Distributed systems
Cassandra

History

July 2008 — open-sourced by Facebook[1]
March 2010 — graduated from the Apache Incubator
Influenced by Amazon Dynamo
Committers: Rackspace, Digg, Twitter, Amazon, Microsoft

[1]Facebook. Cassandra — A Decentralized Structured Storage System: http://www.cs.cornell.edu/projects/ladis2009/papers/ lakshman-ladis2009.pdf

Features

- Decentralized — no SPoF, every node is identical
- Multi data center replication
- Elastic Scalability
- High Availability and Fault Tolerance
- Tunable consistency
- CQL

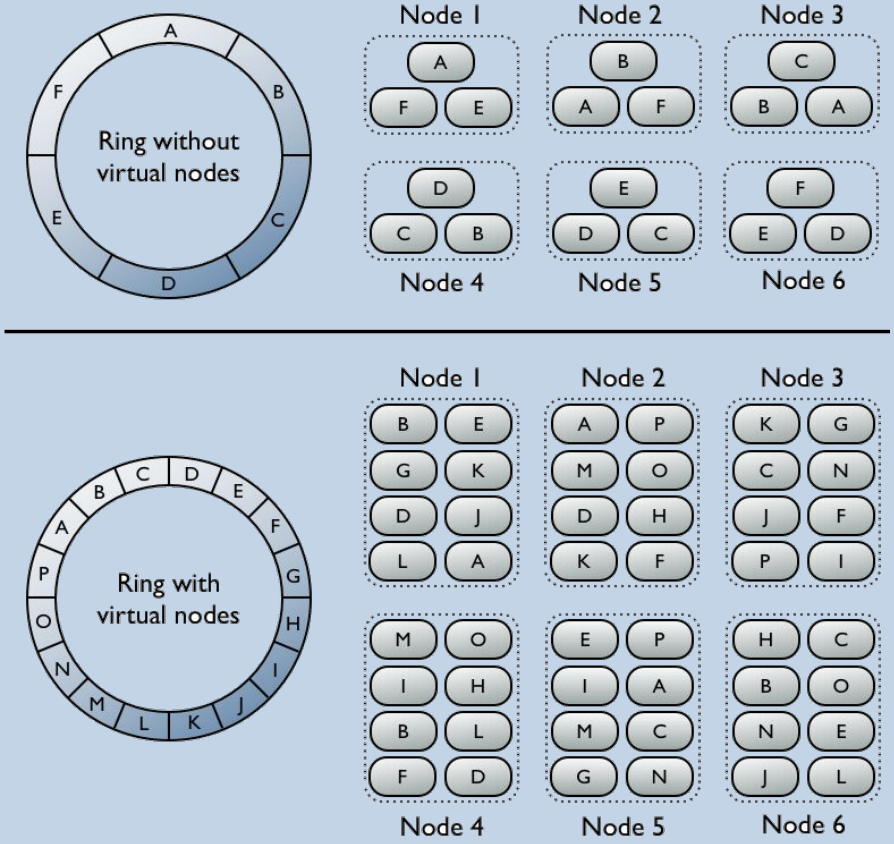
ACID perspective

- Никаких JOIN-ов и внешних ключей
- *Атомарность* на уровне столбцов, нет rollback*
- Можно получить ошибку при записи, но запись состоится
- Отметки времени от клиента при конфликтах
  Настраиваемая *консистентность* (количество реплик)
- *Изоляция* на уровне столбцов
- *Durability* : commit log + replication

Components

Consistent Hashing + **VNodes** (см. лекцию 1)
Gossip — состав и состояние узлов
Partitioner — данные по узлам
Replica placement strategy — реплики по узлам
Snitch — топология (ДЦ и стойки)

Virtual Nodes



Gossip

- Узлы периодически обмениваются информацией о себе и других узлах
- Каждую секунду не больше чем с тремя узлами
- Все узлы быстро узнают информацию друг о друге
- Сообщения имеют версию — устаревшая информация затирается
- Seed nodes во избежания партиционирования
- Gossip-информация хранится на каждой ноде персистентно

Partitioner

- `Murmur3Partitioner (default)`
- `RandomPartitioner (md5)`
- `ByteOrderedPartitioner (specific)`
  - Лексикографически по байтам ключа
  - Сложная балансировка нагрузки
    (расчёт диапазонов партиций вручную)
  - Неравномерная балансировка для разных таблиц
  - Последовательная запись упирается в один узел

Replica Placement Strategy

Replication Factor — количество реплик на кластер

- *SimpleStrategy:*
  - 1 ДЦ (если планируете расширяться, не используйте)
  - Первая реплика на узел от Partitioner
  - Остальные — по часовой стрелке по кольцу
- *NetworkTopologyStrategy*:
  - Определяет количество реплик в каждом ДЦ
  - Узлы для реплик — идём по кольцу до следующей стойки

  Часто стойки вырубаются целиком (питание, охлаждение, сеть, ...)

