The Google File System
The Chubby lock service for loosely-coupled distributed systems
Paxos Made Live - An Engineering Perspective
Bigtable: A Distributed Storage System for Structured Data
MapReduce: Simplified Data Processing on Large Clusters

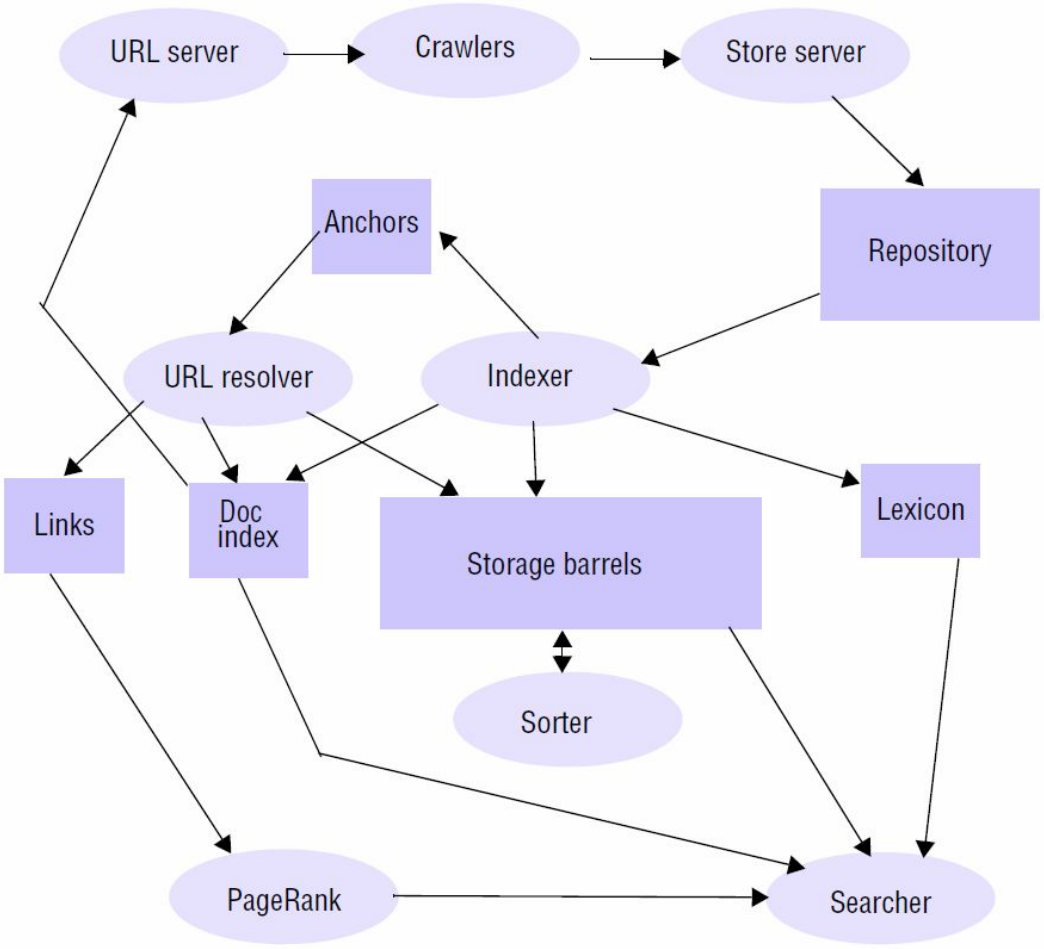
The Google Stack

Инфраструктура Google:

1. Поиск
2. SaaS -- предоставляемые сервисы (Gmail, Docs, Calendar)
3. PaaS -- предоставляемые платформы для разработки (Google App Engine)

Поиск:

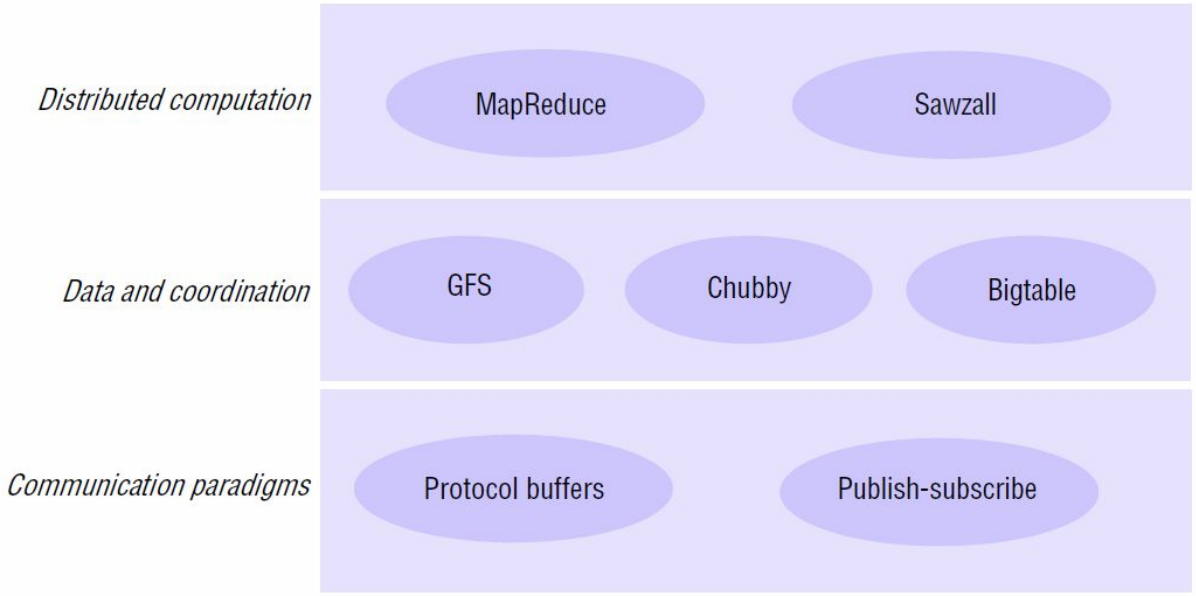
1. Поисковые роботы (Crawlers) -- ползают по интернету, сохраняют страницы в репозиторий
2. Построение индекса (Indexer) -- строит обратный индекс по словам страницы (по словам получаем страницы) и извлекает дополнительную информацию для ранжирования (статистику про слова -- в рамках каких тегов/заголовков/подписях к картинкам/… встречались и как часто)
3. Ранжирование (PageRank) -- использует информацию о ссылках (чем больше, тем лучше) и информацию из indexer'а, по которой ранжирует страницу
4. Lexicon хранит набор слов, которые можно распознавать (например, чтобы исправлять опечатки в поисковом запросе)
5. Кружки на диаграмме -- действия (процессы), квадраты -- данные



Все это накладывает определенные требования к системе:

- масштабируемость
  - обрабатывать больше данных
  - обрабатывать больше запросов в минуту
  - лучше результаты запросов (по релевантности)
- надёжность
  - поиск (актуальность результатов)
  - сервисы (есть соглашение для всех клиентов про доступность системы, например, сервисы работают 98% времени)
- производительность
  - меньше задержка
  - больше результатов
  - совокупный эффект (количество памяти, качество сети, ...)
- открытость (API некоторых сервисов)

Физическая модель: используются обычные (“домашние”) компьютеры; в стойке 40-80 компьютеров, в кластере 30 стоек; на кластер где-то два свитча (это такой условный маршрутизатор, который определяет адреса компьютеров в кластере). Отказывает в год 2-5% компьютеров. Программных отказов больше, чем аппаратных.



Инфраструктура (на диаграмме выше):

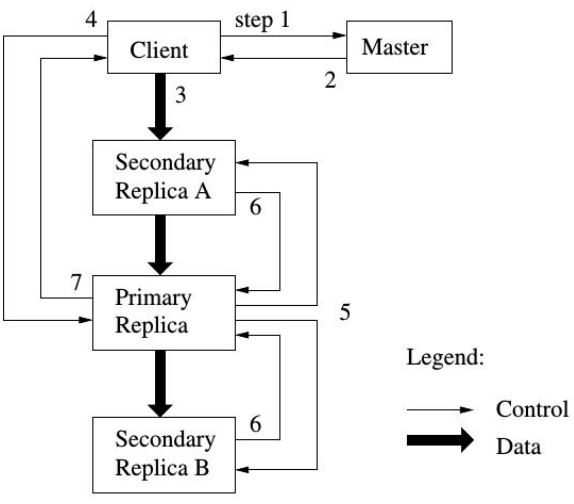
1. Распределенные вычисления (способ сделать вычисления)
2. Данные и координация
    a. GFS (Google File System):
        i. Требования:
            1. совместимость с физической архитектурой
            2. эффективная работа с большими файлами (их очень много)
            3. эффективная работа с последовательными операциями чтения и записи (их тоже очень много)
            4. файлы часто записываются в конец, но редко модифицируются
            5. поддержка конкурентного доступа
            6. масштабируемость, надёжность, производительность
            7. замечание: обычные распределенные файловые системы работают с обычными файлами произвольных размеров, чаще всего маленькими, у них поддерживается эффективный рандомный доступ для записи/чтения маленькими кусочками из центра, тут такого не будет, потому что у нас файлы специфичные
        ii. Иерархическая структура
        iii. Идентификация файла путем
        iv. POSIX-like API
            1. create, delete, open, open, close, read, write -- как в POSIX
            2. нет функции перемещения файла между каталогами -- не как в POSIX
            3. record append -- позволяет дописывать в конец эффективно (новая опция)
            4. snapshot -- создаёт эффективный снимок того или иного файла (новая опция)

v. Особенности реализации:
   1. сегменты (они же chunks, они же фрагменты) по 64МБ (для сравнения, в Linux по 4КБ)
   2. есть один мастер-сервер и куча серверов-репликаторов
   3. обычно несколько тысяч chunkserver на один мастер
   4. каждый chuck реплицируется на три разных сервера по умолчанию (чаще внутри кластера, но на разных стойках)
   5. разделение управления и работы с данными (управляющая информация идет через мастера, он отвечает за операции создания и т.д., но данные через него не ходят -- клиент напрямую запрашивает данные у одного из трех chuck-серверов)

