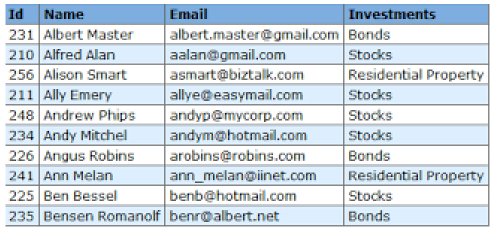
HBase

Random reads and Writes

NoSQL Database

Challenges with traditional database?

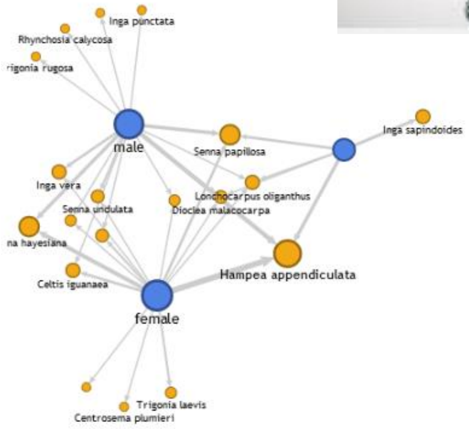
•Traditional databases are designed for holding structured data.

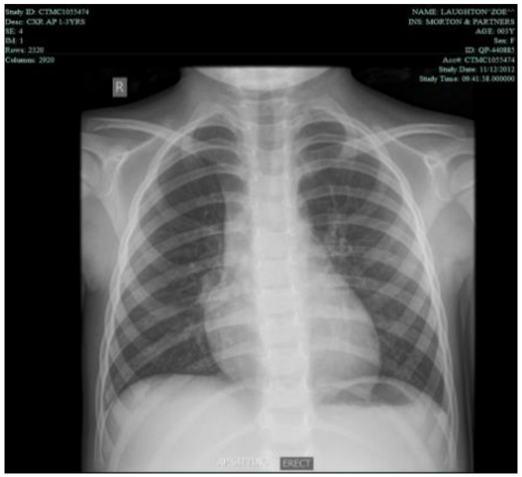


Id	Name	Email	Investments
231	Albert Master	albert.master@gmail.com	Bonds
210	Alfred Alan	aalan@gmail.com	Stocks
256	Alison Smart	asmart@biztalk.com	Residential Property
211	Ally Emery	allye@easymail.com	Stocks
248	Andrew Phips	andyp@mycorp.com	Stocks
234	Andy Mitchel	andym@hotmail.com	Stocks
226	Angus Robins	arobins@robins.com	Bonds
241	Ann Melan	ann_melan@iinet.com	Residential Property
225	Ben Bessel	benb@hotmail.com	Stocks
235	Bensen Romanolf	benr@albert.net	Bonds

•Can it hold the following data?

- Semi structured data
- Images
- Network graphs
- Documents
- Human-Genome data
- Health care multi media
- GPS tracker data
- HTML pages
- Facebook likes, groups
- Customer-Genome in banking, retail and any other sector
- NASA astronomy logs





- Think about this, can you create a table with 10000 columns? If yes, how will you manage it?
- You might assume, this can be stored on RDBMS. But at what scale and what cost?

How can you handle this semi and unstructured datasets?

Definitely Not RDBMS!!!

Rise of NoSQL Database

NoSQL databases

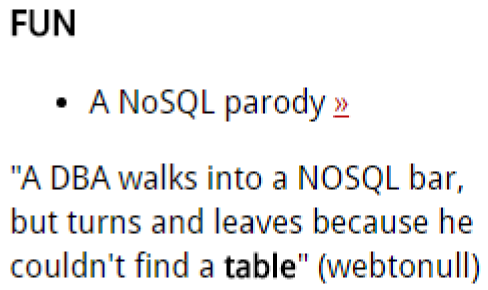
• NoSQL refers to Not Only SQL databases

•Next Generation Databases mostly addressing business problems; <u>some of the points</u>: being **non-relational, distributed, open-source** and **horizontally scalable**.

•The movement began early 2009 and is growing rapidly.

•Main Characteristics:

- Schema free
- Easy replication support
- Simple API
- Eventually Consistent(Referred as BASE – Basically Available Eventually Consistent properties)
- Huge Amount data