Snitch

Используется для анализа топологии кластера:
в каком дц, в какой стойке находится нода

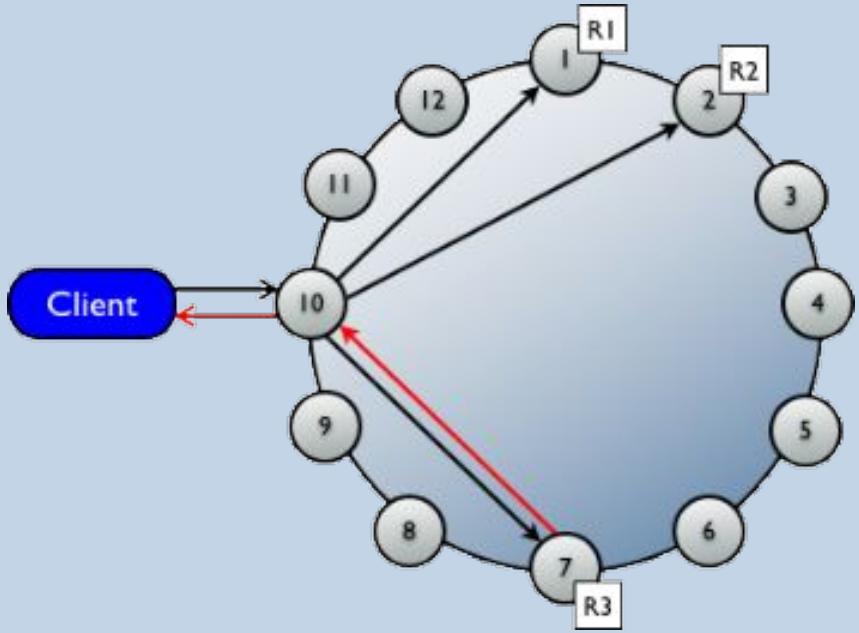
- Dynamic snitching
- SimpleSnitch
- RackInferringSnitch
- PropertyFileSnitch

Клиентские запросы

- Можно обращаться к любому узлу
- Узел становится координатором для текущего запроса
- Координатор проксирует с учётом Partitioner и Replica Placement Strategy

Write

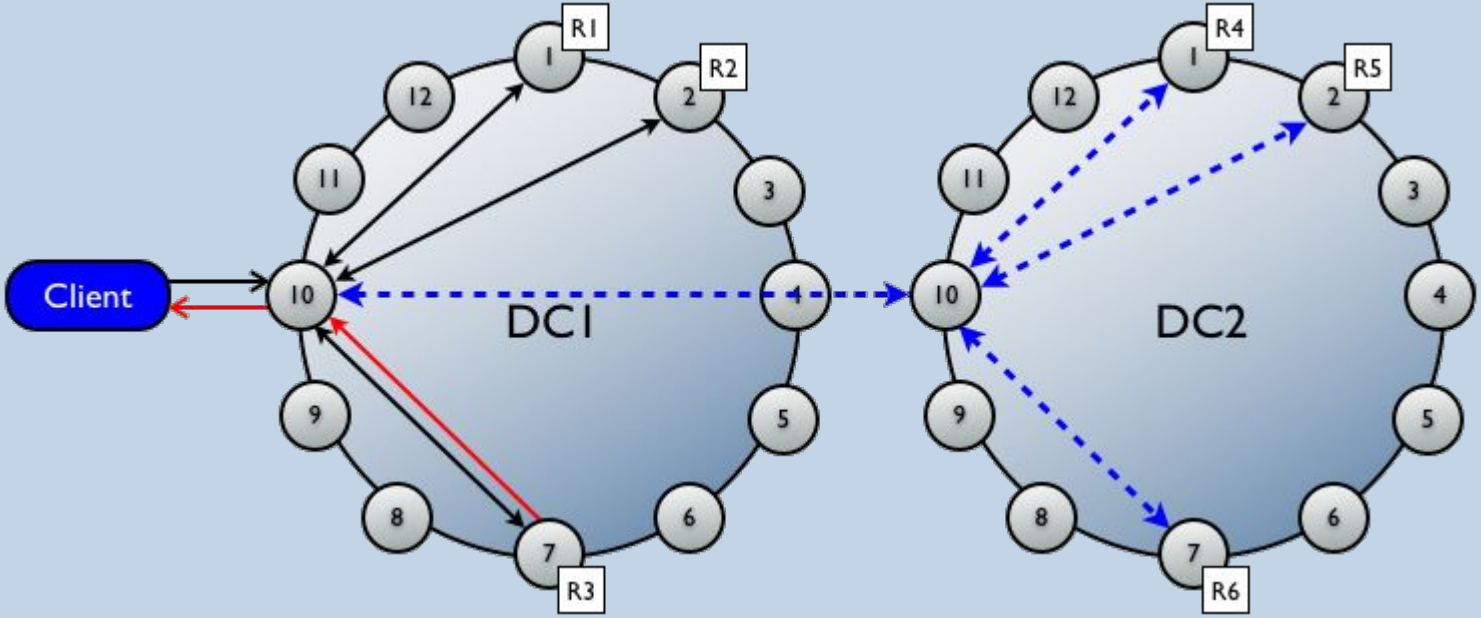
Запрос всем репликам независимо от `ConsistencyLevel`, `ConsistencyLevel` определяет, сколько реплик должно ответить для успеха



MultiDC Write

- Оптимизация — один координатор в каждом удалённом ДЦ
- ConsistencyLevel.ONE или ConsistencyLevel.LOCAL_QUORUM — обязаны ответить только локальные узлы (география не влияет на задержку)

