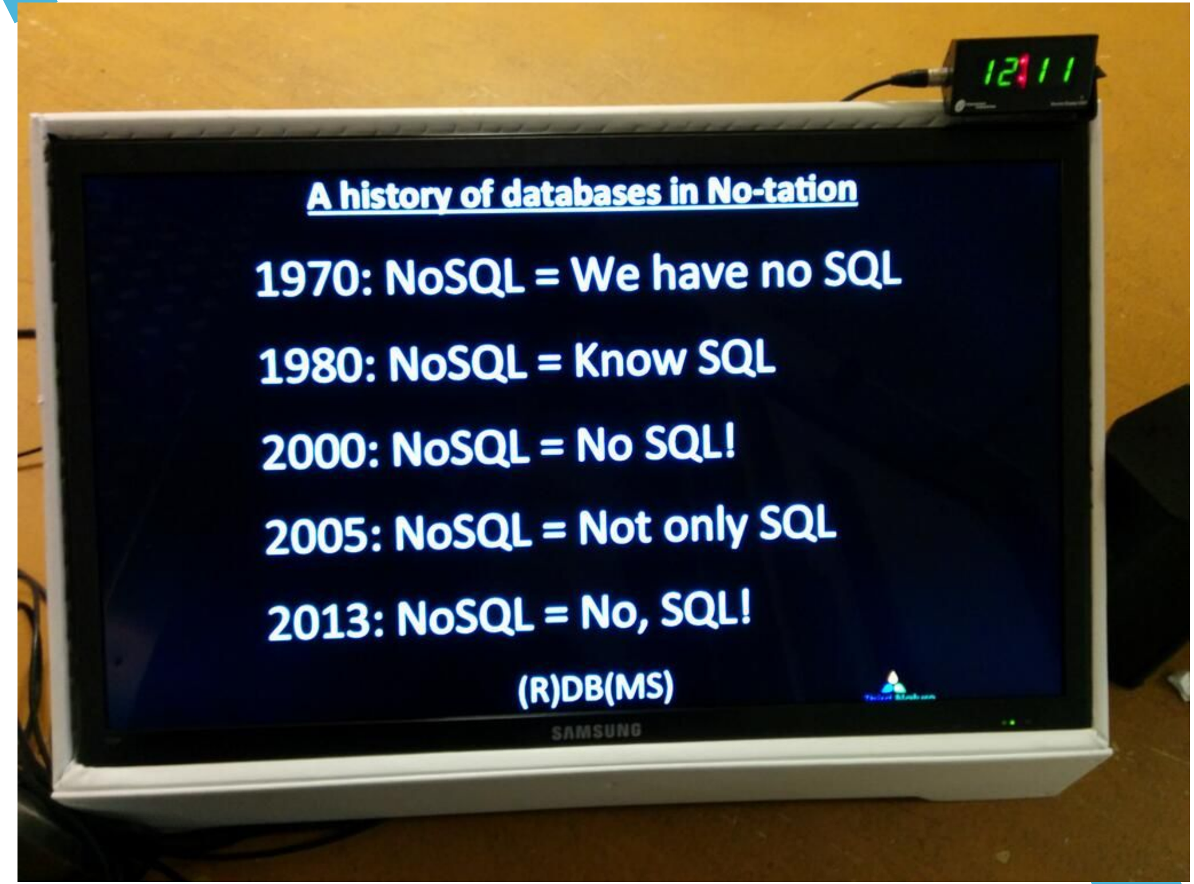
Introduction to NO SQL

What is No SQL


A history of databases in No-tation
1970: NoSQL = We have no SQL
1980: NoSQL = Know SQL
2000: NoSQL = No SQL!
2005: NoSQL = Not only SQL
2013: NoSQL = No, SQL!
(R)DB(MS)
SAMSUNG
12:11

What is NoSQL?

There is **no clear definition of a NoSQL** data system but some **characteristics** the systems **could share** are:

- **Not** using the **relational model** for storing data.
- **Not** using the **SQL** language for retrieving data.
- **No schema**, allowing fields to be added to any record without controls.
- Ability to run on **clusters** of commodity hardware.
- Designed with web-scale **horizontal scalability** in mind.
- Have the ability to **trade off traditional consistency** for other useful properties (no ACID support or transactions support).

SQL

refreshing concepts of RDBMS

What is SQL?

- **SQL:** Structured Query Language is a domain-specific language used in programming and designed for managing data held in a relational database management system (**RDBMS**), or for stream processing in a relational data stream management system (RDSMS).
- Originally based upon **relational algebra** and **tuple relational calculus**, SQL **consists of** a data definition language (**DDL**), data manipulation language (**DML**), data control language (**DCL**), a query Language and the possibility of procedural statements.
- Was one of the first commercial languages for Edgar F. Codd's relational model, **Became a standard of ANSI and ISO.**
- Eventually became a **synonym of RDBMS**.

SQL Statements

DDL

```
CREATE TABLE Book(
 isbn INTEGER,
 title VARCHAR(50),
 publication DATE NOT NULL,
 PRIMARY KEY (isbn)
);

ALTER TABLE Book ADD type VARCHAR(3);

DROP TABLE Book;
```

Query

```
SELECT Book.isbn AS Title,
       count(*) AS Authors
  FROM  Book
  JOIN  Book_author
    ON  Book.isbn =
Book_author.isbn
 GROUP BY Book.title;
```

DML

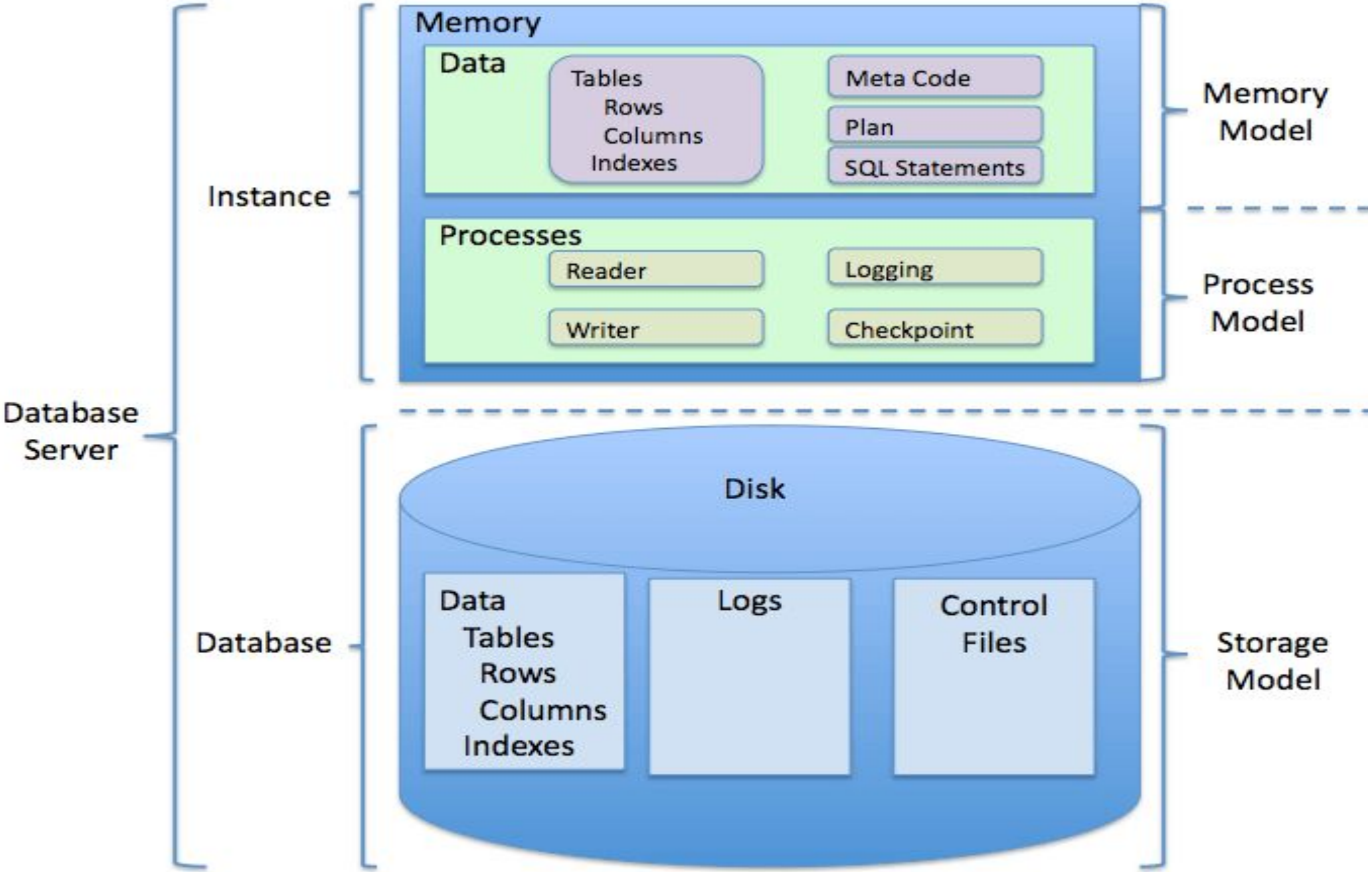
```
INSERT INTO Book (isbn, title)  VALUES (978-1501142970, 'IT');

UPDATE Book  SET isbn = 'IT: a Novel'  WHERE isbn = ' 978-1501142970;

DELETE FROM Book  WHERE isbn =  978-1501142970;
```