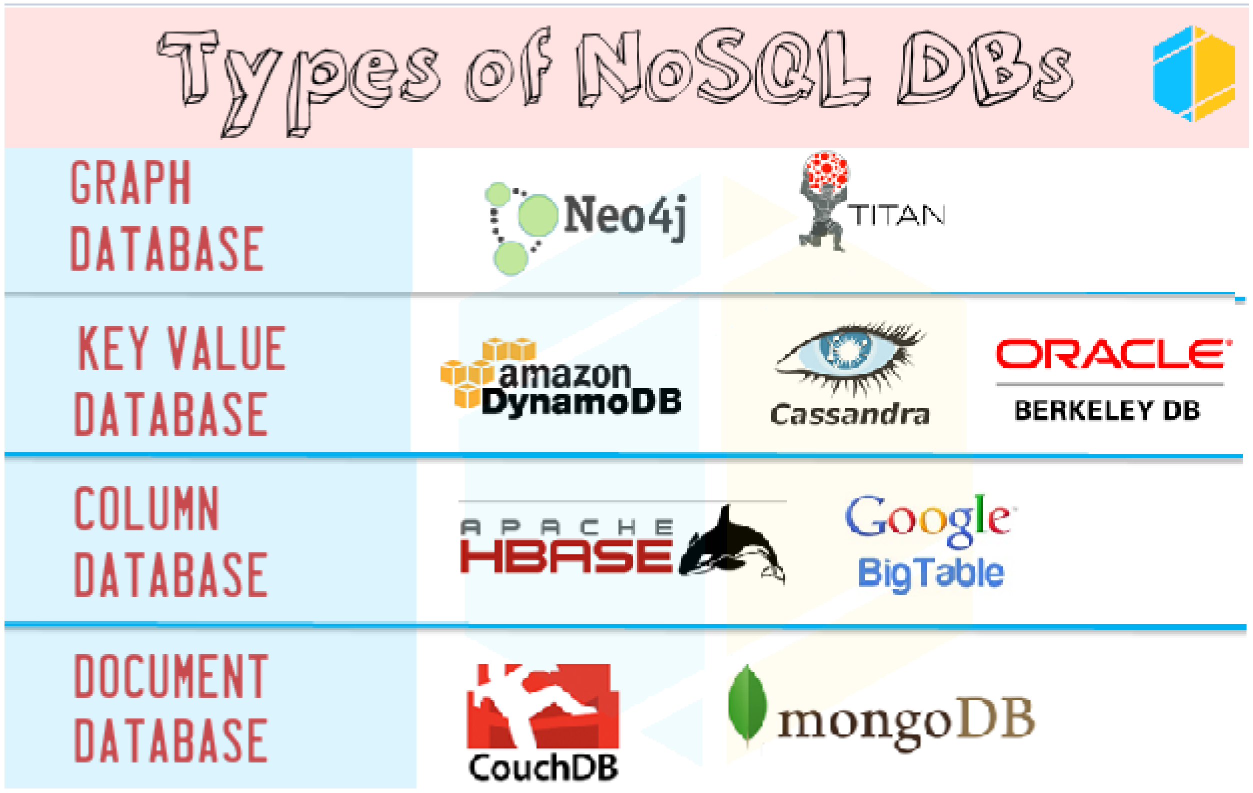


FUN

- A NoSQL parody »

"A DBA walks into a NOSQL bar, but turns and leaves because he couldn't find a **table**" (webtonull)

Types of NoSQL databases



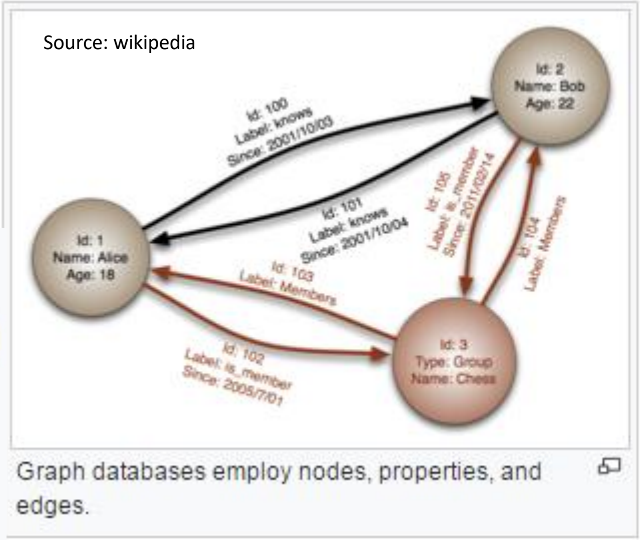
Types of NoSQL DBs

GRAPH DATABASE	Neo4j	TITAN	
KEY VALUE DATABASE	amazon DynamoDB	Cassandra	ORACLE BERKELEY DB
COLUMN DATABASE	APACHE HBASE	Google BigTable	
DOCUMENT DATABASE	CouchDB	mongoDB	

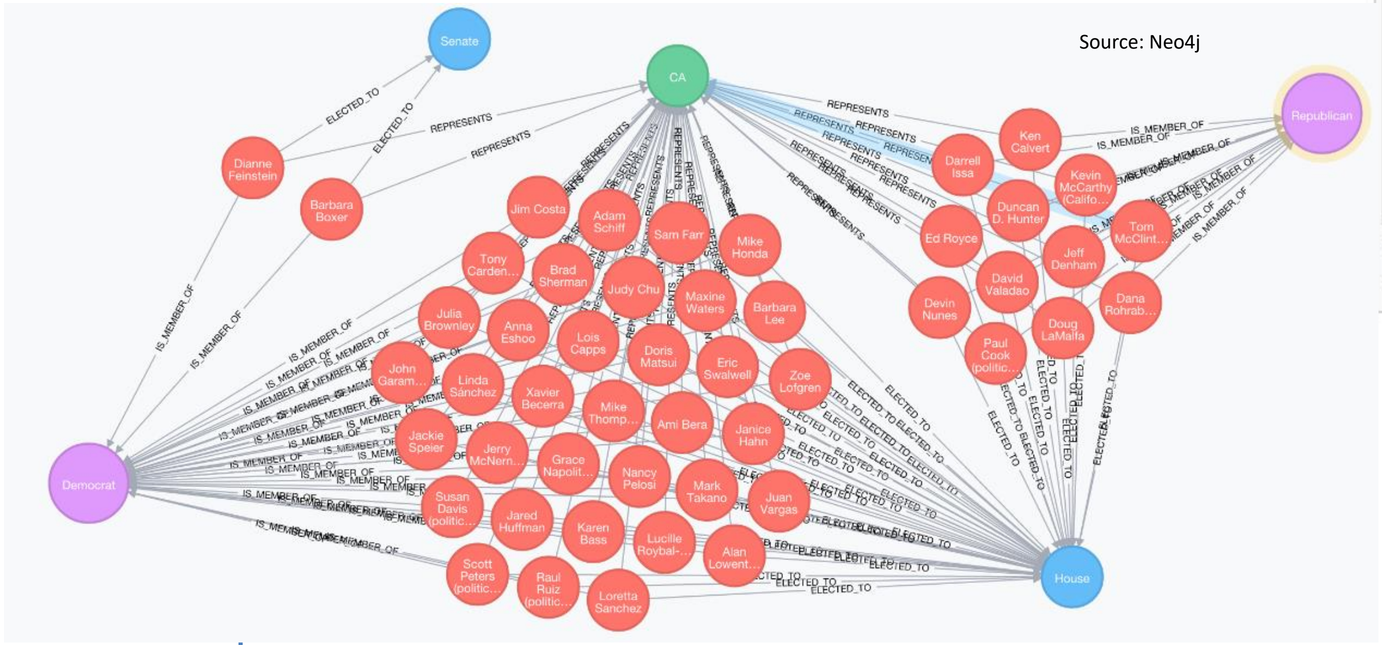
Reference : http://www.vikramtakkar.com/2015/12/nosql-types-of-nosql-database-part-2.html