MultiDC Write: Пример



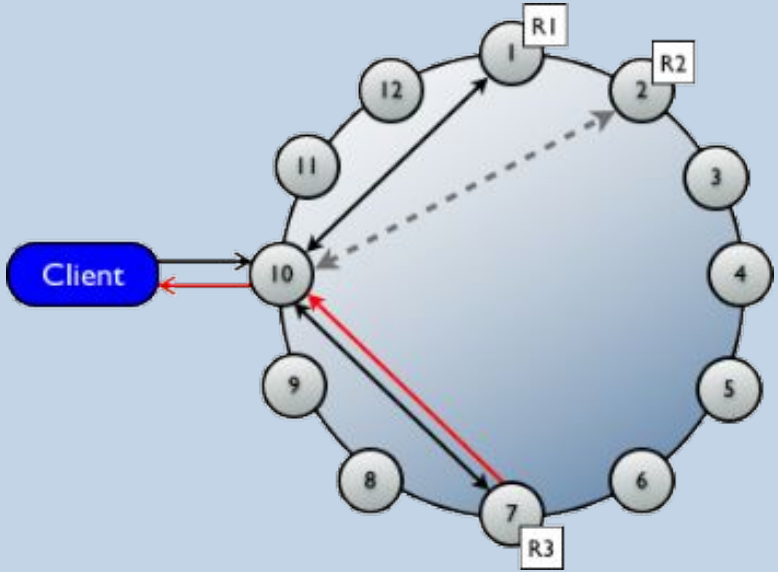
Write: ConsistencyLevel

- ANY — всегда успех (hinted handoff)
- ONE — в commit-log одного узла
- TWO
- THREE
- QUORUM — inter DC + нужен запас прочности
- LOCAL_QUORUM — быстрее, чем QUORUM
- EACH_QUORUM — выше консистентность
- ALL

Read

- Read-запросы от координатора репликам:
  - Прямой запрос на чтение (в соответствии с ConsistencyLevel)
  - Сравниваем ответы
  - Если не совпадают, то самая свежая (по timestamp) — клиенту
- Фоновый read repair (всем остальным репликам)
  - Для синхронизации «горячих» данных read_repair_chance по умолчанию 0.1 Если не совпадают хэши, то перезаписываем

Read: Пример



Read ConsistencyLevel

- ONE — ближайшая реплика, возможно, устаревшие данные
- TWO
- THREE
- QUORUM — inter DC + нужен запас прочности
- LOCAL_QUORUM — быстрее, чем QUORUM
- EACH_QUORUM — выше консистентность
- ALL

Клиентские запросы

- Проблема
  - Узел сбойнул: железо или (чаще) сеть А мы хотим на него записать
  - Что делать?

Hinted Handoff - пусть известно (Gossip), что узел лежит, или узел не отвечает. Координатор запоминает hint локально, когда обнаружится, что узел поднялся (Gossip), ему перешлют накопленные hints

Hint

- Содержимое:
  - Кому предназначается
  - Значение ключа
  - Данные

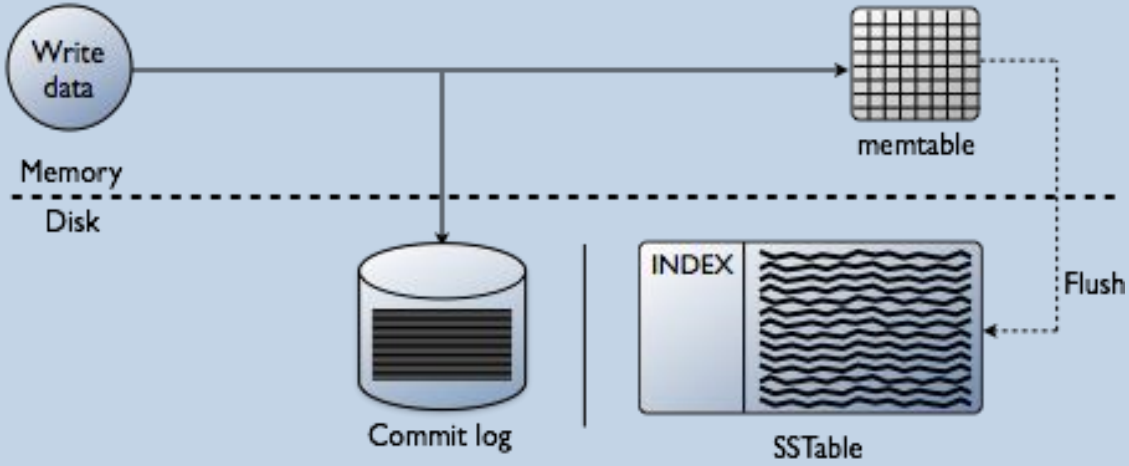
Hints хранятся ограниченное время (по умолчанию 3 часа)
Официально рекомендуется периодический запуск *repair*

Anti-entropy

- Проблема
  - Узел умер — пропустил удаление данных
    Вернулся к жизни — данные возродились

- Anti-entropy - инициируем с помощью *nodetool repair*
  - Запускаем *readonly major* compaction
  - Строим *Merkle Tree*
  - Обмениваемся деревьями и ищем отличия
  - Обмениваемся отличающимися сегментами