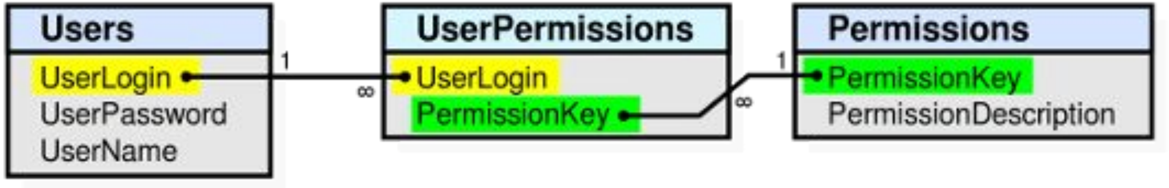
What is RDBMS?

A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database.



Benefits of a RDBMS

Relational Model



- » Stores the **data as a set of relations** represented by tables.
- » A **table has** rows and columns, where **rows** (or tuples) represents **records** and **columns** represent the **attributes**.
- » Each **row** may have a **key** that **identifies** the row **uniquely** (primary key.
- » **Relation between tables** are shown by common attributes (**foreign key**).
- » Constraints are derived from the use of keys.

Benefits of a RDBMS

Schema is the physical implementation of the Model and provides the ability to define:

- **Data types** to each attribute of the Tables.
  - **Define constraints** on the domain of the data that can be inserted/updated.
  - **Allow** for some **optimization on storage**.
- **Indexes:** to optimize search and filtering on certain attributes.

Benefits of a RDBMS

ACID Transactions: Allow for a set of operations can be perceived as single logical operation

- **Atomicity**: requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails.
- **Consistency**: Any data written to the database must be valid according to all defined rules.
- **Isolation:** ensures that the concurrent execution of transactions results as if transactions were executed sequentially.
- **Durability**: ensures that once a transaction has been committed, it will remain so.

Scaling with RDBMS

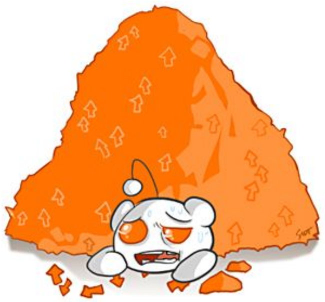
» **Initial public launch:** Move from local workstation to shared, remote hosted MySQL instance with a well-defined schema.

» **Service becomes more popular: too many reads** hitting the database. Add memcached to cache common queries. Reads are now no longer strictly ACID; cached data must expire. Other option add read replicas; more maintenance and infrastructure complexity.

» **Service continues to grow in popularity**: **too many writes** hitting the database. Scale database vertically by buying a beefed up server with 16 cores, 128 GB of RAM, and banks of 15k RPM hard drives.

Scaling with RDBMS

- **New features increases query complexity:** now we have **too many joins**. Denormalize your data to reduce joins.
- **Rising popularity swamps the server:** **things are too slow**. Stop doing any server-side computations.
- **Some queries are still too slow**: Periodically pre-materialize the most complex queries, try to stop joining in most cases.
- **Reads are OK, but writes are getting slower and slower:** Drop secondary indexes and triggers.



reddit is under heavy load right now

What's wrong with current RDBMS?

- Current RDBMS are **too rigid** for certain scenarios.
- Not ideal for storing **non-structured data**.
- **One-size-fits-all** approach not valid anymore.
- **Hard to scale** and maintain when handling billions of rows.
- **Data structures** used in RDBMS are optimized for systems with small amounts of memory.

No SQL

To the rescue...

Why NoSQL help?

NoSQL stores try to solve scalability issues by

- Relaxing schema constraints or even removing schema altogether.
- Removing or relaxing the ACID transaction properties.
- Using indexes more suited to fast modification.
- Scale horizontally on commodity hardware.

Not all stores use the same strategy the all pick and choose given the objective they set up to solve.

But as they solve scalability issues, other concessions must be made

Brewer's CAP theorem

- **Consistency**: refers to strong consistency of updates; all nodes see the same data at the same time.
- **Availability**: guarantees that every request receives a response about whether it succeeded or failed.
- **Partition tolerance**: the system continues to operate despite arbitrary message loss or failure of part of the system.

Brewer's theorem states that **in a distributed data system** (in the event of a partition) **can guarantee either strong consistency or availability**, <u>but not both</u>.

NoSQL systems **usually choose to relax the consistency** requirements and favor availability. They tend to use <u>eventual consistency</u>.

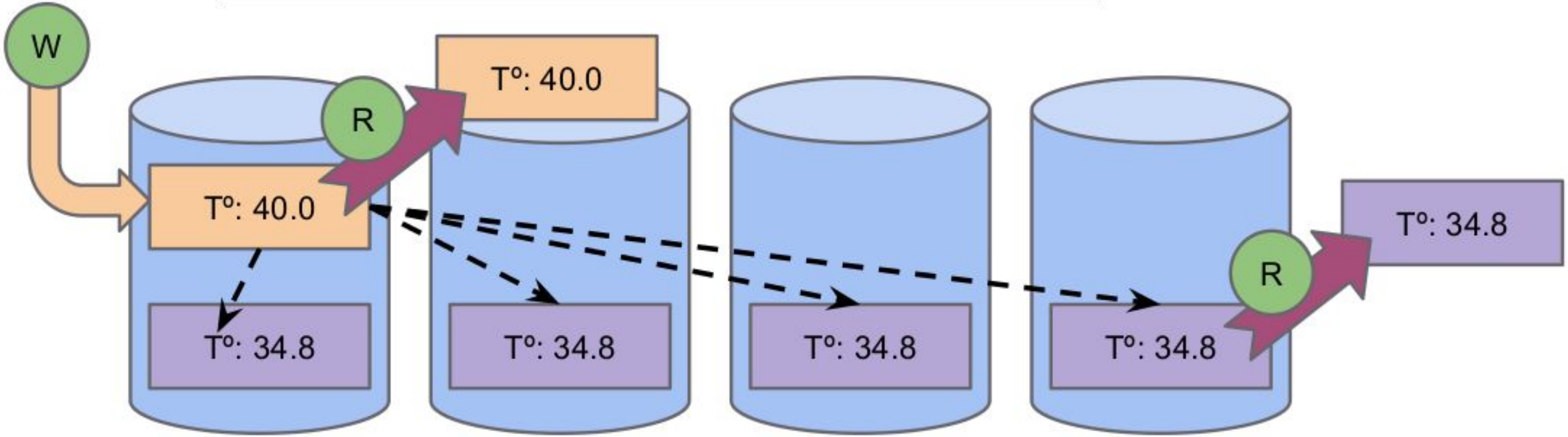
NoSQL: B+ Trees vs LSM Trees

» **B+ Trees**
- Used as **indexes for traditional SQL databases**.
- Store data for **efficient retrieval** in file systems.
- **Multiple random disk seeks** when performing a write.

» **LSM Trees**: Log-structured merge-tree
- Indexes for **Big-Table like databases** (HBase, Cassandra, DynamoDB, etc.).
- Maintain **data in two or more separate structures**, each of which is optimized for its respective underlying storage medium.
- Oriented towards **write heavy workloads**.
- **Reading data suffers a penalty** because multiples levels have to be searched (Bloom Filters help improve this).

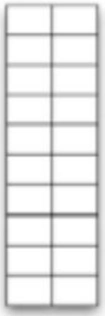
Eventual consistency

- It is a **promise that eventually**, in the absence of new writes, **all replicas** that are responsible for a data item **will agree on the same version**.
- By **defining fewer replicas**, read and write operations complete more quickly, **lowering latency**. But increases data loss risk.