Graph Databases

- A database, which uses graph structures for semantic queries with nodes, edges
- Graph databases employ nodes, edges and properties:
    - Nodes: Represent entities such as people, businesses, accounts, or any other item you might want to keep track of.
    - Edges: Relationship between nodes
    - Properties: Refers to characteristics of nodes
- Neo4j is one of the leading graph databases.



Graph databases employ nodes, properties, and edges.



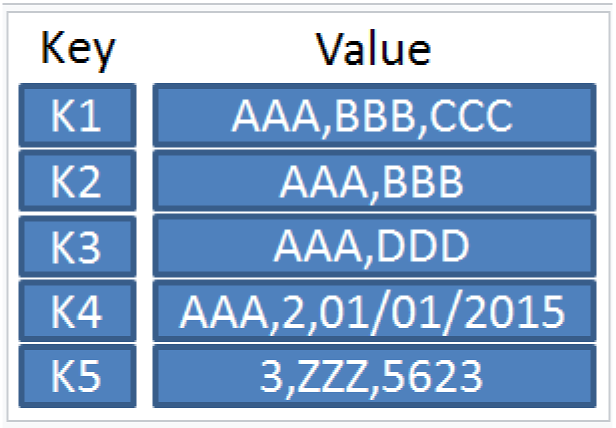
Key value databases

•A key value store, is a simple DB where each key is associated with one and only one value in collection.

•Values can hold any complex data, and can vary from record to record as it is schema free.

•Key Features:

- Scalability and Reliability
- Portability and Lower Operational Costs



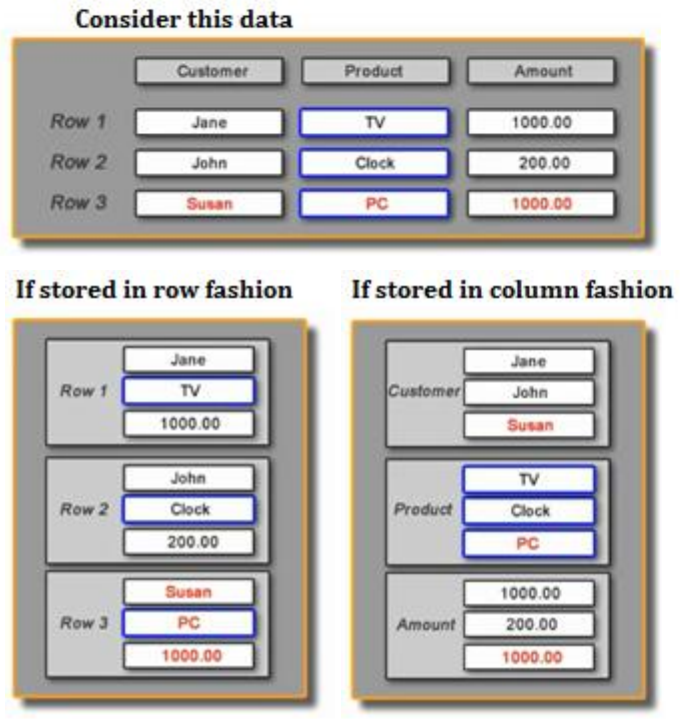
Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Source: Wikipedia

Column Databases

• Column oriented databases store tables in columnar format, rather than rows.

•Column oriented databases are faster in access compared to row oriented like RDBMS, as data retrieval becomes easy when stored as separate columns.



Now to get product column, in row oriented databases, all 3 columns will be scanned in each row to retrieve Product column

In column oriented database, each column is stored seperately, so Product column can be directly accessed, which is 100 times faster than general RDBMS.

Document database

•A **document**-oriented **database** will be the most suitable for Semi-structured data which is in document –oriented format.

•MongoDB is the leading database.

•Example : Aadhar card.

What is HBase?

- HBase is the Hadoop database modeled after Google's Bigtable
  - "A Bigtable is a **sparse**, **distributed**, **persistent multidimensional sorted map**"

- Use HBase for random, real-time read/write access to your BigData
  - Runs on clusters of commodity hardware

- HBase and its native API is written in Java, but you do not have to use Java to access its API.

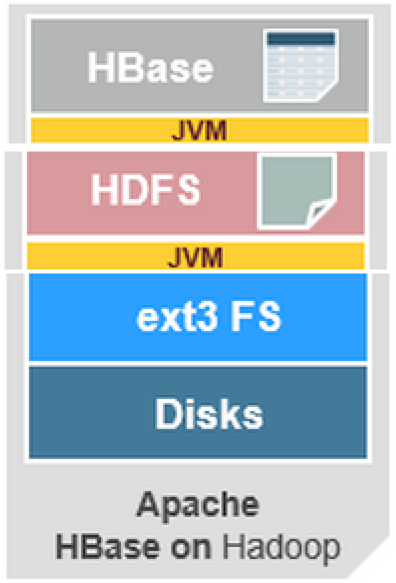
When you have HDFS, why HBase ?

