Data Management in Large-Scale Distributed Systems

NoSQL Databases

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

http://tropars.github.io/

2019

References

- The lecture notes of V. Leroy
- The lecture notes of F. Zanon Boito
- Designing Data-Intensive Applications by Martin Kleppmann
  - Chapters 2 and 7

In this lecture

- Motivations for NoSQL databases
- ACID properties and CAP Theorem
- A landscape of NoSQL databases

Agenda

Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models

NoSQL databases design and implementation

Common patterns of data accesses

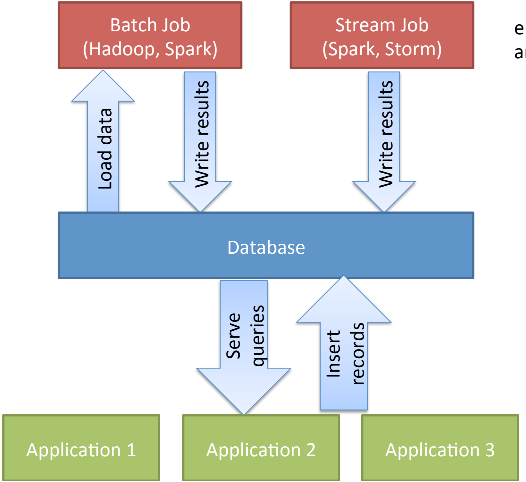
Large-scale data processing

- Batch processing: Hadoop, Spark, etc.
- Perform some computation/transformation over a full dataset
- Process all data

Selective query

- Access a specific part of the dataset
- Manipulate only data needed (1 record among millions)
- Main purpose of a database system

Processing / Database Link



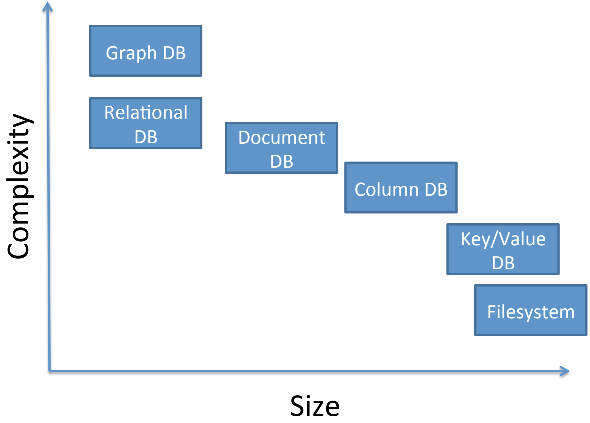
e.g. sentiment analysis

e.g. Twitter trends page

Different types of databases

- So far we used HDFS
  - A file system can be seen as a very basic database
  - Directories / files to organize data
  - Very simple queries (file system path)
  - Very good scalability, fault tolerance …
- Other end of the spectrum: Relational Databases
  - SQL query language, very expressive
  - Limited scalability (generally 1 server)

Size / Complexity



The NoSQL Jungle

Document Database	Graph Databases
Couchbase MarkLogic mongoDB	Neo4j InfiniteGraph The Distributed Graph Database

Wide Column Stores	Key-Value Databases
redis amazon DynamoDB AEROSPIKE riak	accumulo HYPERTABLE INC Cassandra APACHE HBASE Amazon SimpleDB

@cloudtxt http://www.aryannava.com

Agenda

Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models

NoSQL databases design and implementation

Relational databases

SQL

- Born in the 70's – Still heavily used
- Data is organized into relations (in SQL: tables)
- Each relation is an unordered collection of tuples (rows)

Students

ID#	Name	Phone	DOB
500	Matt	555-4141	06/03/70
501	Jenny	867-5309	3/15/81
502	Sean	876-9123	10/31/82

ID#	ClassID	Sem
500	1001	Fall02
501	1002	Fall02
501	1002	Spr03
502	1003	S203

Takes_Course

ClassID	Title	ClassNum
1001	Intro to Informatics	I101
1002	Data Mining	I400
1003	Internet and Society	I400