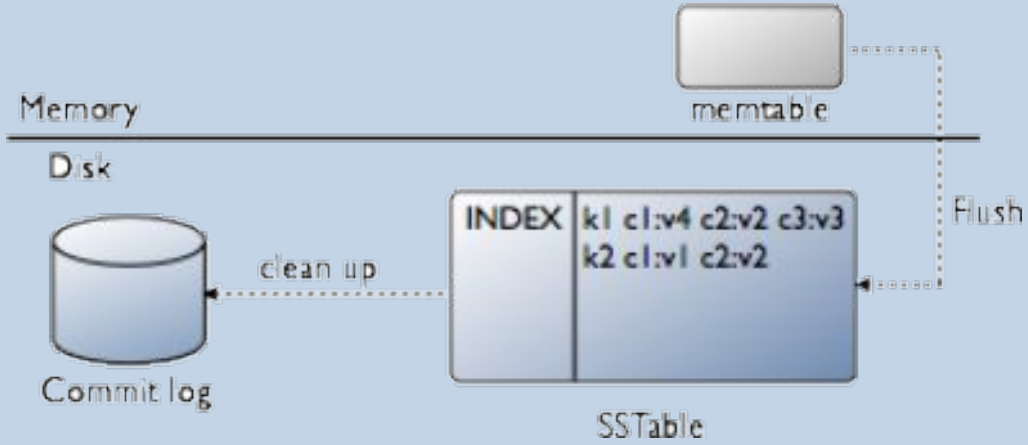
Write Path


Write data
memtable
Memory
Disk
Flush
INDEX
Commit log
SSTable

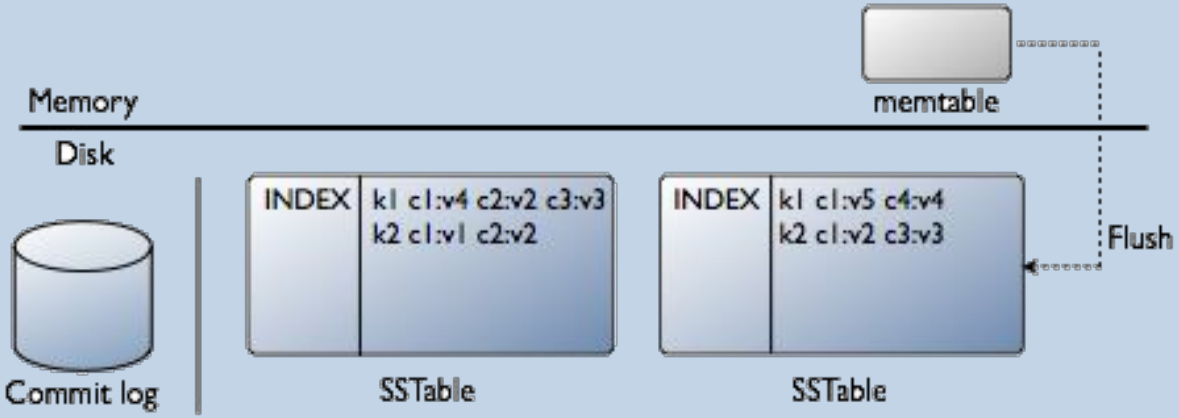
Пояснения

- Memtables and SSTables per table
- Memtable — отсортирована лежит в памяти
- SSTable — неизменяемая отсортированная ассоциативная таблица
- Обычно строка распределена по нескольким SSTable

Flush


memtable
Memory
Disk
Flush
INDEX
k1 c1:v4 c2:v2 c3:v3
k2 c1:v1 c2:v2
clean up
Commit log
SSTable

Update Path


Memory
Disk
memtable
Flush
INDEX
k1 c1:v4 c2:v2 c3:v3
k2 c1:v1 c2:v2
SSTable
INDEX
k1 c1:v5 c4:v4
k2 c1:v2 c3:v3
SSTable
Commit log

Delete

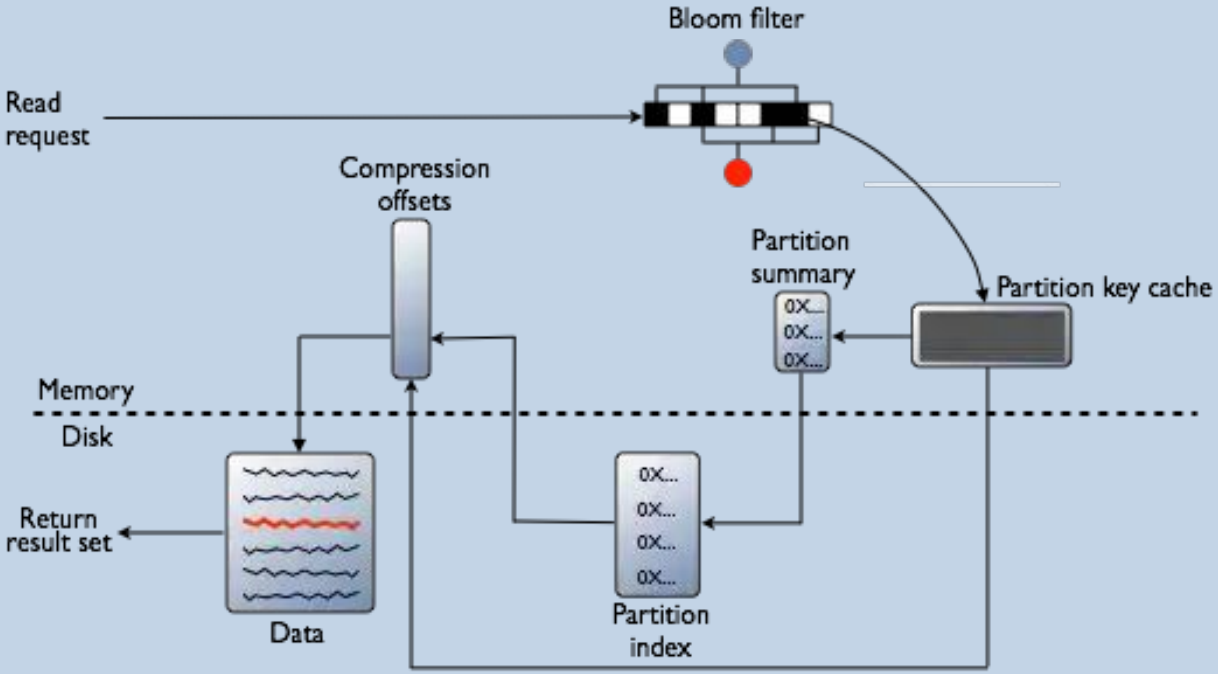
- SSTable неизменяема
- Delete — tombstone marker