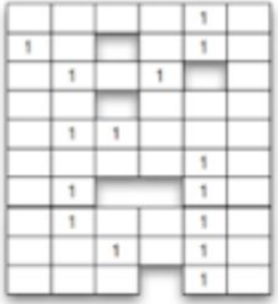


Types of NoSQL stores

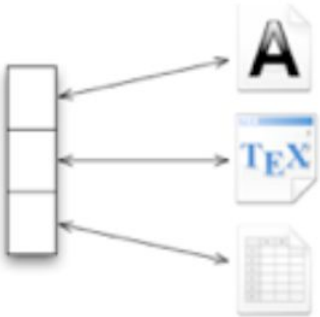
Key-Value



Column



Document



Graph

And a few more

Key Value Store

Key Value Stores

- A key-value store is a simple database that **uses** an associative array (**a map or dictionary**) as the fundamental data model where e**ach key is associated with one and only one value** in a collection. This relationship is referred to as a key-value pair.
- In each key-value pair the key is represented by an arbitrary **string** such as a **filename, URI or hash**.
- The v**alue can be any kind of data** and is st**ored as a blob** requiring **no** upfront data modeling or **schema** definition.
- This **removes the need to index** the data to improve performance. However, **you cannot filter** or control what's returned from a request based on the value because the value is opaque.

Key Value Stores

- In general, key-value stores **have no query language**. They provide a way to store, retrieve and update data **using simple get, put and delete commands**.
- The path to retrieve data is a direct request to the object in memory or on disk.
- **The simplicity of this model makes a key-value store fast, easy to use, scalable, portable and flexible.**

redis

Redis

REmote **DI**ctionary **S**erver created by Salvatore Sanfilippo.

- Is an open source, **in-memory data structure store**, used as a **database, cache and message broker.**
- It supports data structures such as:
  - strings.
  - hashes.
  - lists, sets, sorted sets with range queries.
  - bitmaps.
  - hyperloglogs
  - geospatial indexes with radius queries.

Redis

Other features include:.

- Transactions.
- Pub/Sub.
- Lua scripting.
- Keys with a limited time-to-live.
- LRU eviction of keys.
- Different levels of on-disk persistence, and
- Automatic failover and high availability via Redis Sentinel
- Built-in replication and automatic partitioning with Redis Cluster

Redis: most common use cases

- Cache.
- Rankings (TOP K).
- Session Store.
- Job Queue.
- PubSub.
- Cálculo de Unique Visitors.
- Real-time Analytics.

Redis Types and Examples of Use

Key Value as store

```
$ SET session:john "j#233ABEFDFD"
$ GET session:john
"j#233ABEFDFD"
$ DEL session:john
```

Key Value as Counter

```
$ SET /user/1234/count 10 #init in 10
$ INCR /user/1234/count #add one
$ INCRBY /user/1234/count 5 # add five
$ GET /user/1234/count
(integer) 16
```

Ranking - Ordered Set

```
$ ZADD top/author 500 "Alan"
$ ZADD top/author 200 "Grace"
$ ZADD top/author 700 "Richard"
$ ZINCRBY top/author 400 "Grace"
#add 400
$ ZADD top/author 1200
"Yukihiro"
$ ZCARD top/author  #how many in
the set (integer) 4
$ ZCOUNT top/author 300 500 #how
many in the range
(integer) 1
$ ZRANGE top/author 2 5 #who's
3, 4, 5
1) "Richard"
2) "Yukihiro"
```

Unique count

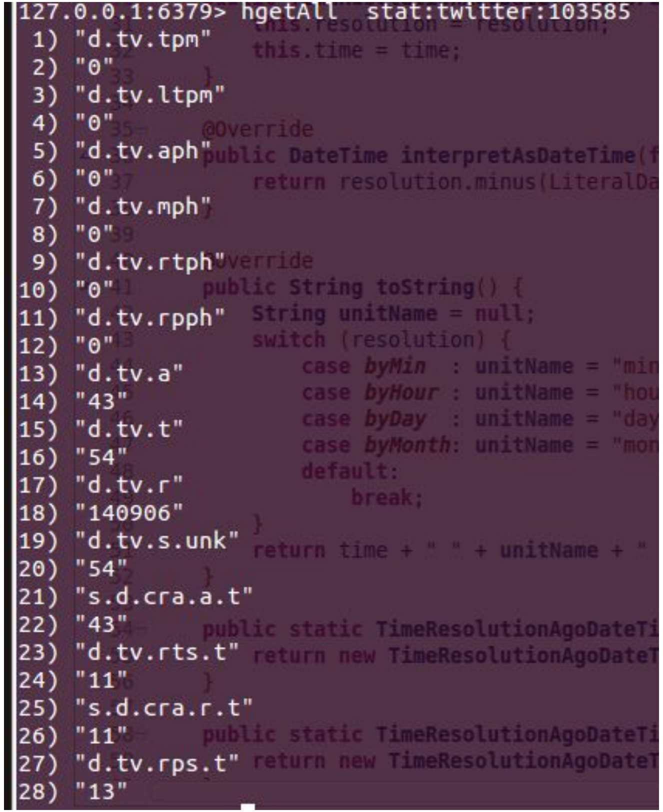
```
$ PFADD uniqes/author/1 john jack john
jill
$ PFCOUNT uniqes/author/1
(integer) 3
$ PFCOUNT uniqes/author/2 ann peter jack
$ PFMERGE uniqes/author/all
uniqes/author/2 uniqes/author/1
$ PFCOUNT uniqes/author/all
(integer) 5
```

List

```
RPUSH tags "coffe" #append to the list
RPUSH tags "tea"
RPOP tags  #removes last value
"tea"
LINSERT tags BEFORE "coffe" "water"
LLEN tags #size of the list
(integer) 2
LRANGE tags 0 -1 #show all the elements
```

Redis Types and Examples of Use

A key value pair where the value is of type **Hash**. Used to store "objects".



REDIS: considerations

- Supports MEMCACHE protocol.
- Data must fit in memory.
- Not to be used as a persistent data store.
- Cluster mode is a recent addition.

REDIS: references

- http://redis.io/topics/introduction
- https://redis.io/topics/quickstart
- https://try.redis.io/
- http://www.slideshare.net/enebo/redis-an-introduction

Columnar Store

Columnar Storage

- A columnar database is **optimized for reading and writing columns of data** as opposed to rows of data.
- Column-oriented storage is an important factor in analytic query performance because **it drastically reduces the overall disk I/O requirements** and reduces the amount of data you need to load from disk.
- Column-oriented databases **are designed to scale “out” using distributed clusters** of low-cost hardware to increase throughput, making them ideal for data warehousing and Big Data processing.

Columnar Storage

Here is an example of a table with 3 columns and 3 rows:

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

In a row oriented database the information is saved in files as follows:

A1	B1	C1	A2	B2	C2	A3	B3	C3

Whereas in a column oriented database the information is saved in files as follows:

A1	A2	A3	B1	B2	B3	C1	C2	C3

and probably each column could be in a separate file

Columnar Storage

This way of storing data in columns bring the following advantages:

» Allows for better compression, as data is more homogenous.
» I/O will be reduced as we can efficiently scan only the columns that are needed for the query (instead of the dreaded "full scan").
» As each column is stored separately we can use encodings better suited for modern processors.

APACHE

HBASE

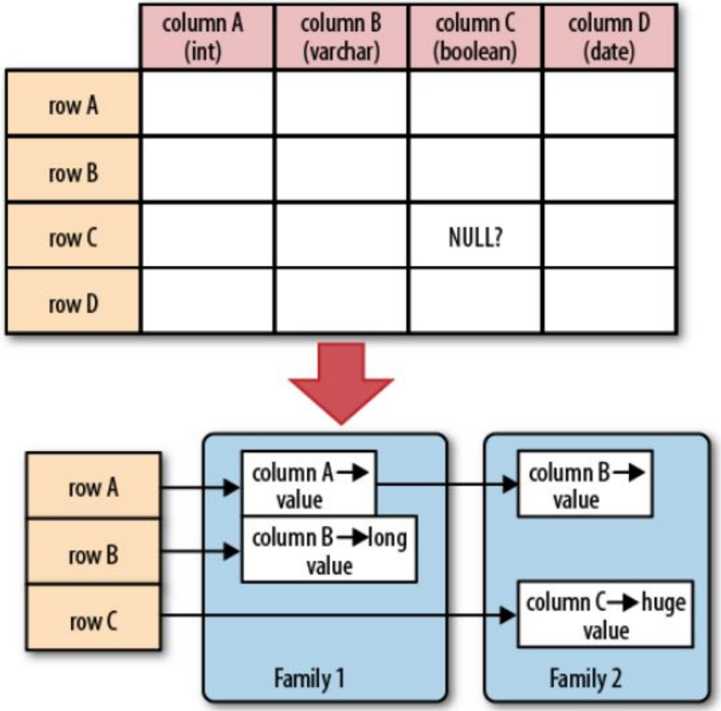
HBASE

Apache HBase is an **open source, distributed, scalable, consistent, low latency, random access non-relational (column-oriented) database** built over Apache Hadoop.