Courses

About SQL

Advantages

- Separate the data from the code
  - High-level language
  - Space for optimization strategies
- Powerful query language
  - Clean semantics
  - Operations on sets
- Support for transactions

Motivations for alternative models

see https://blog.couchbase.com/nosql-adoption-survey-surprises/

Some limitations of relational databases

- Performance and scalability
  - Difficult to partition the data (in general run on a single server)
  - Need to scale up to improve performance
- Lack of flexibility
  - Will to easily change the schema
  - Need to express different relations
  - Not all data are well structured
- Few open source solutions
- Mismatch between the relational model and object-oriented programming model

Illustration of the object-relational mismatch

Figure by M. Kleppmann

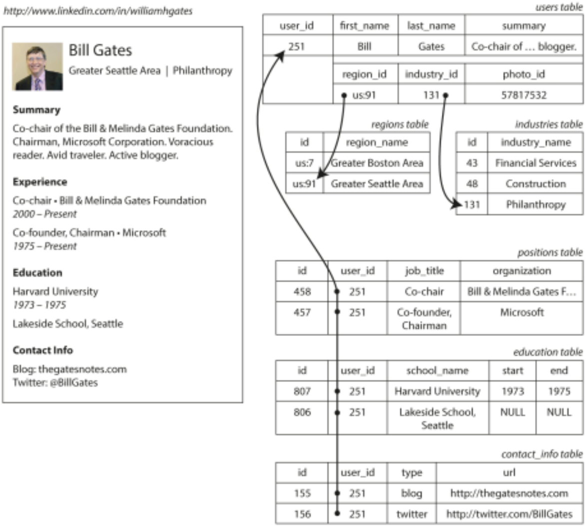


Figure: A CV in a relation database

Illustration of the object-relational mismatch

Figure by M. Kleppmann


```
{
  "user_id":251,
  "first_name": "Bill",
  "last_name": "Gates",
  "summary": "Co—chair of the Bill & Melinda Gates; Active blogger.",
  "region_id": "us:91",
  "industry_id": 131,
  "photo_url": "/p/7/000/253/05b/308dd6e.jpg",
  "positions": [
    {"job_title": "Co—chair", "organization": "Bill & Melinda Gates
          Foundation"},
    {"job_title": "Co—founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University", "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog": "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```


Figure: A CV in a JSON document

About NoSQL

What is NoSQL?

- A hashtag
  - NoSQL approaches were existing before the name became famous
- No SQL
- New SQL
- Not only SQL
  - Relational databases will continue to exist alongside non-relational datastores

About NoSQL

A variety of NoSQL solutions

- Key-Value (KV) stores
- Wide column stores (Column family stores)
- Document databases
- Graph databases

Difference with relational databases

There are several ways in which they differ from relational databases:

- Properties
- Data models
- Underlying architecture

Agenda

Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models

NoSQL databases design and implementation

About transactions

The concept of transaction

- Groups several read and write operations into a logical unit
- A group of reads and writes are executed as one operation:
  - The entire transaction succeeds (commit)
  - or the entire transaction fails (abort, rollback)
- If a transaction fails, the application can safely retry

About transactions

The concept of transaction

- Groups several read and write operations into a logical unit
- A group of reads and writes are executed as one operation:
  - The entire transaction succeeds (commit)
  - or the entire transaction fails (abort, rollback)
- If a transaction fails, the application can safely retry

Why do we need transactions?

- Crashes may occur at any time