Реальное удаление по истечении gc_grace_seconds во время compaction Удалённые данные могут возродиться (см. Anti-entropy)

Compaction

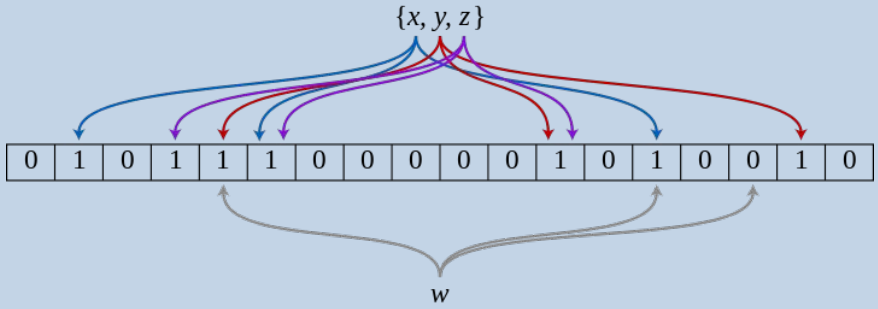
- Объединяет SSTable-файлы
  - Фрагменты строк
  - Протухшие tombstones
  - Перестраивает индексы
- SSTable отсортирована ⇒ последовательный проход
- SizeTieredCompactionStrategy и LeveledCompactionStrategy

Read


Bloom filter
Read request
Compression offsets
Partition summary
0X...
0X...
0X...
Partition key cache
Memory
Disk
Return result set
Data
0X...
0X...
0X...
0X...
Partition index

Bloom filter

Фильтр Блума — это вероятностная структура данных, придуманная Бёртоном Блумом в 1970 году, позволяющая проверять принадлежность элемента к множеству. При этом существует возможность получить ложноположительное срабатывание (элемента в множестве нет, но структура данных сообщает, что он есть), но не ложноотрицательное.



Фильтр Блума может использовать любой объём памяти, заранее заданный пользователем, причём чем он больше, тем меньше вероятность ложного срабатывания.

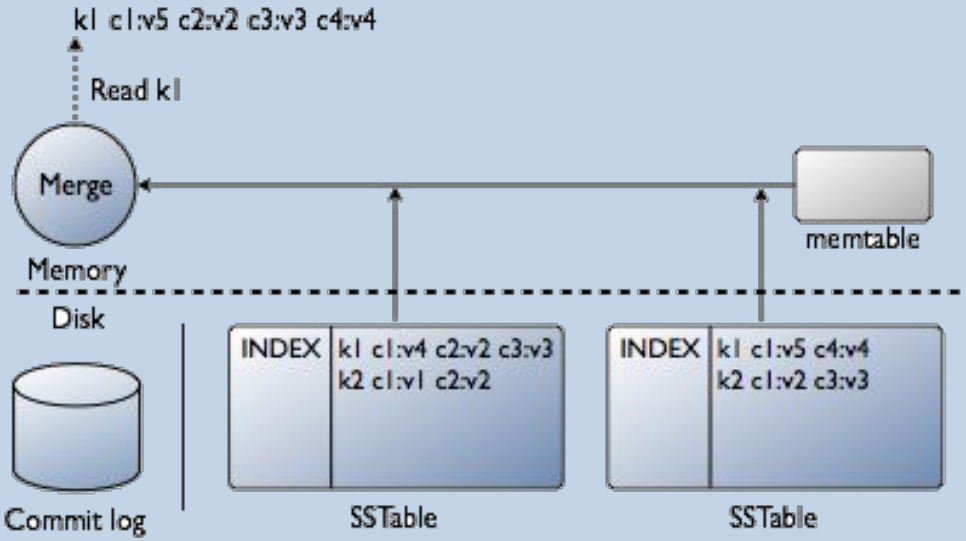
Комментарии

- Каждая SSTable имеет Bloom filter — вероятность нахождения ключа в файле
- Если вероятность отлична от 0, идём в partition key cache
- Если нашли ключ в кэше, идём по смещению, находим сжатый блок и достаём данные
- Если не нашли ключ в кэше:
  - В partition summary примерно находим смещение на диске
  - Читаем последовательно блок с диска
  - Из compression offset map вынимаем индекс блока
  - Читаем сжатые данные и возвращаем клиенту