- » Based on Google's BigTable.
- » Sparse, distributed, persistent, multidimensional map.
- » Provides NoSQL database capabilities on top of Hadoop.
- » The main company supporting the HBase project nowadays is Cloudera.

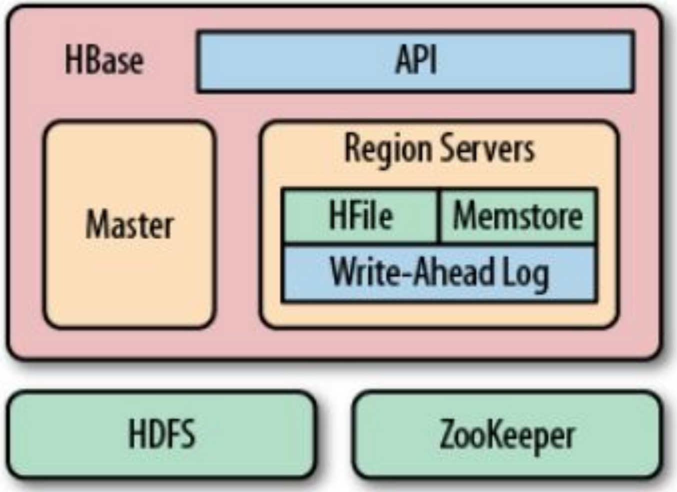
HBase: Data Model

- **Tables** are sorted maps of rows.
- **Each row has a primary row key** which used to sort the rows in the table.
- Each **row has a set of column families** (group of columns) which are specified in the schema.
- **Columns can be added on the fly**.
- **Cells** (row, column combination) are byte[] and timestamped. NULL values don't take up space.



HBase Architecture

- **Master Nodes**
  - JobTracker
  - HBase Master
  - NameNode
- **Slave Nodes**
  - TaskTracker
  - HBase RegionServer
  - DataNode
- **ZooKeeper**



HBase Components

HBase Master

- » Controls which Regions are served by which Region Servers.
- » Manages Region splits when they grow too much.
- » Assigns regions to new region servers when they arrive or fail.
- » A standby Master can be created which will transition to active master if Zookeeper determines the active one has failed.

Zookeeper

- » It is a highly available system for coordination.
- » Hosts the location of the root catalog table and the address of the current cluster Master.
- » Mediates in the assignment of regions.
- » Contains additional information about the cluster like servers location.

HBase Components

- **Tables are composed of one or more Regions.**
- **RegionServer:**
    - Can serve multiple regions but regions are only served one at a time.
    - Regions are **collocated** with HDFS DataNodes to take advantage of data locality.
- When Data is written to a Region three components come into play:
    - **HLog:** a **Write Ahead Log** that stores the operations sequentially.
    - **MemStore:** an LSM-Tree stored in memory that orders the last inserts.
    - **HFiles**
        - A file that is generated after the MemStore exceeds certain size and is flushed to disk.
        - When the amount of HFiles that compose a Region become too much a **compaction process** is executed to merge all the files into one. (this process is named as minor compaction).

HBase: read and delete

When a client asks for a row in HBase it triggers the search in the order:

» Search in the MemStore.
» If the search fails, search in the Region Server's Block cache (an LRU cache of rows).
» If the search fails, load the HFiles in memory and search through each one.

The **delete process writes a "tombstone" record** which marks that record as not eligible for read operations. **The only time** where **data** for the record **is actually erased** is **during a major compaction**.

HBase: Shell

Hbase provee un shell para correr comandos que se invoca mediante a

```
$> hbase shell
```

```
$> create ‘<table name>’, ‘<column family>’ … ;CREATE TABLE
$> put ‘<table name>’, ‘row_id’, ‘<colfamily:colname>’,‘<value>’ ;Insert & Update
$> enable ‘<table name>’;  disable ‘<table name> ;Enable, Disable:
$> drop '<table name>' ;Drop Table (should be disabled)
$> list; describe ‘<table name>’ ;Table lookup:
$> get ‘<table name>’,  ‘row_id’  ;Reading data:
$> delete ‘<table name>’, ‘<row>’, ‘<column name >’, ‘<time stamp>’ ;Delete Data
$> deleteall ‘<table name>’, ‘<row>’ ;Delete Data
```

more commands

Hbase Example of use

```
$> create 'empl', 'personal data', 'professional data'
$> put 'empl','1','personal data:name','mark'
$> put 'empl','1','personal data:city','ohio'
$> put 'empl','1','professional data:designation','manager'
$> put 'empl','1','professional data:salary','50000'
$> put 'empl','2','personal data:name','john'
$> put 'empl','2','personal data:city','nyc'
$> put 'empl','2','professional data:designation','typist'
$> put 'empl','2','professional data:salary','40000'
$> get 'empl', '1', 'personal data:name'
$> delete 'empl', '2', 'professional data:designation'
$> delete 'empl', '1'
```

HBase: Clients

- **Facebook** initially developed Apache Cassandra a close competitor to HBase. After the initial version of t**heir chat system they switched to HBase**.
- **OpenTSDB** is a massively scalable, open source, time series database that is implemented over HBase.
- **Pinterest** implements it's following feed using HBase as a backend for data storage.
- The **Apache Slider** project enables the use of **HBase over YARN.**

HBase: Apache Phoenix

- **Relational layer over HBase** which provides the ability to query HBase using low latency SQL queries.
- Tables are created with SQL DDL statements and Phoenix translates this to the HBase equivalent.
- Several enterprises use Phoenix for **sub second latency analytical queries** over billions of rows.
- Queries are compiled to parallel scans to HBase tables.

HBase Summary

Advantages

- **Consistent** reads/writes.
- Automatic **sharding**.
- Automatic **RegionServer failover and HTable compaction**.
- HDFS integration and MapReduce support: maps are handed a single region to work on.
- Java Client and Thrift/REST APIs.

Disadvantages

- Does not support **SQL** (there are projects like Apache Phoenix that remedy this).
- Does not support **transactions**.
- **Requires** having a **Hadoop** cluster and **Zookeeper**.
- **Tendency for hot spots** if not correctly managed.

HBase: references

Book

- “HBase: The Definitive Guide”, Lars George, First Edition, 2011.

Online

- http://hbase.apache.org/book.html
- http://www.tutorialspoint.com/hbase
- http://www.dbms2.com/2015/03/10/notes-on-hbase/

cassandra

What is NoSQL?

- » Developed at Facebook to power their Inbox Search feature.
- » Data model base on **Google Big Table paper**.
- » Distribution characteristics based on **Amazon's DynamoDB paper**.
- » **Top level Apache** project since **2010**.

Cassandra - A Decentralized Structured Storage System

Avinash Lakshman
Facebook

Prashant Malik
Facebook

ABSTRACT

Cassandra is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure. Cassandra aims to run on top of an infrastructure of hundreds of nodes (possibly spread across different data centers). At this scale, small and large components fail continuously. The way Cassandra manages the persistent state in the face of these failures drives the reliability and scalability of the software systems relying on this service. While in many ways Cassandra resembles a database and shares many design and implementation strategies therewith, Cassandra does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format. Cassandra system was designed to run on cheap commodity hardware and handle high write throughput while not sacrificing read efficiency.

1. INTRODUCTION

Facebook runs the largest social networking platform that serves hundreds of millions users at peak times using tens of thousands of servers located in many data centers around the world. There are strict operational requirements on Facebook's platform in terms of performance, reliability and efficiency, and to support *continuous growth* the platform needs to be highly scalable. Dealing with failures in an infrastructure comprised of thousands of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such, the software systems need to be constructed in a manner that treats failures as the norm rather than the exception. To meet the reliability and scalability needs described above Facebook has developed Cassandra.

Cassandra uses a synthesis of well known techniques to achieve scalability and availability. Cassandra was designed to fulfill the storage needs of the Inbox Search problem. Inbox Search is a feature that enables users to search through their Facebook Inbox. At Facebook this meant the system was required to handle a very high write throughput, billions of writes per day, and also scale with the number of users. Since users are served from data centers that are geographically distributed, being able to replicate data across data centers was key to keep search latencies down. Inbox Search was launched in June of 2008 for around 100 million users and today we are at over 250 million users and Cassandra has kept up the promise so far. Cassandra is now deployed as the backend storage system for multiple services within Facebook.