  - On the database side
  - On the application side
  - The network might not be reliable
- Several clients may write to the database at the same time

ACID

ACID describes the set of safety guarantees provided by transactions

- Atomicity
- Consistency
- Isolation
- Durability

Having such properties make the life of developers easy, but:

- ACID properties are not the same in all databases
  - It is not even the same in all SQL databases
- NoSQL solutions tend to provide weaker safety guarantees
  - To have better performance, scalability, etc.

ACID: Atomicity

Description

- A transactions succeeds completely or fails completely
  - If a single operation in a transaction fails, the whole transaction should fail
  - If a transaction fails, the database is left unchanged
- It should be able to deal with any faults *in the middle* of a transaction
- If a transaction fails, a client can safely retry

In the NoSQL context:

- Atomicity is still ensured

ACID: Consistency

Description

- Ensures that the transaction brings the database from a valid state to another valid state
  - Example: Credits and debits over all accounts must always be balanced
- It is a property of the application, not of the database
  - The application cannot enforce application-specific invariants
  - The database can check some specific invariants
    - A foreign key must be valid

In the NoSQL context:

- Consistency is (often) not discussed

ACID: Durability

Description

- Ensures that once a transaction has committed successfully, data will not be lost
  - Even if a server crashes (flush to a storage device, replication)

In the NoSQL context:

- Durability is also ensured

ACID: Isolation

Description

- Concurrently executed transactions are isolated from each other
  - We need to deal with concurrent transactions that access the same data
- Serializability
  - High level of isolation where each transaction executes as if it was the only transaction applied on the database
    - As if the transactions are applied *serially*, one after the other
  - Many SQL solutions provide a lower level of isolation

In the NoSQL context:

- **What about the CAP theorem?**

The CAP theorem

3 properties of databases

- Consistency
  - What guarantees do we have on the value returned by a read operation?
  - It strongly relates to Isolation in ACID (and not to consistency)
- Availability
  - The system should always accept updates
- Partition tolerance
  - The system should be able to deal with a partitioning of the network

Comments on CAP theorem

- Was introduced by E. Brewer in its lectures (beginning of years 2000)
- Goal: discussing trade-offs in database design

What does the CAP theorem says?

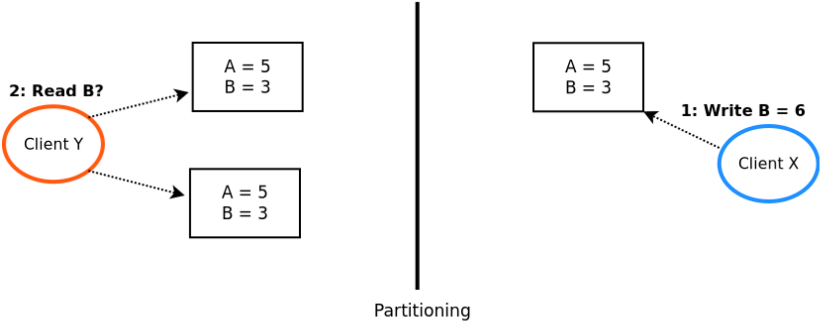
The theorem

It is impossible to have a system that provides Consistency, Availability, and partition tolerance.

How it should be understood:

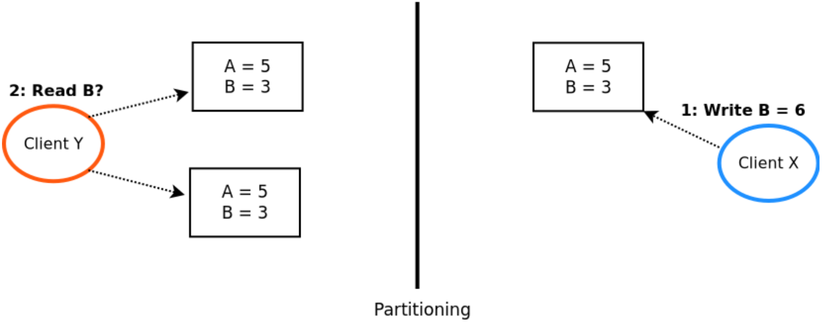
- Partitions are unavoidable
  - It is a fault, we have no control on it
- We need to choose between availability and consistency
  - In the CAP theorem:
    - Consistency is meant as *linearizability* (the strongest consistency guarantee)
    - Availability is meant as *total availability*
  - In practice, different trade-offs can be provided

The intuition behind CAP



- Let inconsistencies occur? (No C)
- Stop executing transactions? (No A)

The intuition behind CAP



- Let inconsistencies occur? (No C)
- Stop executing transactions? (No A)

Note that in a centralized system (non-partitioned relational database), no need for Partition tolerance

- We can have Consistency and Availability

The impact of CAP on ACID for NoSQL

source: E. Brewer

The main consequence

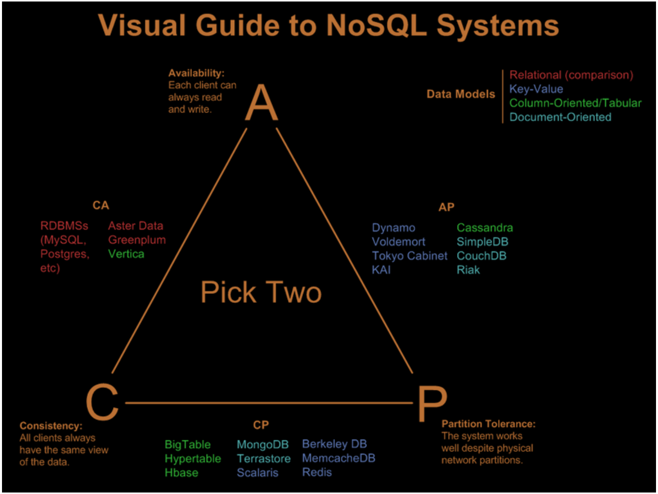
- No NoSQL database with strong Isolation

Discussion about other ACID properties

- Atomicity
  - Each side should ensure atomicity
- Durability
  - Should never be compromised

A vision of the NoSQL landscape

Source: https://blog.nahurst.com/visual-guide-to-nosql-systems



To be read with care:

- Solutions often provide a trade-off between CP and AP
- A single solution may often a different trade-off depending on how is is configured.
- **We don't pick two !**

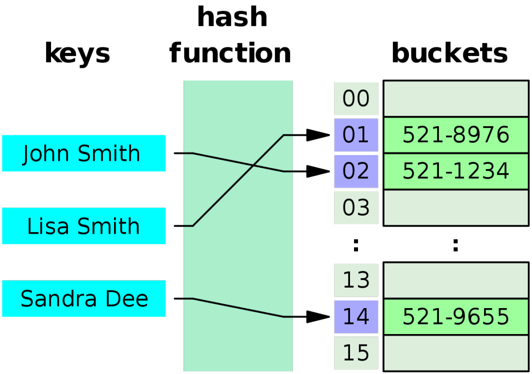
Agenda

Key-Value store

- Data are stored as key-value pairs
  - The value can be a data structure (eg, a list)
- In general, only support single-object transactions
  - In this case, key-value pairs
- Examples:
  - Redis
  - Voldemort
- Use case:
  - Scalable cache for data
  - Note that some solutions ensure durability by writing data to disk

Key-value store

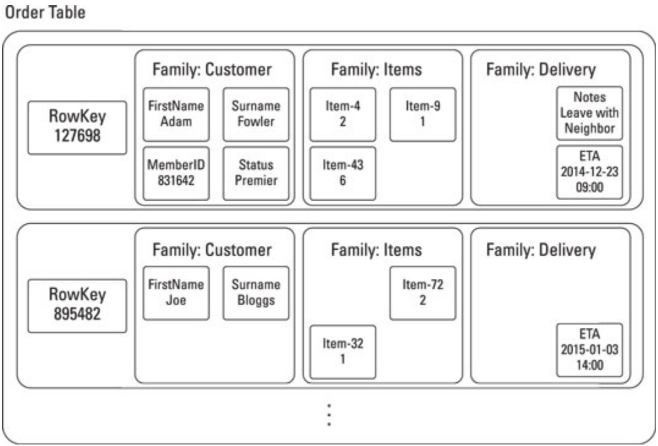
Image by J. Stolfi



Column family stores

- Data are organized in rows and columns (Tabular data store)
  - The data are arranged based on the rows
  - Column families are defined by users to improve performance
    - Group related columns together
- Only support single-object transactions
  - In this case, a row
- Examples:
  - BigTable/HBase
  - Cassandra
- Use case:
  - Data with some structure with the goal of achieving scalability and high throughput
  - Provide more complex lookup operations than KV stores

Column family stores



Note that not a row does not need to have an entry for all columns

Document databases

- Data are organized in Key-Document pairs
  - A document is a nested structure with embedded metadata
  - No definition of a global schema
  - Popular formats: XML, Json
- Only support single-object transactions
  - In this case, a document or a field inside a document
- Examples:
  - MongoDB
  - CouchDB
- Use case:
  - An alternative to relational databases for structured data
  - Offer a reacher set of operations compared to KV stores: Update, Find, etc.

Document DB

A document can have one or more documents inside.