•HDFS is suited for Batch processing applications.

•HDFS is write once, read many times.

•HDFS doesn't suit for applications where random reads and writes are required.

•HBase is known for random reads and writes, and is best suited for OLAP applications.



https://www.mapr.com/blog/in-depth-look-hbase-architecture

When you have Hive, why HBase ?

•Hive is known for Data warehousing applications, and is built on top of HDFS for storage and Map Reduce for Processing.

•Since HDFS doesn't support random reads and writes, so hive doesn’t support.

•Hive can query data on HDFS.

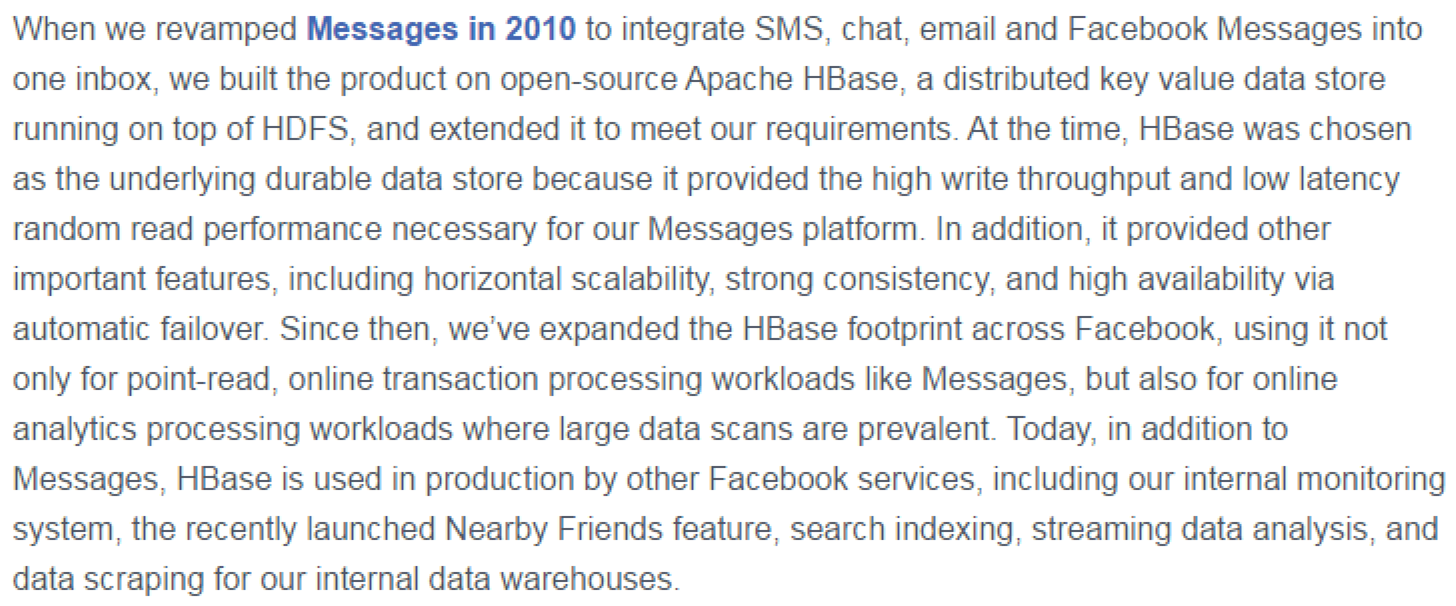
•HBase is a NOSQL database, designed on top of HDFS for random reads/writes.

•HBase doesn’t have any querying interface, now Hive comes to the rescue.

•Hive can enable access to HBase through HQL.