vi. Метаданные:
   1. менее 64 байтов на сегмент
   2. хранятся на мастере (пространства имен файлов и сегментов, отображение файлов на сегменты, расположение сегментов на серверах) -- на мастере хранятся *только* метаданные
   3. все операции, которые происходят с сегментами, сохраняются в лог операций; он хранится на диске мастера, когда становится большим, делается его snapshot и реплицируется на другие устройства
   4. опрос серверов фрагментов (живы ли)
   5. при удалении помечаем как hidden, раз в три дня сборка мусора
   6. в фоновом режиме перераспределение сегментов для балансировки

vii. Поддержка целостности:
   1. для содержимого файлов состояния consistent (реплики совпадают, могут быть дубликаты данных) и defined (порядок данных совпадает)
   2. timeout'ы для неактуальной (неполная, т.к. добавляем в конец файла) информации
   3. есть контрольные суммы, которые сверяются

viii. Пассивная репликация:
   1. репликаторы назначаются на какое-то время, потом продление лицензии
   2. клиент посылает данные ближайшему репликатору, он пересыдает другим
   3. основной репликатор (один из трех) контролирует процесс -- строит порядок выполнения операций



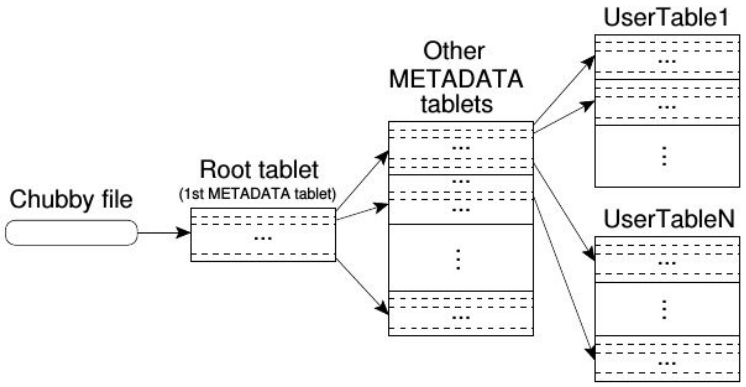
ix. Итоги:
    1. Большие сегменты (хорошо для больших файлов, минимизирует метаданные; плохо для random access)
    2. централизованный мастер (единая точка управления, она же и отказа, но есть реплики)
    3. разделение потоков управления и данных (не очень нужен мастер, эффективно с данными, сложная логика для клиента)
    4. простая модель согласованности (эффективно, возможно дублирование)

b. Chubby (система распределенных блокировок) -- ФС маленьких файлов-блокировок:
    i. Механизм голосования для выбора основных репликаторов
    ii. Внутренний DNS
    iii. Интерфейс:
        1. Абстракция ФС (иерархическая структура, reader-writer блокировки)
        2. Глобальные уникальные пути
        3. API (общее: Open, Close, Delete; файл: GetContentsAndStat, GetStat, ReadDir, SetContents, SetACL; блокировка: Acquire, TryAquire, Release)
    iv. Архитектура: ячейка = мастер + 4 бэкапа (на каждом сервере БД: namespaces, файлы, блокировки, ...)
    v. Paxos -- протокол консенсуса для операционных систем (у репликаторов разная скорость, они иногда исчезают/появляются, сообщения могут быть перемешаны, но если доставлены, то корректны): гарантирует корректность, но не завершаемость
    vi. Итоги:
        1. Комбинация абстракций файлов и блокировок
        2. Работа с файлами целиком
        3. Строгое кеширование на клиенте

c. Bigtable -- огромные трехмерные таблицы (нереляционная распределенная БД):
    i. Строка -- до 64КБ под имя (URL), упорядочены лексикографически; операции атомарны; между строками транзакций нет
    ii. Столбец -- группировка по типу; итерирование по группам
    iii. Временная метка -- время/версия (их немного из-за сборки мусора)
    iv. Разбивается на фрагменты (tablets) по 100-200MB, они размещаются в GFS
    v. Архитектура:
        1. Кластер = ~10-100 серверов
        2. До 1000 фрагментов на кластер

3. Система управления кластером
4. Один мастер, много реплик

vi. Индексация (не более трех уровней)



1.

vii. Хранение фрагментов:
1. Фрагмент = набор файлов в формате SSTable
2. Все операции в логе
3. B+-дерево индекс, фоновое уплотнение и слияние

viii. Мониторинг: блокировка как индикатор работы сервера

ix. Итоги:
1. абстракция таблиц (эффективно для структурированных данных)
2. централизованный мастер
3. разделение потоков управления и данных
4. мониторинг и балансировка нагрузки

3. Коммуникационные парадигмы (способ пересылки сообщений)
a. Protobuf (бинарный протокол сериализации данных) + gRPC = компактно, быстро, кроссплатформенно (использует HTTP/2)
b. Publish-subscribe (надежный канал массовой рассылки сообщений, фильтрация)