```
{
    {
        "_id": ObjectId ("51c4218"),
        "name": "Claudia",
        "NumberKids": 3,
        "isActive": true,
        "interests": ["swimming", "tennis"]
        "favoriteCountries":
            [
              {
                   "name": "France",
                   "capital" : ”Paris"
              },
              {
                   "name": "Japan"
              }
            ]
    },
    {
        "_id": 2,
        "name": "Rubby"
        "friends": 354,
        "favorite Country":
        {
            "name": "Italy",
            "capital": "Rome"
        }
    }
}
```

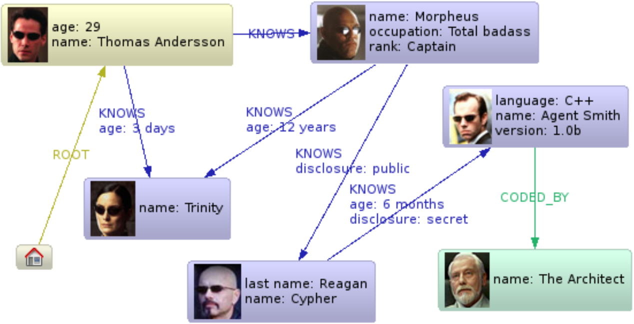
Embedded document

Embedded document

Embedded document

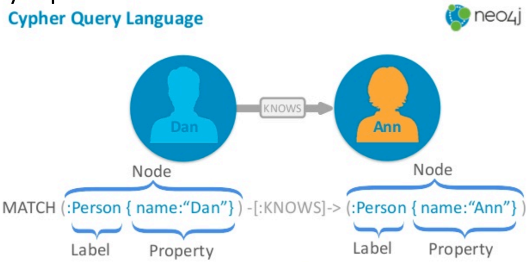
Graph DB

- Represent data as graphs
  - Nodes / relationships with properties as K/V pairs



Graph DB: Neo4j

- Rich data format
  - Query language as pattern matching
  - Limited scalability
    - Replication to scale reads, writes need to be done to every replica



On the Many-to-one relationship

Many-to-one relationship

- Many items may refer to the same item
- Example: Many people went to the same university

Relational vs Document DB

On the Many-to-one relationship

Many-to-one relationship

- Many items may refer to the same item
- Example: Many people went to the same university

Relational vs Document DB

- Relational databases use a foreign key
  - Consistency and low memory footprint (normalization)
  - Easy updates and support for joins
  - Difficult to scale
- Document databases duplicate data
  - Efficient read operations
  - Easy to scale
  - Higher memory footprint and updates are more difficult (risk of consistency issues)
    - Transactions on multiple objects could be very useful in this case
  - Join operations have to be implement by the application

More on relations

One-to-many relationship

- An item may have several entries of the same kind
- Example: One person may have had several positions during her career.
- Document DB allow storing such information easily and allow simple read operations

Many-to-many relationship

- An item may have several entries of the same kind that are referred by multiple items
- Example: Several persons may have worked in the same company.
- Document DB may not have good support for such relationships

Agenda

Introduction

Why NoSQL?

Transactions, ACID properties and CAP theorem

Data models

NoSQL databases design and implementation

Google BigTable

- Column family data store
  - Data storage system used by many Google services: Youtube, Google maps, Gmail, etc.
- Paper published by Google in 2006 (F. Chang et al)
  - Now available as a service on Google Cloud
- Many ideas reused in other NoSQL databases

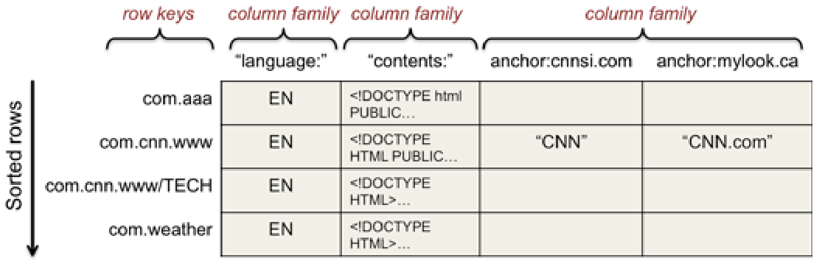
Motivations