Create external tables to access data from HBase

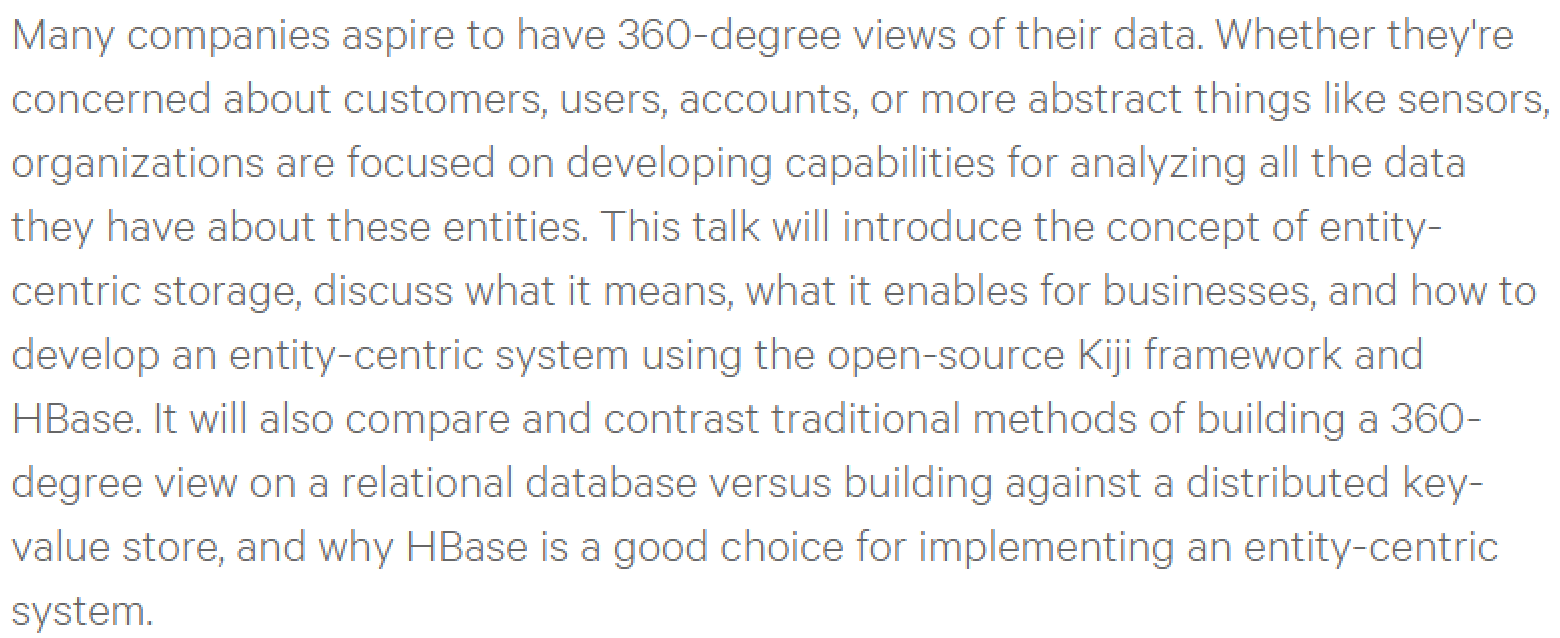
Casestudy1 – Facebook



When we revamped **Messages in 2010** to integrate SMS, chat, email and Facebook Messages into one inbox, we built the product on open-source Apache HBase, a distributed key value data store running on top of HDFS, and extended it to meet our requirements. At the time, HBase was chosen as the underlying durable data store because it provided the high write throughput and low latency random read performance necessary for our Messages platform. In addition, it provided other important features, including horizontal scalability, strong consistency, and high availability via automatic failover. Since then, we've expanded the HBase footprint across Facebook, using it not only for point-read, online transaction processing workloads like Messages, but also for online analytics processing workloads where large data scans are prevalent. Today, in addition to Messages, HBase is used in production by other Facebook services, including our internal monitoring system, the recently launched Nearby Friends feature, search indexing, streaming data analysis, and data scraping for our internal data warehouses.

https://code.facebook.com/posts/321111638043166/hydrabase-the-evolution-of-HBase -facebook/

Casestudy2 - design-patterns-for-building-360-degree-views-with-HBase



Many companies aspire to have 360-degree views of their data. Whether they're concerned about customers, users, accounts, or more abstract things like sensors, organizations are focused on developing capabilities for analyzing all the data they have about these entities. This talk will introduce the concept of entity-centric storage, discuss what it means, what it enables for businesses, and how to develop an entity-centric system using the open-source Kiji framework and HBase. It will also compare and contrast traditional methods of building a 360-degree view on a relational database versus building against a distributed key-value store, and why HBase is a good choice for implementing an entity-centric system.

http://www.cloudera.com/content/dam/www/marketing/resources/events/HBase -con/2014/design-patterns-for-building-360-degree-views-with-HBase -and-kij.png.landing.html

Companies using HBase

Adobe

We currently have about 30 nodes running HDFS, Hadoop and HBase in clusters ranging from 5 to 14 nodes on both production and development. We plan a deployment on an 80 nodes cluster. We are using HBase in several areas from social services to structured data and processing for internal use. We constantly write data to HBase and run mapreduce jobs to process then store it back to HBase or external systems. Our production cluster has been running since Oct 2008.

Project Astro

Astro provides fast Spark SQL/DataFrame capabilities to HBase data, featuring super-efficient access to multi-dimensional HBase rows through native Spark execution in HBase coprocessor plus systematic and accurate partition pruning and predicate pushdown from arbitrarily complex data filtering logic. The batch load is optimized to run on the Spark execution engine. Note that Spark-SQL-on-HBase is the release site. Interested parties are free to make clones and claim to be "latest(and active)", but they are not endorsed by the owner.

Axibase Time Series Database (ATSD)

ATSD runs on top of HBase to collect, analyze and visualize time series data at scale. ATSD capabilities include optimized storage schema, built-in rule engine, forecasting algorithms (Holt-Winters and ARIMA) and next-generation graphics designed for high-frequency data. Primary use cases: IT infrastructure monitoring, data consolidation, operational historian in OPC environments.

Full list: https://HBase .apache.org/poweredbyHBase .html

NoSQL technology

- HBase is a NoSQL data store
  - NoSQL = "Not only SQL"
  - Not intended to replace RDBMS
  - Suited for specific business needs

- Traditional RDBMS is not very scalable when data reaches the terabytes and petabytes realm

  - Sharding occurs when data volume goes up too high.
  - Sharding becomes a major challenge with RBMS