This paper is structured as follows. Section 2 talks about related work, some of which has been very influential on our design. Section 3 presents the data model in more detail. Section 4 presents the overview of the client API. Section 5 presents the system design and the distributed algorithms that make Cassandra work. Section 6 details the experiences of making Cassandra work and refinements to improve performance. In Section 6.1 we describe how one of the applications in the Facebook platform uses Cassandra. Finally Section 7 concludes with future work on Cassandra.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$10.00.

2. RELATED WORK

Distributing data for performance, availability and durability has been widely studied in the file system and database communities. Compared to P2P storage systems that only support flat namespaces, distributed file systems typically support hierarchical namespaces. Systems like Ficus[14] and Coda[16] replicate files for high availability at the expense of consistency. Update conflicts are typically managed using specialized conflict resolution procedures. Farsite[2] is a distributed file system that does not use any centralized server. Farsite achieves high availability and scalability using replication. The Google File System (GFS)[9] is another distributed file system built for hosting the state of Google's internal applications. GFS uses a simple design with a single master server for hosting the entire metadata and where the data is split into chunks and stored in chunk servers. However the GFS master is now made fault tolerant using the Chubby[3] abstraction. Bayou[18] is a distributed relational database system that allows disconnected operations and provides eventual data consistency. Among these systems, Bayou, Coda and Ficus allow disconnected operations and are resilient to issues such as network partitions and outages. These systems differ on their conflict resolution procedures. For instance, Coda and Ficus perform system level conflict resolution and Bayou allows application level

Cassandra: Key Features

- **Decentralized**: every node in the cluster has the same role (NO SPOF)
- **Linear scalability** of read and writes when increasing the cluster size.
- **Tunable consistency** levels.
- **Multi datacenter** replication.
- **Commercial support** by Datastax (DSE: Datastax Enterprise).

CQL = Cassandra Query Language

```
CREATE KEYSPACE devices

WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};

USE devices;
```

```
CREATE TABLE device_events (
  deviceID int,
  time int,
  event_count int
  PRIMARY KEY (deviceID, time)
);
```

```
SELECT event_count

FROM device_events

WHERE device_id = 1

      AND time > '2011-02-03'

      AND time <= '2012-01-01';
```

No support for JOINS, LIKE, Subqueries, Aggregations.

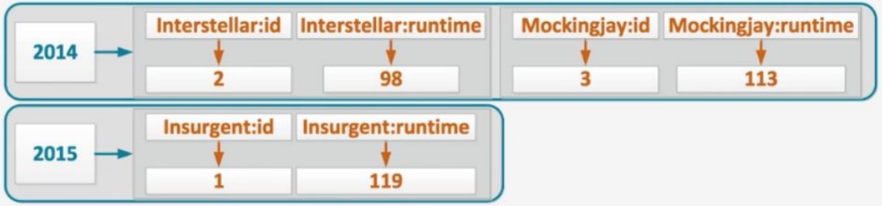
WHERE is only supported with the Primary KEY (Partition Key, Clustering Key) (unless ALLOW FILTERING is specified)

Cassandra: Data Model

Side by Side Comparison

PRIMARY KEY((id))

PRIMARY KEY((year), title)