- A system that can stores very large amount of data
  - TB or PB of data
  - A very large number of entries
  - Small entries (each entry is an array of bytes)
- A simple data model
  - Key-value pairs (A key identifies a row)
  - Multi-dimentional data
  - Sparse data
  - Data are associated with timestamps
- Works at very large scale
  - Thousands of machines
  - Millions of users

About the data model

- Rows are identified by keys (arbitrary strings)
  - Modifications on one row are atomic
  - Rows are maintained in lexicographic order
- Columns are grouped in columns families
  - Columns can be sparsed
  - Clients can ask to retrieve a column familly for one row
- Each cell can contain multiple versions indexed by a timestamp
  - Assigned by BigTable or by the client
  - Most recent versions are accessed first
  - GC politics:
    - Keep last n versions
    - Keep all new-enough versions

About the data model



Partitioning and performance

see https://cloud.google.com/bigtable/docs/schema-design

Partitioning

- Partitioning on the rows
- Rows with close keys are in the same partition

Recommendations about the schema for performance

- Accesses can be made based on key, key-prefix or key-range
  - Choose keys appropriatly to make sequential accesses to a single host
  - Example: Reverse domain name, timestamps
  - To avoid: Domain name, hash values
  - Take advantage of the concept of key prefix
- Group related columns in a column family
  - Avoids retrieving all data from a single row when not needed
- Creating plenty of tables is not a good pattern
  - Use column families instead

Sparse columns

see `https://cloud.google.com/bigtable/docs/overview`

"follows" column family

	Follows			
Row Key	gwashington	jadams	tjefferson	wmckinley
gwashington		1		
jadams	1		1	
tjefferson	1	1		1
wmckinley			1	

Multiple versions

Building blocks of BigTable

A master

- Assign tablets to severs
- With the help of a locking service

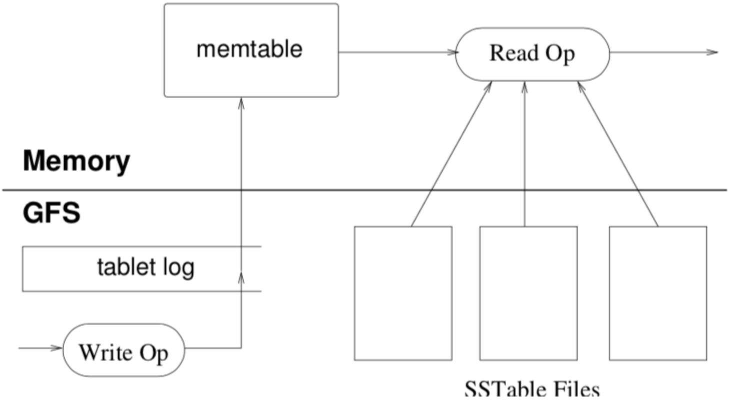
Tablet servers

- Store the tables (divided in tablets)
- Process client requests

Tablets

- Stored as SSTables (Sorted string tables)
- Stored in the Google File System for durability

Implementation of tablets



Implementation of tablets

Write operations

- Data stored in memory (Memtable)
- Any update is written to a commit log on GFS for durability
  - The log is shared between all hosted tablets

Periodic writes to disk

- When the memtable becomes too big:
  - Copied as a new SSTable to GFS
    - Multiple SSTables are created if locality groups are defined (based on column families)
  - Reduces the memory footprint and reduces the amount of work to do during recovery
  - SSTables are immutable (no problem of concurrency control)
- Operation called minor compaction

Implementation of tablets

Read operations