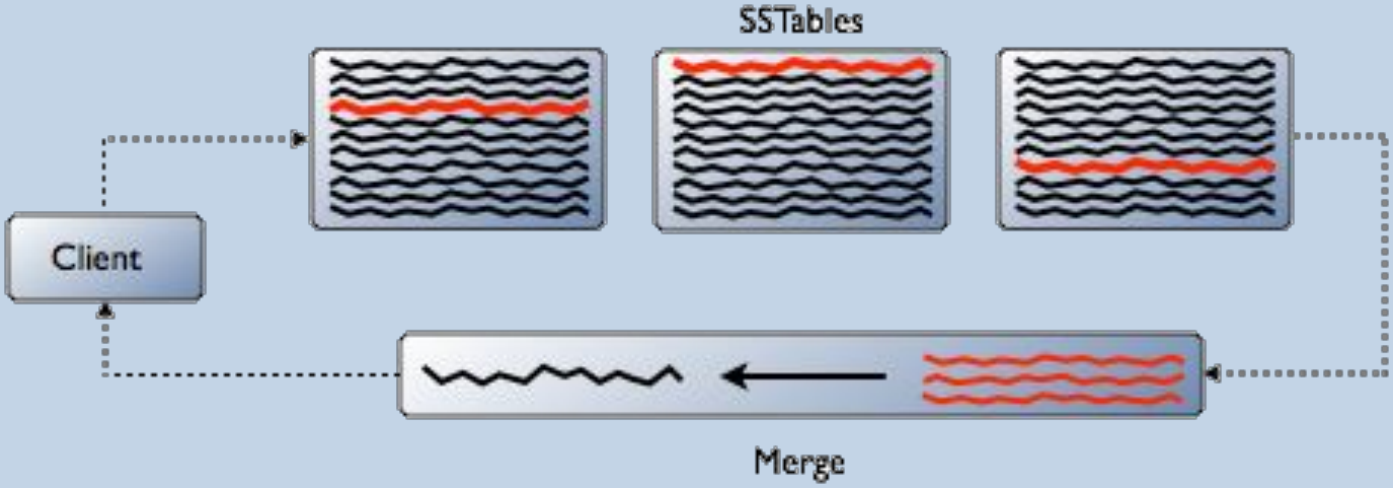
Дополнительные структуры

- Bloom filter — 1-2 ГБ / млрд. ключей
- Partition summary — по умолчанию шаг 128
- Compression offset map — 1-3 ГБ / 1 ТБ сжатых данных

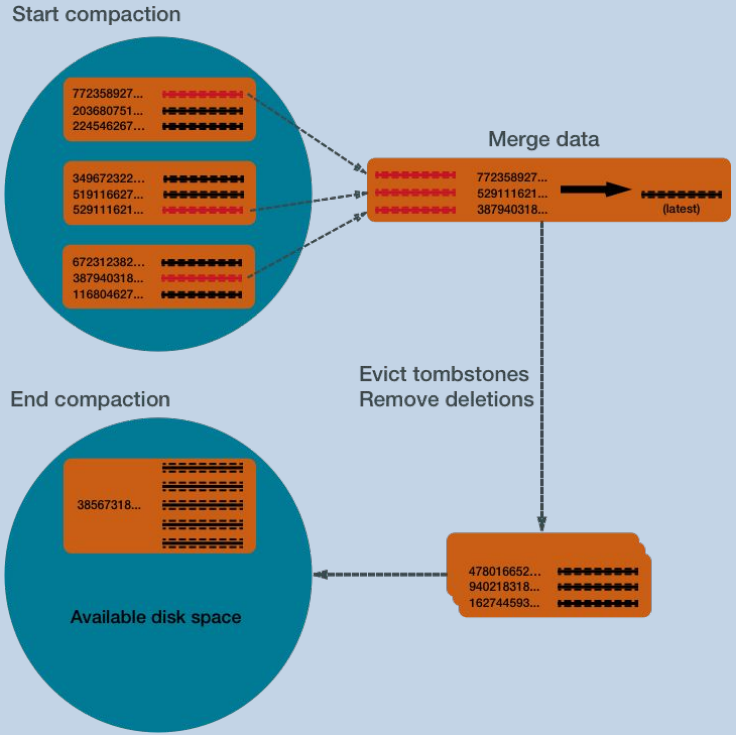
Read Path


k1 c1:v5 c2:v2 c3:v3 c4:v4
Read k1
Merge
memtable
Memory
Disk
INDEX
k1 c1:v4 c2:v2 c3:v3
k2 c1:v1 c2:v2
INDEX
k1 c1:v5 c4:v4
k2 c1:v2 c3:v3
Commit log
SSTable
SSTable

Read Path: from multiple SSTables



Compaction



Compaction strategies:

- **SizeTieredCompactionStrategy (STCS)**
  - **Write-intensive load**
- **LeveledCompactionStrategy (LCS)**
  - **Read-intensive load**
- **TimeWindowCompactionStrategy (TWCS)**
  - **Time-series data**
- **DateTieredCompactionStrategy (DTCS)**
  - **Deprecated, similar to STCS**