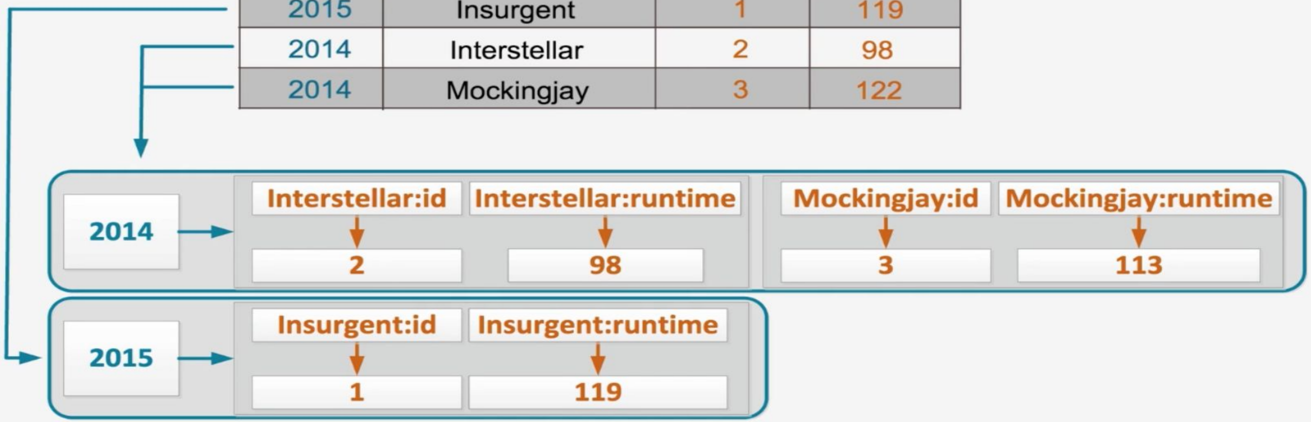


Cassandra: Data Model

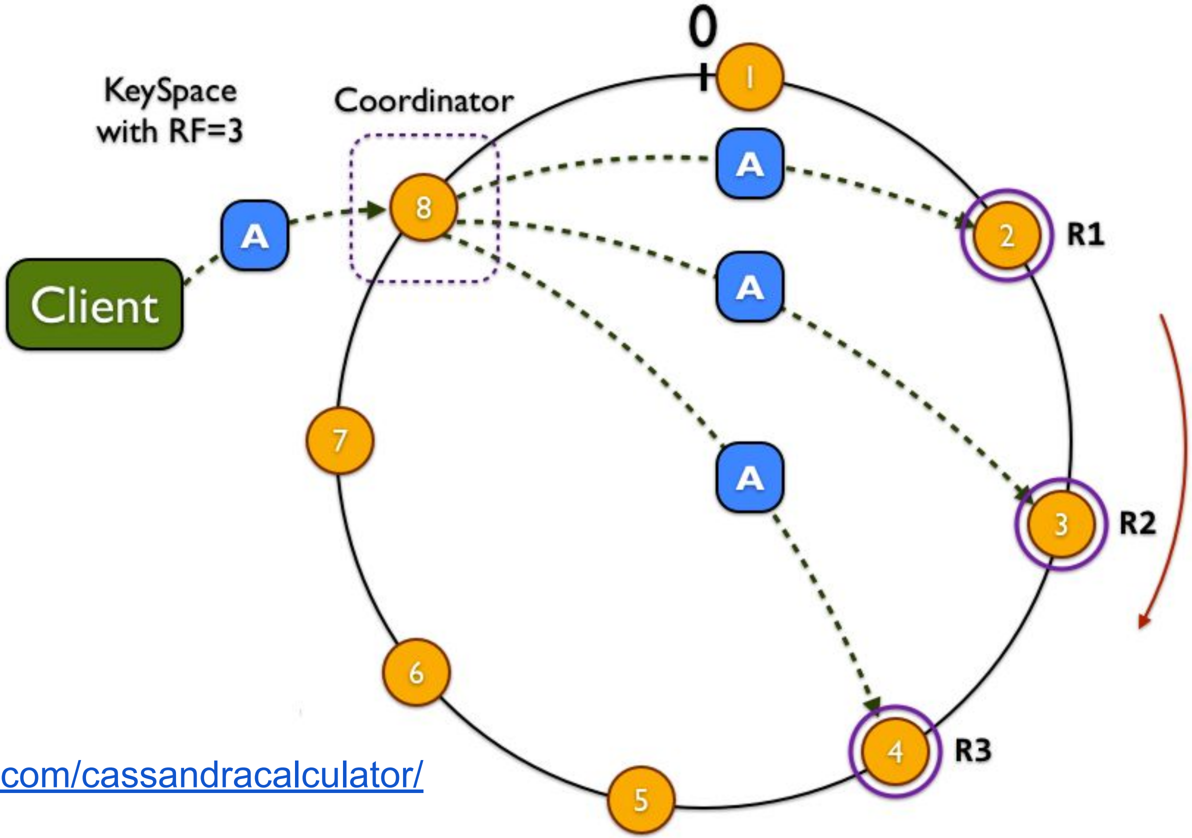
Clustering Columns

- Clustering columns divide CQL rows between partitions

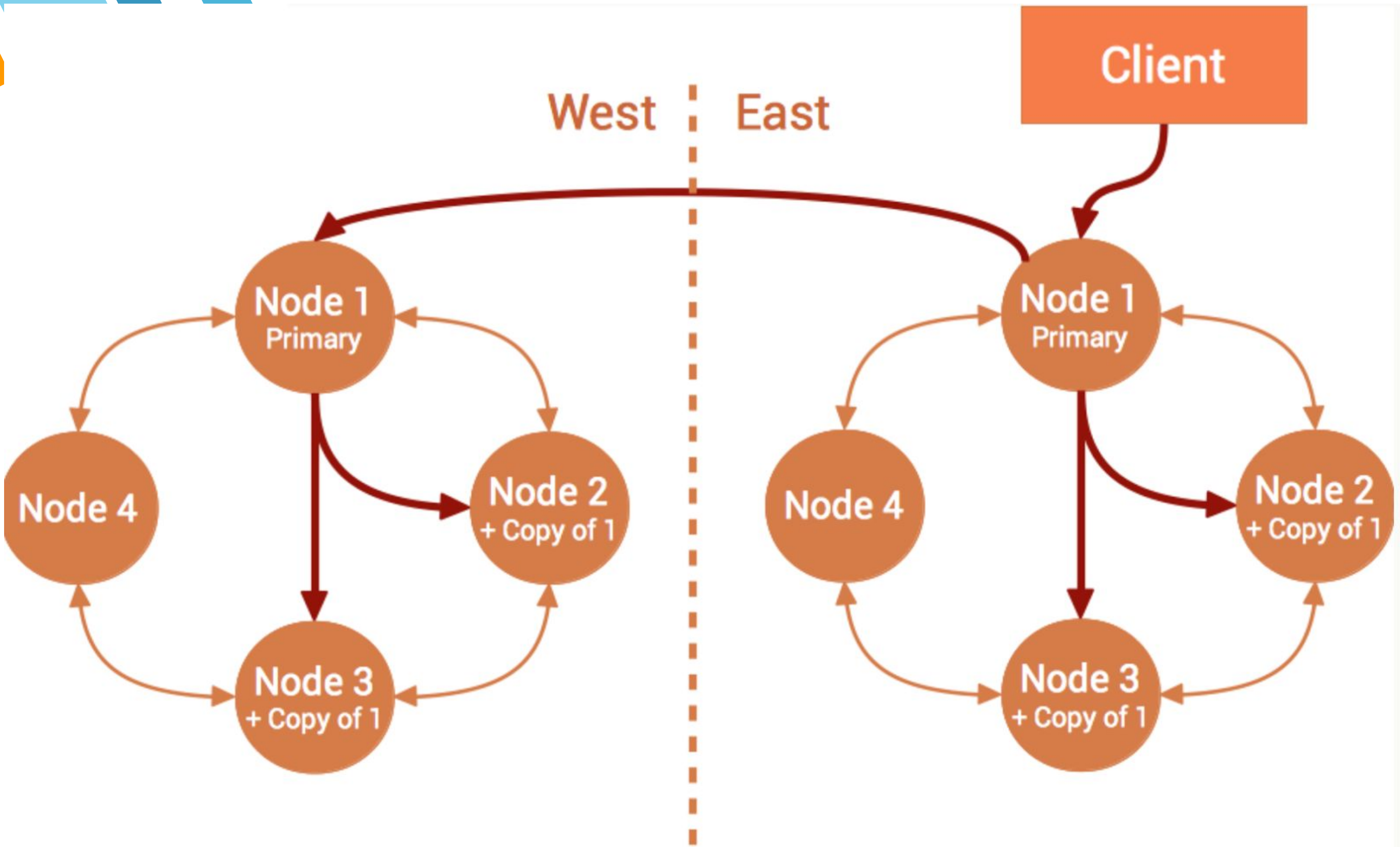
year	name	id	runtime
2015	Insurgent	1	119
2014	Interstellar	2	98
2014	Mockingjay	3	122



Cassandra: Data Partitioning



http://www.ecyrd.com/cassandracalculator/

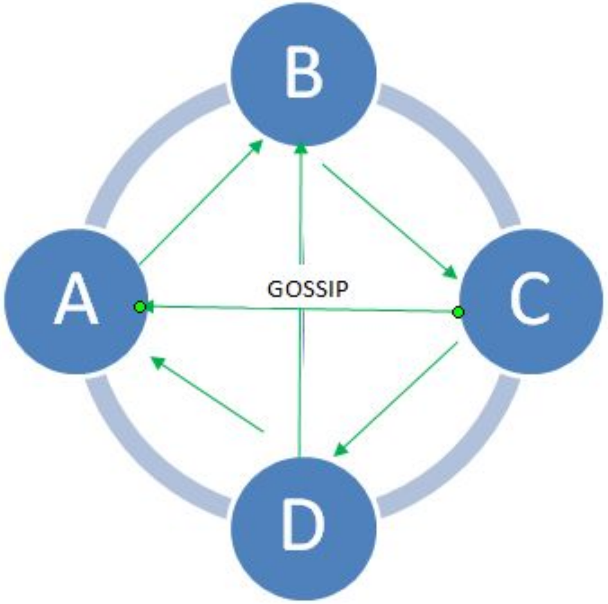

Client
West
East
Node 1
Primary
Node 4
Node 2
+ Copy of 1
Node 3
+ Copy of 1
Node 1
Primary
Node 4
Node 2
+ Copy of 1
Node 3
+ Copy of 1

Cassandra: consistency

- **ALL**: all replicas have to ACK
- **QUORUM**: > 51% of replicas should ACK.
- **LOCAL_QUORUM**: > 51% of replicas in local DC should ACK
- **EACH_QUORUM**: > 51% of replicas on ALL DC should ACK
- **ONE**: only one replica should ACK
- **TWO**: two replicas should ACK
- **THREE**: three replicas should ACK
- **LOCAL_ONE**: only one replica in local DC should ACK

Cassandra: Gossip Protocol

- **Nodes periodically exchange state information** about themselves and about other nodes they know about.
- **Runs every second** and exchanges state messages with up to three other nodes in the cluster.
- Gossip messages have version numbers so new information can be differentiated.



Cassandra: references

» http://www.slideshare.net/patrickmcfadin/introduction-to-cassandra-2014
» http://www.slideshare.net/jaykumarpatel/cassandra-data-modeling-best-practices
» https://cassandra.apache.org/doc/cql3/CQL.html

MPP over HDFS

MPP over HDFS

- **Open-source massively parallel distributed SQL query engines** for running interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes.
- **Based on Google Dremel** a scalable, interactive ad-hoc query system for analysis of read-only nested data. By **combining multi-level execution trees and columnar data layout**, it is capable of running aggregation queries over trillion-row tables in seconds.
- **Do not translate queries to MapReduce.**
- **Small tolerance for task failures.**
- **Data is read-only.**
- **Parquet** is becoming the defacto standard for columnar data storage format.
- Try to break the false choice between having fast analytics using an expensive commercial solution or using a slow "free" solution that requires excessive hardware.

MPP over HDFS

Cloudera Impala:

- Enables users to issue low-latency SQL queries to data stored in HDFS and Apache HBase without requiring data movement or transformation.
- Integrates with Hive metastore.

Facebook Presto:

- Can combine data from multiple sources (HDFS,PostgreSQL, MySQL, Cassandra, Hive, Apache Kafka, etc.)
- Has predicate push down capabilities.
- Has support for functions that calculate approximate values.

Document Oriented Store

Document Oriented Store

- Is usually a specialization of a key value store where the **value is a "document"** (a complex object or all the rows in a table).
- The main difference with key value is that the document is not opaque and **we can query by and update its components**.
- The documents stored do **not need to comply with a given schema** (each record could be different).
- This make document oriented stores **great for semistructured data.**

Benefits

- **Documents are independent units** which makes performance better (related data is read contiguously off disk) and makes it easier to distribute data across multiple servers while preserving its locality.
- **Application logic is easier to write**. You don't have to translate between objects in your application and SQL queries, you can just turn the object model directly into a document.
- **Unstructured data can be stored easily**, since a document contains whatever keys and values the application logic requires. In addition, costly migrations are avoided since the database does not need to know its information schema in advance.

mongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

MongoDB - Key Features

- **High Performance**: MongoDB provides high performance data persistence.
- **Rich Query Language**: to support read and write operations (CRUD) as well as Data Aggregation, Text Search and Geospatial Queries.
- **High Availability:** MongoDB's replication facility, called replica set, provides: automatic failover and data redundancy.
- **Horizontal Scalability**: provides horizontal scalability as part of its core functionality.
- **supports multiple storage engines**, such as:
  - WiredTiger Storage Engine
  - MMAPv1 Storage Engine.