- The state of the tablet = the memtable + all SSTables
  - A merged view needs to be created
  - The memtable and the SSTables may contain delete operations
- Locality groups help improving the performance of read operations

Major compaction

- When the number of SSTables becomes too big, merge them into a single SSTable
  - Allow reclaiming resources for deleted data
  - Improve the performance of read operations

Improving the performance of read operations

- During a read operation, potentially several SSTables need to be read

- How to avoid reading all SSTables when not needed?
  - Use of Bloom filters
  - Data structure that allows us to know if a SStable contains an entry for a given key-column pair

Improving the performance of read operations

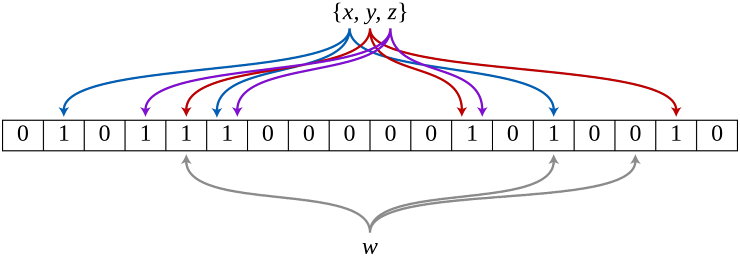
- During a read operation, potentially several SSTables need to be read
- How to avoid reading all SSTables when not needed?
  - Use of Bloom filters
  - Data structure that allows us to know if a SStable contains an entry for a given key-column pair

Bloom filter

- Implements a membership function (is X in the set?)
- If the bloom filter answers no: it is guaranteed that X is not present
- If the bloom filter answers yes: the element is in the set with a high probability
- Good trade-off between accuracy and memory footprint

About bloom filters

- A vector of n bits and k hash functions
- On insert:
  - Compute the k hash values
  - Set the corresponding bits to 1 in the vector
- On lookup:
  - Compute the k hash values
  - Test whether all bits are set to 1



About the logs

On one node, a single commit log is created even if it hosts multiple tablets.

Advantages

Drawbacks

About the logs

On one node, a single commit log is created even if it hosts multiple tablets.

Advantages

- Write a single append-only file on disk
  - Improves performance by avoiding long seeks

Drawbacks

- Recovery is more complex since the log includes data associated with different tablets
- The tablets might be distributed over multiple nodes

Apache Cassandra

- Column family data store
- Paper published by Facebook in 2010 (A. Lakshman and P. Malik)
  - Used for implemeting search functionalities
  - Released as open source
- Build on top of several ideas introducted by BigTable
  - Warning: Many changes in the design have been made since the first version of Cassandra

To be continued

Additional references

Mandatory reading

- *Bigtable: A Distributed Storage System for Structured Data.*, F. Chang et al., OSDI, 2006.
- *Cassandra: a decentralized structured storage system* ., A. Lakshman et al., SIGOPS OS review, 2010.

Suggested reading

- `http://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html`, M. Kleppmann, 2015.
- `https://jvns.ca/blog/2016/11/19/a-critique-of-the-cap-theorem/`, J. Evans, 2016.