Lightweight transactions

LWT - не имеет ничего общего с привычными транзакциями, придумана для того, чтобы обеспечить некоторые констрейнты на уровне данных, может быть использована в insert и update запросах с помощью ключевого слова **IF** … :

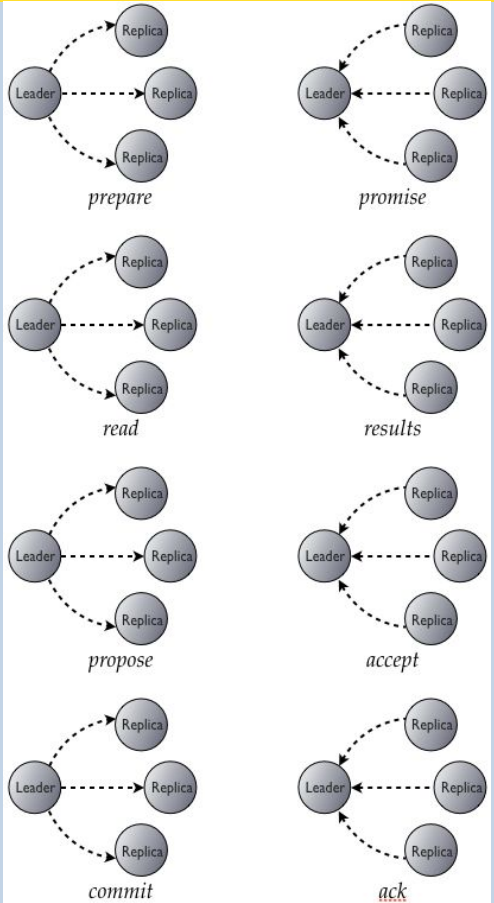
```
cqlsh> INSERT INTO cycling.cyclist_name (id, lastname, firstname)
  VALUES (4647f6d3-7bd2-4085-8d6c-1229351b5498, 'KNETEMANN', 'Roxxane')
  IF NOT EXISTS;

cqlsh> UPDATE cycling.cyclist_name
  SET firstname = 'Roxane'
  WHERE id = 4647f6d3-7bd2-4085-8d6c-1229351b5498
  IF firstname = 'Roxxane';
```

Lightweight transactions

LWT реализованны с помощью алгоритма распределенного консенсуса - Paxos (linearizable consistency)

- Необходимо помнить о затратах на коммуникацию при выполнении запроса. В общем случае кассандре понадобится 4 round trips для выполнения операции
- LWT работаю только для single partition запросах
- В условии If можно использовать только столбцы входящие в ключ таблицы



Materialized view

```
CREATE TABLE cyclist_mv (cid UUID PRIMARY KEY, name text, age int, birthday date, country text);

CREATE MATERIALIZED VIEW cyclist_by_age
AS SELECT age, birthday, name, country
FROM cyclist_mv
WHERE age IS NOT NULL AND cid IS NOT NULL
PRIMARY KEY (age, cid);
```