In addition, MongoDB provides pluggable storage engine API that allows third parties to develop storage engines for MongoDB.

MongoDB Ecosystem

To work with MongoDB there is an ecosystem provided by the company and the community

- Drivers written in several languages: C, C++, C#, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Go, Erlang.
- There are several integrations: Hadoop, Admin UI, HTTP interface
- And it can be deployed both self hosted or as a service in clouds like: Amazon, Azure, etc.

MongoDB Documents

MongoDB documents are composed of field-and-value pairs and have the following structure:

```
{
   field1: value1,
   field2: value2,
   field3: value3,
   ...
   fieldN: valueN
}
```

```
var mydoc = {
        _id: ObjectId("5099803df3f4948bd2f98391"),
        name: { first: "Alan", last: "Turing" },
        birth: new Date('Jun 23, 1912'),
        death: new Date('Jun 07, 1954'),
        contribs: [ "Turing machine", "Turing test", "Turingery" ],
        views : NumberLong(1250000)
      }
```

MongoDB Documents

- The field name _id is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
- The field names cannot
  - start with the dollar sign ($) character.
  - The field names cannot contain the dot (.) character.
  - The field names cannot contain the null character.
- The **document type is BSON** and field types are one of the [BSON types]
- To access internal fields we use the dot notation (user.name).
- The maximum BSON document size is 16 megabytes.

MongoDB Collections and Databases

MongoDB stores BSON documents, i.e. data records, in collections; the collections in databases.

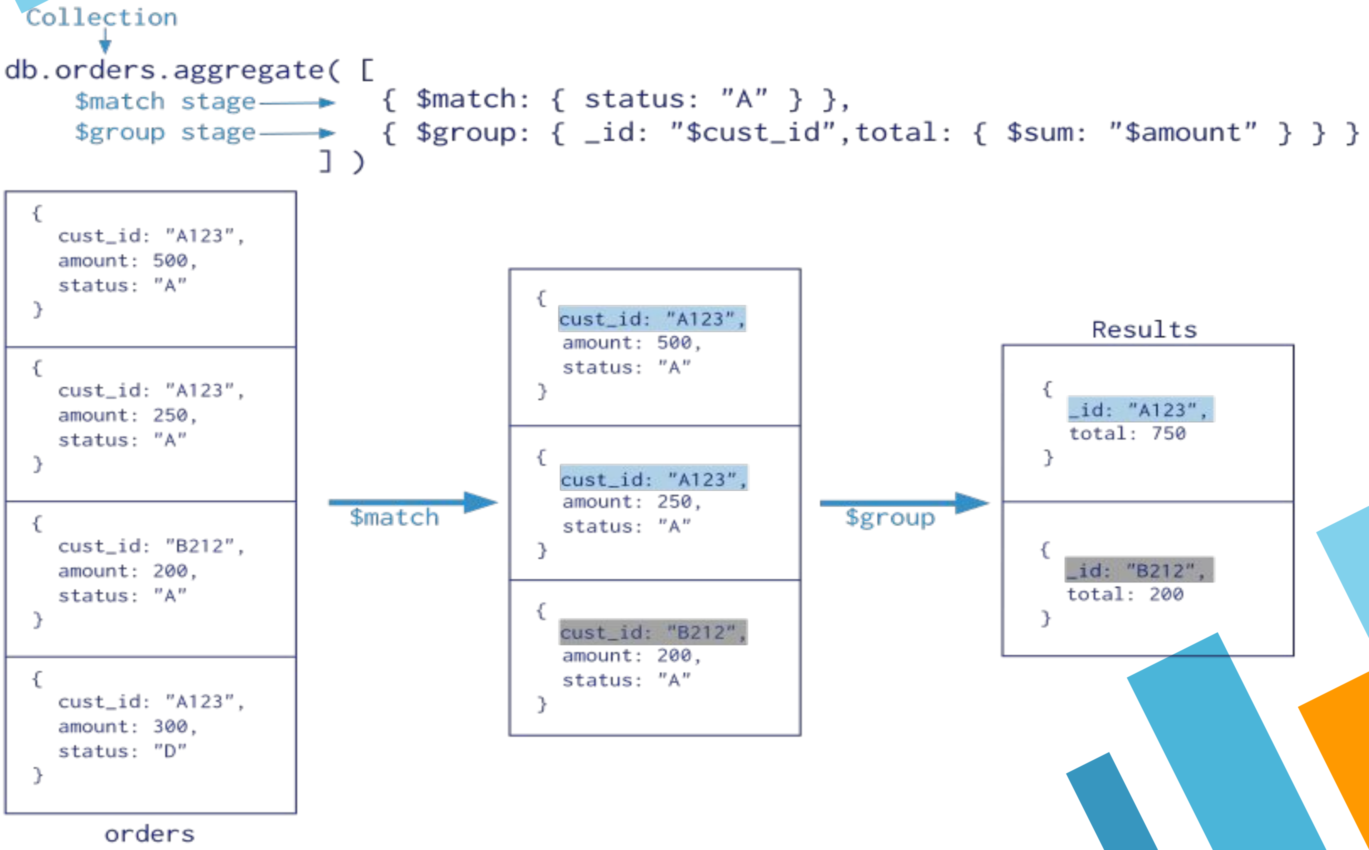
MongoDB Commands

coll = db.<collection>	Generates an alias for the collections
db.collection.find()	Find all documents in the collection and returns a cursor.
db.collection.insert()	Insert a new document into the collection.
db.collection.update()	Update an existing document in the collection.
db.collection.save()	Insert either a new document or update an existing document in the collection.
db.collection.remove()	Delete documents from the collection.
db.collection.drop()	Drops or removes completely the the collection.
db.collection.createIndex()	Create a new index on the collection if the index does not exist; otherwise, the operation has no effect.
db.dropDatabase()	drops the database

MongoDB Query

```
rs1:SECONDARY> db.accumStats.find({"fid":103585},{"fid" : 1, "d.tv":1} ).pretty()
{
        "_id" : ObjectId("57dee7129fc6a984f95e9beb"),
        "fid" : NumberLong(103585),
        "d" : {
                "tv" : {
                        "rps" : {
                                "t" : NumberLong(13),
                                "l" : NumberLong(0)
                        },
                        "l" : NumberLong(0),
                        "rts" : {
                                "t" : NumberLong(11),
                                "l" : NumberLong(0)
                        },
                        "s" : {
                                "neu" : NumberLong(0),
                                "unk" : NumberLong(54),
                                "pos" : NumberLong(0),
                                "neg" : NumberLong(0)
                        },
                        "t" : NumberLong(54),
                        "r" : NumberLong(140906),
                        "tpm" : 0,
                        "ltpm" : 0
                }
        }
}
```

MongoDB Aggregations



References

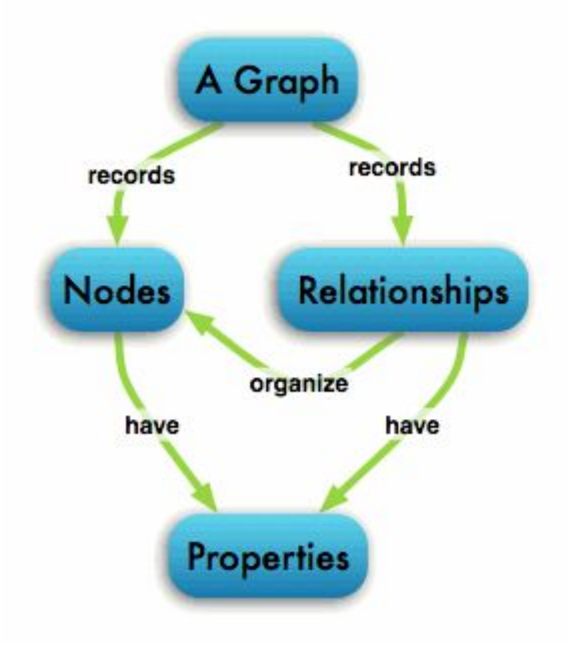
- Manual
- MongoDB University
- MongoDB Blog

