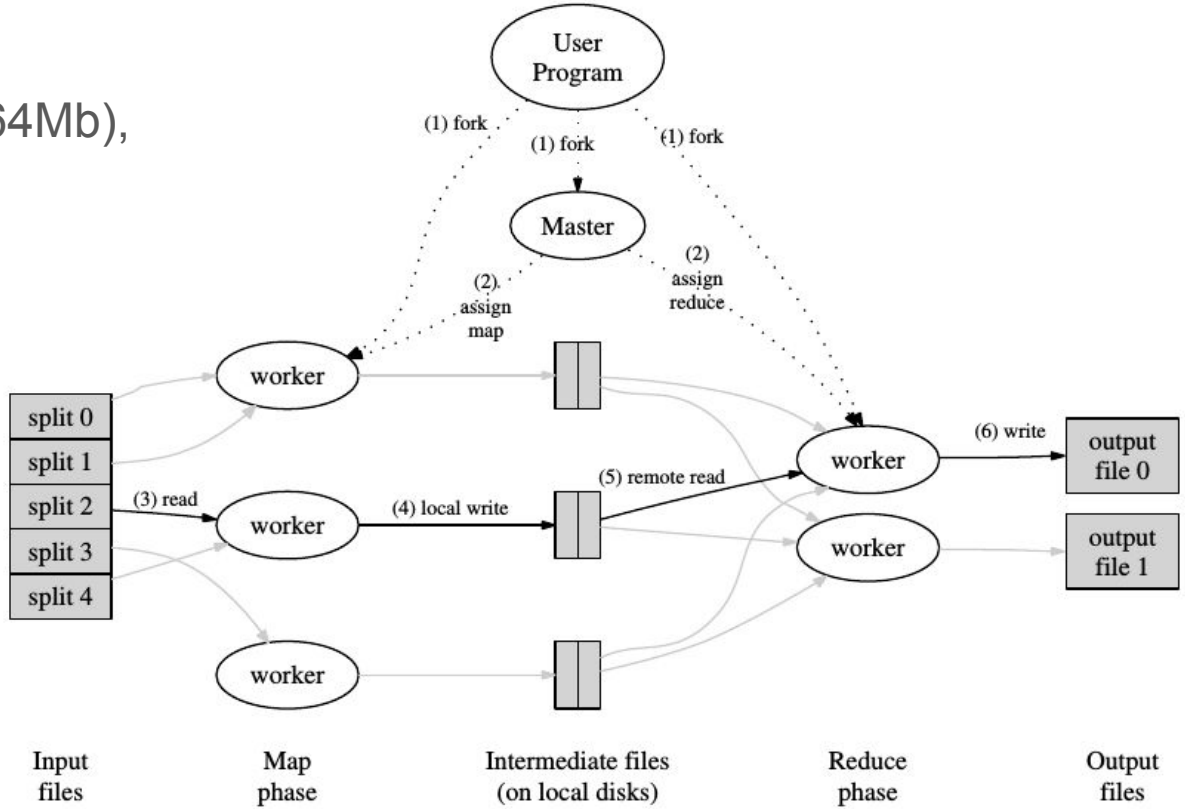
- типовой шаблон обработки данных
  - разбиение данных на небольшие фрагменты
  - параллельная обработка фрагментов
  - объединение промежуточных результатов
- поиск, сортировка, построение обратного индекса и т.п.
- работа с парами ключ-значение
- от программиста требуется:
  - map(): (k1, v1) -> list(k2, v2)
  - reduce(): (k2, list(v2)) -> list(v2)
- реализация обеспечивает
  - разбиение данных на фрагменты для обработки
  - параллельный распределённый запуск операций
  - мониторинг исполнения, балансировка нагрузки, управление результатами

MapReduce: примеры

Function	*Initial step*	*Map phase*	*Intermediate step*	*Reduce phase*
Word count	*Partition data into fixed-size chunks for processing*	For each occurrence of word in data partition, emit <*word*, 1>	*Merge/sort all key-value keys according to their intermediary key*	For each word in the intermediary set, count the number of 1s
Grep		Output a line if it matches a given pattern		Null
Sort *N.B. This relies heavily on the intermediate step*		For each entry in the input data, output the key-value pairs to be sorted		Null
Inverted index		Parse the associated documents and output a <*word*, document ID> pair wherever that word exists		For each word, produce a list of (sorted) document IDs

MapReduce: архитектура

- RPC + BigTable + GFS
- M фрагментов (по 16-64Mb), R значений ключей
- узел-мастер
  - распределение задач, мониторинг, координация результатов
- обработка локально на узлах с данными



MapReduce: устойчивость

- гарантия повторяемости при детерминированных map и reduce
  - даже при условии отказов
- мониторинг доступности узлов мастером (ping)
  - если узел делал map, задача перепланируется всегда
  - если узел делал reduce, задача перепланируется, если была не доделана
- сохранение локаций промежуточных результатов мастером
- запуск дублирующих узлов при долгом выполнении

Пример: Google Maps (2008)

- несколько уровней тайловых карт
  - некоторые меняются часто, некоторые нет
- набор исходных данных разных видов и форматов
  - KML как единый формат метаданных
- всё в одной таблице
  - размером в 70 Tb и 9 млрд ячеек
  - строки обеспечивают локальность объектов
- Map: преобразование данных об объектах в плоскую структуру тайлов
- Reduce: рендеринг тайлов в растровые изображения
- 8 часов на генерацию карты, генерация 1 Mb данных в секунду
- хранение в GFS, индекс в 500 гб (большей частью в памяти)

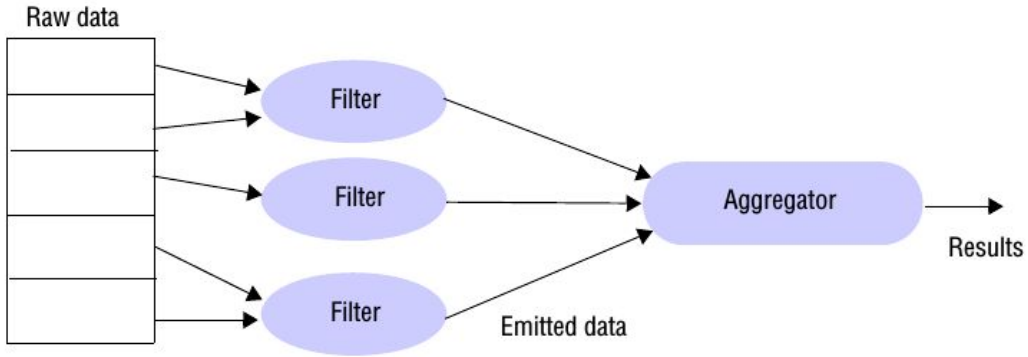
MapReduce: итоги

- использование кастомного фреймворка
  - детали спрятаны от программиста
    - модель может не подойти для всех задач
- программирование через map и reduce
  - простая модель, быстрая разработка
    - модель может не подойти для всех задач
- прозрачная поддержка устойчивости к отказам
  - снимает ответственность с программиста
    - накладные расходы на механизмы восстановления

Sawzall

- интерпретируемый язык для параллельного анализа данных
  - задачи, удовлетворяющие заданному шаблону
  - в 10-20 раз меньше кода, чем MapReduce
- MapReduce для запуска задач, GFS для хранения результатов
- фильтры и агрегаторы коммутативны
- агрегаторы ассоциативны

```
count: table sum of int;
total: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
```



Sawzall: итоги

- специализированный DSL для распределённых вычислений
  - быстрая разработка, детали спрятаны от программиста
    - подходит не для всех задач

А так всё на самом деле :)