Materialized view: how it works

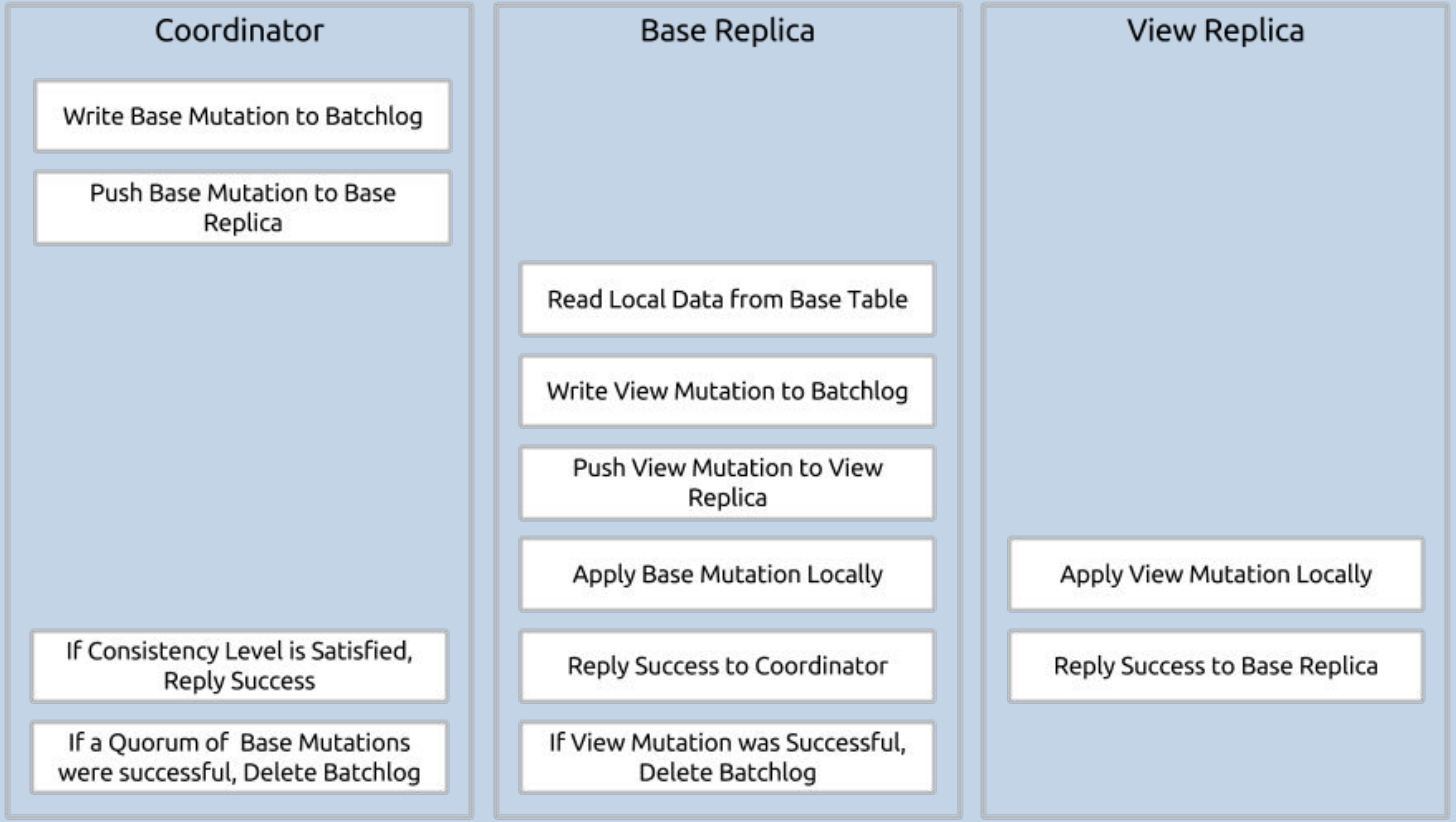
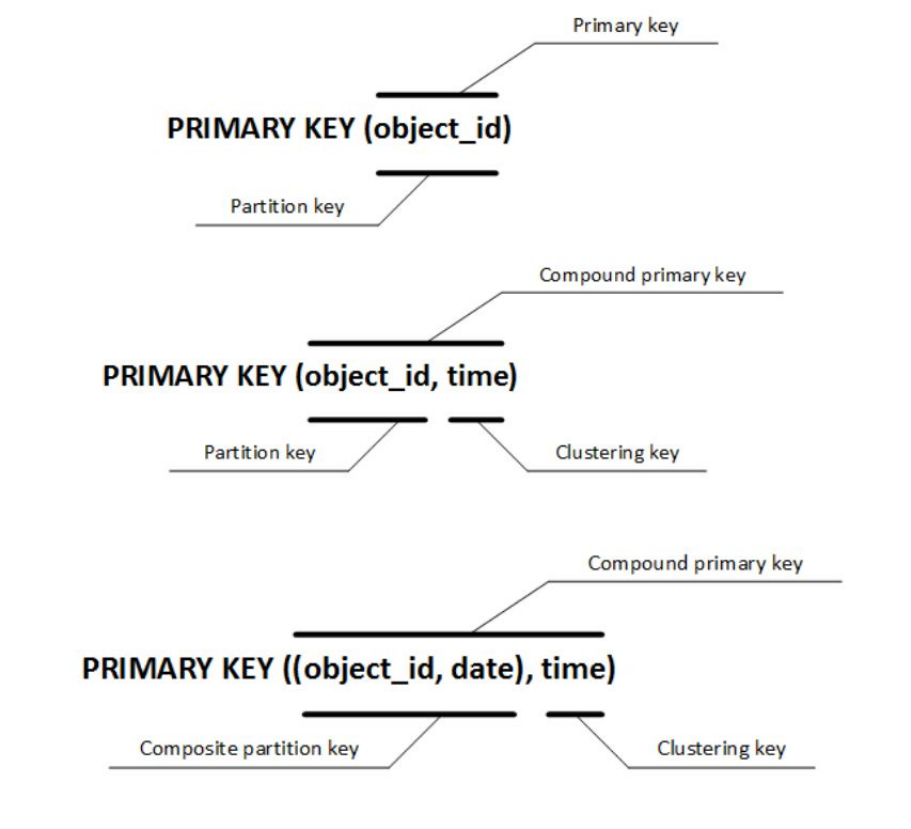


Table keys


Primary key
PRIMARY KEY (object_id)
Partition key
Compound primary key
PRIMARY KEY (object_id, time)
Partition key
Clustering key
Compound primary key
PRIMARY KEY ((object_id, date), time)
Composite partition key
Clustering key

User defined types

- В C* есть возможность создавать пользовательские типы данных на основе стандартных:
  - cqlsh> **CREATE TYPE** cycling.basic_info (
    birthday **timestamp**,
    nationality text,
    weight text,
    height text
    );
- Использовать их в таблицах
  - cqlsh> **CREATE TABLE** cycling.cyclist_stats (id uuid **PRIMARY KEY**, lastname text, basics **FROZEN**<basic_info>);
- UDT поддерживают вложенность
  - cqlsh> **CREATE TYPE** cycling.race (race_title text, race_date **timestamp**, race_time text);
    cqlsh> **CREATE TABLE** cycling.cyclist_races (id UUID **PRIMARY KEY**, lastname text, firstname text, races **list**<**FROZEN** <race>>);