    - Keeping referential integrity
    - Joins
    - Rebalancing

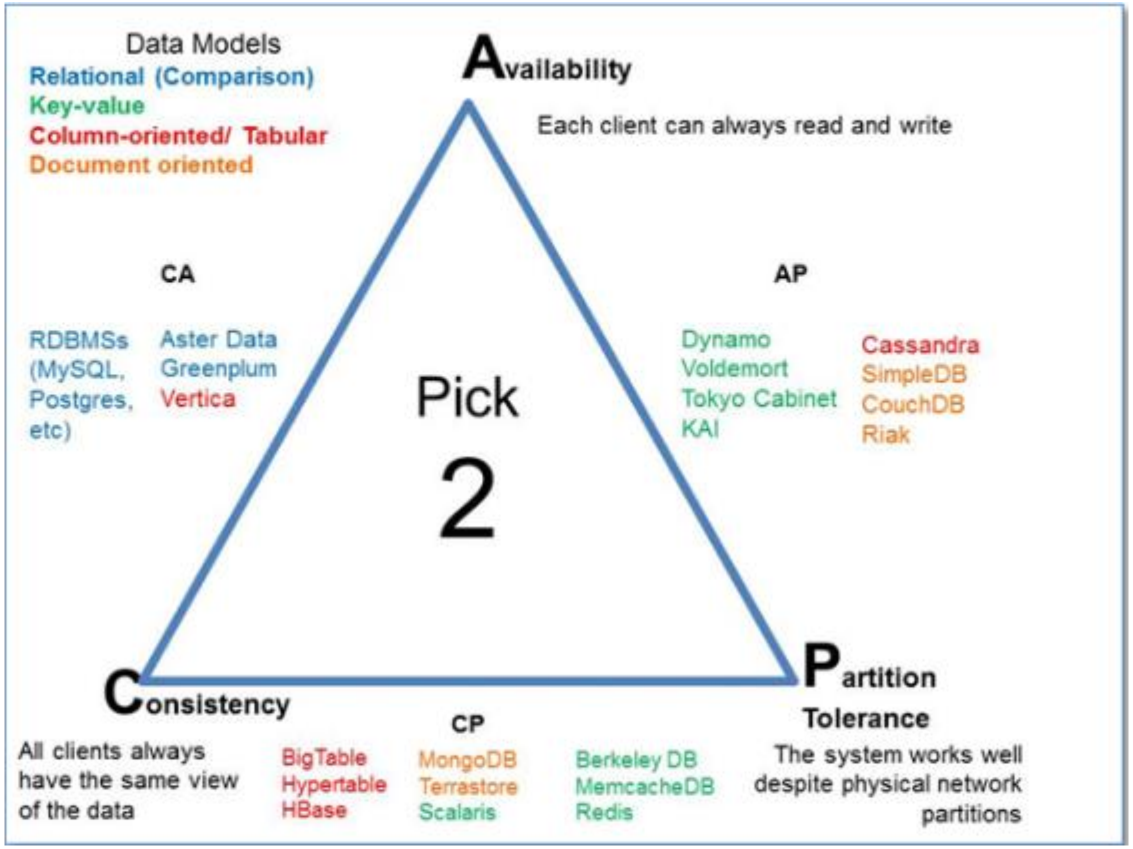
- A cost effective way to use commodity hardware

- Flexible data models are needed to support BigData applications
  - We won't always know the schema up front.
  - Records can be sparse (social media data is variable)

CAP theorem

- Consistency
  - All clients see the same data at the same time
- Availability
  - A guarantee that every request will see a response (success or fail)
- Partition Tolerance
  - The system will continue to operate even one parts of the system fails
- A distributed system can only guarantee two of the three CAP properties
- HBase is eventually consistent and implements consistency and partition tolerance.
  - For example, a Region Server failure is recoverable, but the data will be unavailable for a period of time

CAP theorem



Reference : http://blog.flux7.com/blogs/nosql/cap-theorem-why-does-it-matter

ACID Properties

- **Atomicity**: an operation is atomic if it either completes entirely or not at all
  - ➢Only provides row level atomicity
- Consistency & Isolation
  - ➢All actions cause the table to transition from one valid state to another
  - ➢Scans are not consistent
  - ➢A scan will always reflect the view of the data at least as new as the beginning of the scan
  - ➢Row-level consistency only. All rows returned will be a complete row that existed at some point into he table's history
- Durability: Any successful updates will not be lost
  - ➢Any operation that returns a success will be made durable

Why HBase

- Apache Hadoop includes HBase
  - ✓Allows HBase to leverage the Map/Reduce model

- HBase can replace costly implementation of RDBMS for Big Data applications
  - ✓HBase is not a one size fit all approach. It is designed for big data applications with
  - ✓specific data access patterns in mind. Not meant to replace RDBMS entirely!

- HBase enables horizontal scalability for very large data sets
  - ✓Easily scalable by adding additional commodity hardware without interruption.

- HBase supports flexible data models of sparse records
  - ✓We do not need to know the data types in advance. Keeps our schema very flexible.

- HBase supports random read/write access to the Big Data
  - ✓Very quick and efficient, random reads and writes

- Sharding is automatic without consequences from the RDBMS approach
  - ✓Built-in feature to HBase that can be customized

Important points

- HBase is not designed to replace RDBMS.
  - Does not support SQL
  - Not for transactional processing
  - Does not support table joins

- Not to be used as a general purpose database for Hadoop.
  - Use for specific cases where your data exhibits behavior supported by HBase(sparse data set, variable schemas, key-based access, partial dataset scans, etc.)

- Everything in HBase is stored as an array of bytes with the exception of the timestamp value, which is stored as a long integer.

HBase vs RDBMS

	HBase	RDBMS
Hardware Architecture	Similar to Hadoop. Clustered commodity hardware. Very affordable.	Typically large scalable multiprocessor systems. Very expensive.
Fault Tolerance	Built into the architecture. Lots of nodes means each is relatively insignificant. No need to worry about individual node downtime.	Requires configuration of the HW and RDBMS with the appropriate high availiable options
Typical Database Size	Terabytes to Petabytes – hundred of millions to billions of rows.	Gigabytes to Terabytes – hundred to thousands to millions of rows.
Data Layout	A sparse, distributed, persistent, multidimensional sorted map.	Rows or column oriented.
Data Types	Bytes only.	Rich data type support.
Transactions	ACID support on a single row only	Full ACID compliance across rows and labels
Query Language	API primitive commands only, unless combined with Hive or other technology	SQL
Indexes	Row-Key only unless combined with other technologies such as Hive or IBM's BigSQL	Yes
Throughput	Millions of queries per second	Thousands of queries per second

Example

- **Given this RDBMS:**

ID (Primary key)	Last Name	First Name	Password	Timestamp
1234	Smith	John	Hello, world!	20130710
5678	Cooper	Joyce	Wysiwyg	20120825
5678	Cooper	Joyce	Wisiwig	20130916

- **Logical view in HBase:**

Row-Key	Value (CF, Qualifier, Version)

1234	info {'lastName': 'Smith', 'firstName': 'John'}
	pwd {'password': 'Hello, world!'}

5678	info {'lastName': 'Cooper': 'firstName': 'Joyce'}
	pwd {'password: 'wysiwyg'@ts 20130916,
	'password': 'wisiwig'@ts 20120825}

Reference : IBM

Physical view in HBase

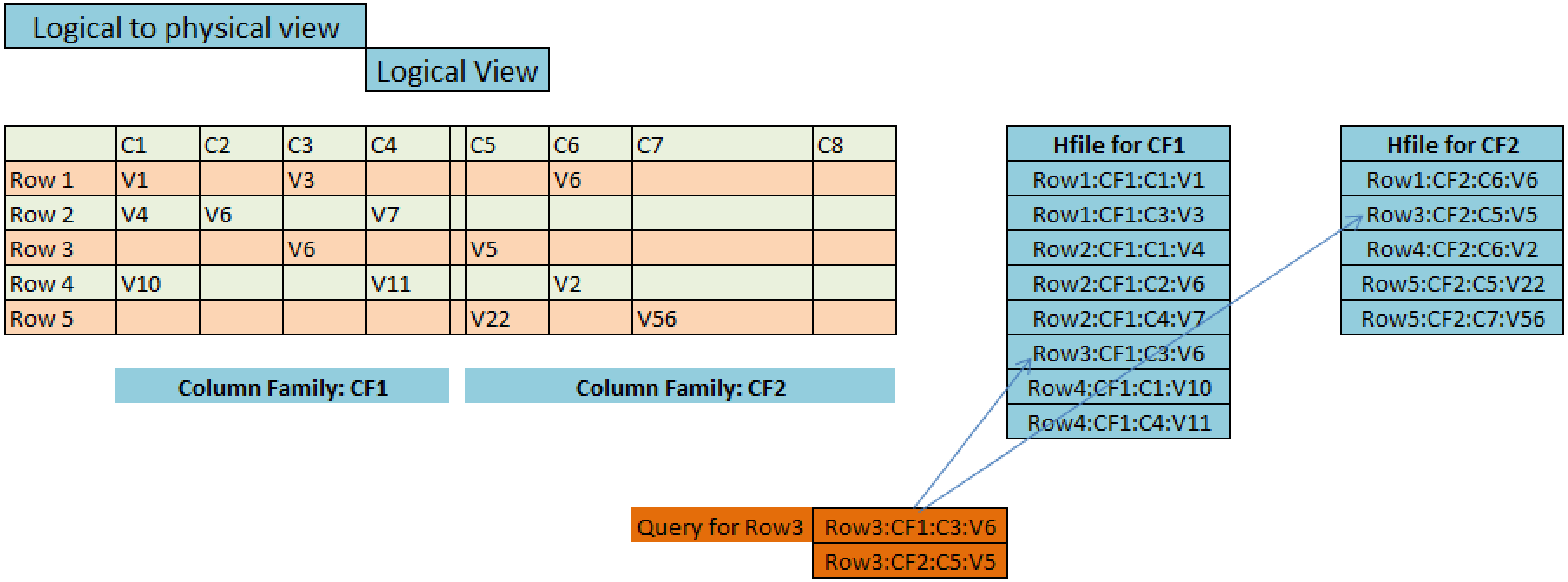
Info column family

Row-Key	column-family:column-key	Timestamp	cell value
1234	info:fn	123456789	John
1234	info:ln	123456789	Smith
5678	info:fn	123456789	Joyce
5678	info:ln	123456789	cooper

Pwd column family

Row-Key	column-family:column-key	Timestamp	cell value
1234	pwd:p	123456789	Hello, world!
5678	pwd:p	123456789	dsajk
5678	pwd:p	123456789	naslna

Logical to Physical view



Logical to physical view

Logical View

	C1	C2	C3	C4	C5	C6	C7	C8
Row 1	V1		V3			V6		
Row 2	V4	V6		V7				
Row 3			V6		V5			
Row 4	V10			V11		V2		
Row 5					V22		V56	
	Column Family: CF1				Column Family: CF2			

Hfile for CF1
Row1:CF1:C1:V1
Row1:CF1:C3:V3
Row2:CF1:C1:V4
Row2:CF1:C2:V6
Row2:CF1:C4:V7
Row3:CF1:C3:V6
Row4:CF1:C1:V10
Row4:CF1:C4:V11

Hfile for CF2
Row1:CF2:C6:V6
Row3:CF2:C5:V5
Row4:CF2:C6:V2
Row5:CF2:C5:V22
Row5:CF2:C7:V56

Query for Row3	Row3:CF1:C3:V6
	Row3:CF2:C5:V5

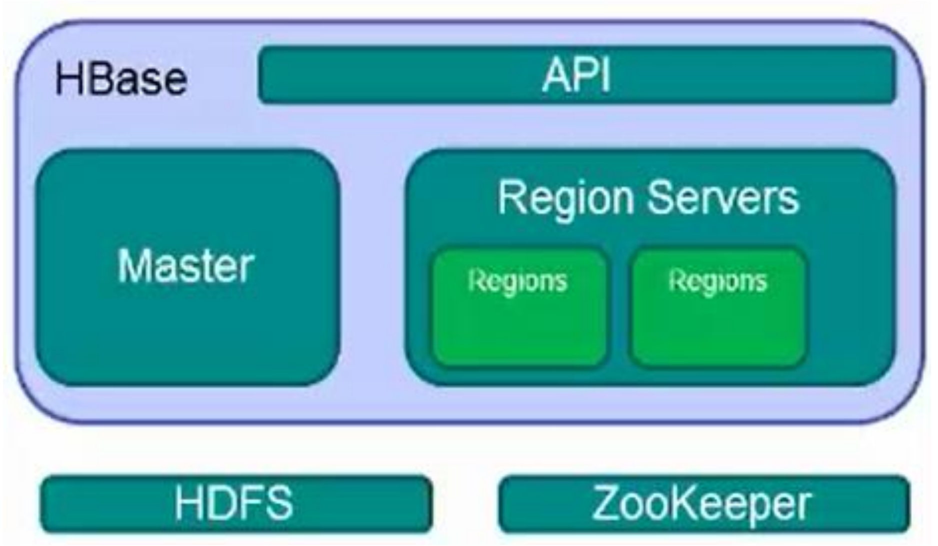
HBase Architecture

Region

- The rows of a table are stored within Region
- A table's data is automatically sharded across multiple regions when the data gets too large
- Each region stores a single Column Family

Region Server

- Contains one or more Regions
- Hosts the tables, perform read/write, buffers
- Client talks directly to the Region Server for their data.



Master

- Coordinating the Region Servers
- Detects status and load rebalancing of the Region Servers
- Assigns Regions to Region Servers
- Multiple Masters are allowed, but only serves as backups
- Not part of the read/write path
- Highly availiable with Zookeeper

HBase Architecture

Zookeeper

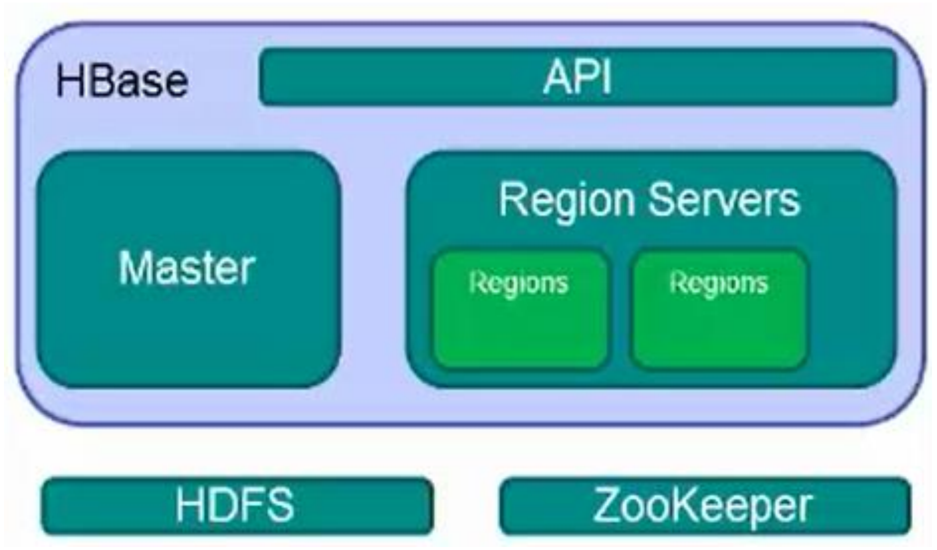
- Critical component for HBase
- Ensures one Master is running
- Registers Region and Region server
- Integral part of the fault tolerance on HBase

HDFS

- The Hadoop filesystem

API

- The JAVA client API



HBase table characteristics

- HBase tables consists of rows.
  - Each row must have a unique row key
- HBase rows are formed by one or more columns
- Columns can have multiple timestamps to indicate the version
  - The most recent version of the data is shown first
- Columns are grouped into Column Families which must be defined up front
- Column-Families contains columns
  - From our example earlier: info: *l*
  - info is the column family
  - *l* is the column qualifier or column key of the last name
  - Column --> Column Family + Qualifier
- Columns can be added on the fly
- Columns can be NULL
  - NULL columns are not stored and free of costs
  - Making HBase sparse.
- Cell --> (Row Key, CF, CQ, Version)

Row key design considerations

- Having an efficient HBase system is highly dependent on how you choose your row key, or primary key(for those still in the RDBMS mindset)

- Depending on how your data is accessed, you will need to design your row key accordingly

- Some examples:
  - Lots of writes to your table: random row keys--> good for performance
  - Lots of reads to your table: sequential row keys--> good for performance
  - Time series data like log files: sequential row keys-->good for scans of specific time

- This is not a trivial task and can be considered and advance topic, but it's good to keep this on the back of your mind as you learn about HBase

- Choose keys that can allow your data to be distributed as evenly as you can for your usage

Lab

HBase shell commands

Data Ingestion in to HBase

- Sqoop <--> HBase
- Java API <--> HBase
- Pig Semi Structured Data <--> HBase

Accessing HBase :

- Pig <--> HBase (Processing)
- Hive <--> HBase (Enable access)

Conclusion

•HBase is a NoSQL Hybrid Key Value, columnar database.

•Should be only used, when you have single key, and any number of values scenario.

•HBase is known for wider formats(any number of columns), RDBMS is known for Long Formats(Less columns, more rows)

•HBase doesn't have any querying interface, using Hive, external tables can be created on HBase to enable access.

•HBase implements master-slave architecture.

•HBase support billions rows, million columns. Columns can be added on the fly.

You know the following:

1. Data Ingestion into HBase from MySQL using Sqoop.
2. Load RAW files to HBase using HBase Java API.
3. Process HBase data using Pig.
4. Enable access to HBase using Hive.