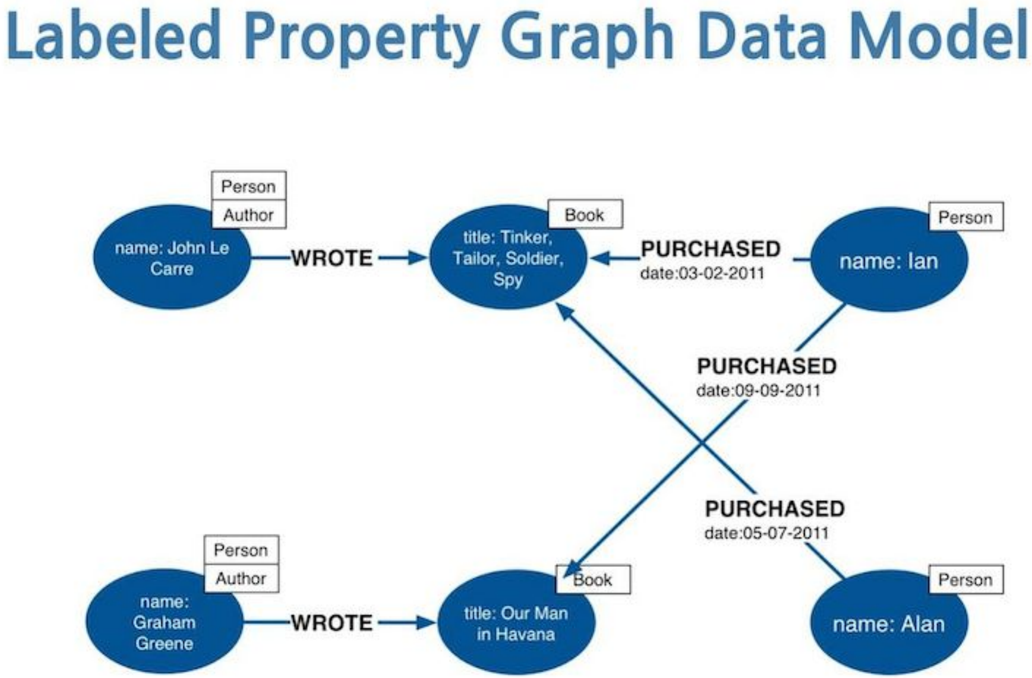
Graph Store

Graph Databases

- A graph database is a collection of nodes and edges.
- Each node represents an entity and each edge represents a connection or relationship between two nodes.
- Every node in a graph database is defined by a unique identifier, a set of outgoing edges and/or incoming edges and a set of properties expressed as key/value pairs.
- Each edge is defined by a unique identifier, a starting-place and/or ending-place node and a set of properties.

Graph Databases





Search As No SQL

Search as NoSQL

- Use **Apache Lucene** (created by Doug Cutting) at their core. An API for **information retrieval based on inverted index searches.**
- The two most important products are **Elasticsearch and SolrCloud.**
- Elasticsearch was created with **scalability and high availability** in mind.
- Elasticsearch provides a RESTful API out of the box and **direct access APIs to read and write data from Hadoop and Spark**.
- Provides flexible schemas for data (JSON) and the indexed data is immediately available for querying.

Elastic Search

» Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time.

» Elasticsearch is a search engine based on Lucene.

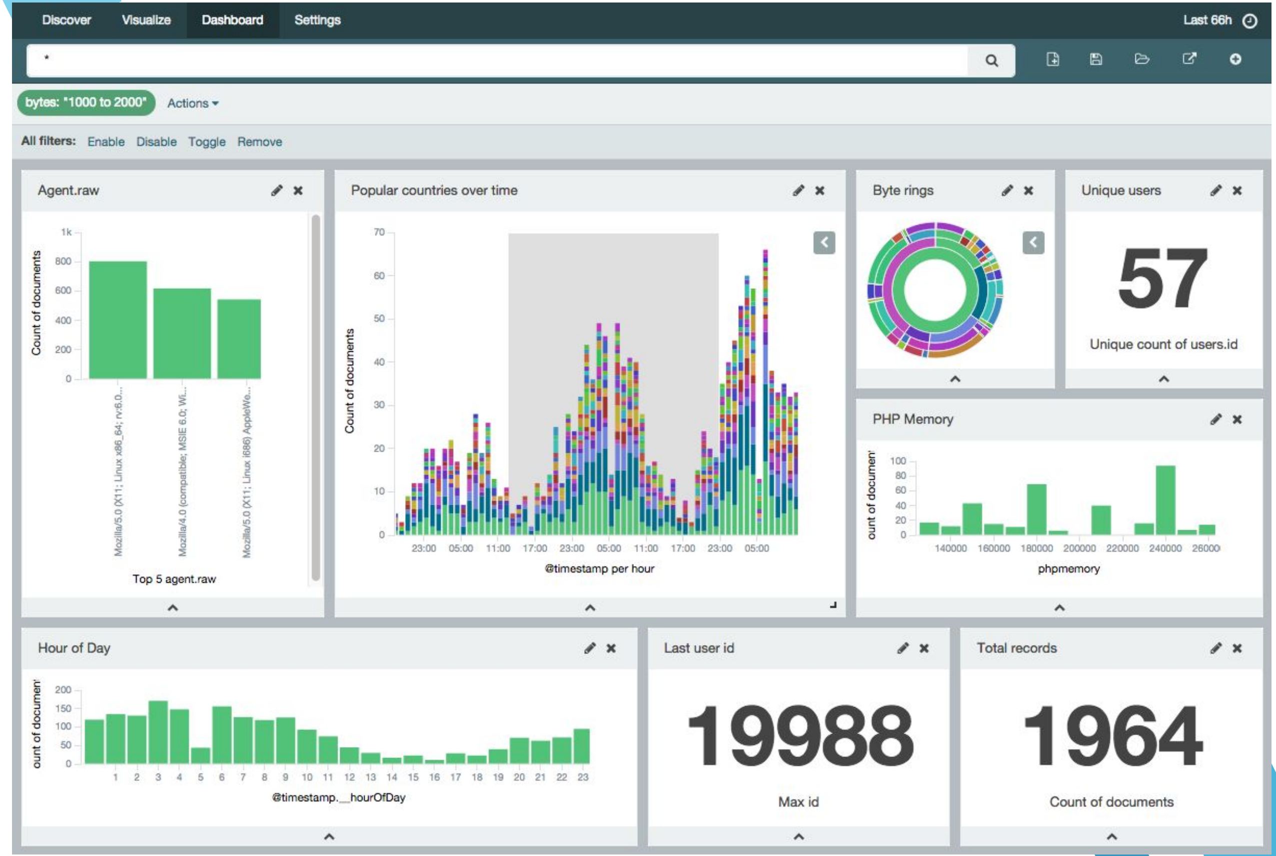
» It provides a full-text search engine

- distributed,
- multitenant-capable
- with an HTTP web interface
- schema-free JSON documents.

ELK Stack

- **Elastic as a company** is positioning Elasticsearch, Logstash and Kibana as the defacto standard for real time event log analytics.
- **Logstash** is in charge of getting the applications logs to Elasticsearch for indexing.
- **Kibana** is a flexible analytics and visualization platform with the ability to generate custom dashboards and graphics with no programming involved.

ELK Stack



Discover
Visualize
Dashboard
Settings
Last 66h
bytes: "1000 to 2000"
Actions
All filters: Enable Disable Toggle Remove
Agent.raw
Count of documents
Top 5 agent.raw
Popular countries over time
Count of documents
@timestamp per hour
Byte rings
Unique users
57
Unique count of users.id
PHP Memory
phpmemory
Hour of Day
@timestamp.__hourOfDay
Last user id
19988
Max id
Total records
1964
Count of documents

New SQL

NewSQL

- They seek to provide the **same scalable performance of NoSQL systems for online transaction processing** (OLTP) read-write workloads while still **maintaining the ACID** guarantees of a traditional database system.
- Targeted towards organizations that need to scale and **can not give up strong transactional and consistency** guarantees.
- Not recommended for analytics use cases.

Key Take Aways

- NoSQL was **initially mostly pointed towards operational workloads** (Redis, HBase, etc.), now analytical workloads are also becoming important.
- What in **SQL** can be stored in **one table** with many indexes becomes **many tables in NoSQL** (one table for each index: query pattern).
- **NoSQL** stores are **simpler in their query capabilities and guarantees**, what makes them complex is their ability to scale horizontally and flexibly.
- With the current Big Data trends the **one-size fits all of RDBMs is no longer valid.**
- **NewSQL is SQL + Scalability** but they still don't cover all specialized use cases.
- We are in the **age of Polyglot Persistence**: a data-store for each specific use case.
- As the variety of data-stores increase in an enterprise, the need for an additional layer that abstracts away the implementation arises. Some people call this the **Logical Datawarehouse**.

CREDITS

Content of the slides:

- Big Data Tools - ITBA

Images:

- Big Data Tools - ITBA
- obtained from: commons.wikimedia.org

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash