CS 398 ACC
NoSQL and Key/Value Stores

Prof. Robert J. Brunner

Ben Congdon
Tyler Kim

MP6

How’s it going?

- Due March 13th at 11:59 pm.

Submit your results as a PDF report on Moodle

Final Project Reminders

- **Group Selection**: Due March 10th at 11:59pm
  - Group commitment form will be posted on Piazza shortly

- **Project Proposal**: Due March 16th at 11:59pm
  - See requirements on the Course Website

A little bit of history

- Most databases became SQL-like in the 1980s

- In 2006 Google published their BigTable paper
  - It was not SQL
  - It was designed to scale to petabytes of data (1000s of gigabytes) on thousands of nodes
  - Solved scaling by relaxing availability

- In 2007 Amazon published their Dynamo paper
  - Again not SQL; Similarly solves the problem of scaling
  - Solved scaling through relaxing consistency

- By 2009 there were tons of systems like these
- Now when you have hundreds of nodes, NoSQL is the normal solution

NoSQL

- Databases which may not be relational and can scale to tons and tons of servers
- Sacrifice SQL compatibility to get higher read/write/storage rates
- Needed when data cannot be managed a few servers

SQL vs NoSQL

- SQL systems are typically good at consistency
  - If data is written to a row, all reads will get that write
  - This can slow down transactions
- The vast majority of databases (not only SQL) are ACID:
  - Atomic
  - Consistent
  - Isolated
  - Durable

- ACID is analogous to the properties of a global variable in a single threaded program

NoSQL Database Types

- Key Value Store
- Document Oriented
- Columnar Storage

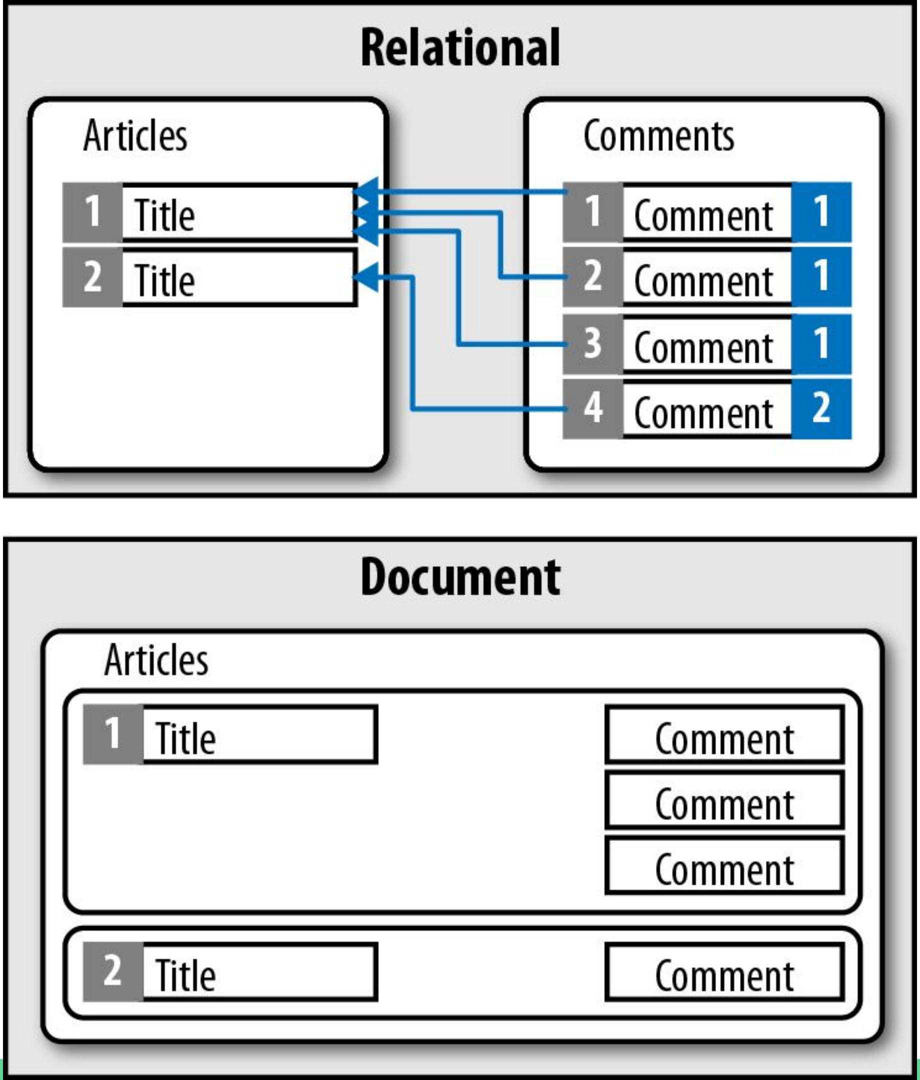
Key Value Store

- These store key value pairs really really well
- Can be used as a distributed cache
- Some document stores are key value stores under the hood

```
{
        Key1: val1,
        Key2: val2
}
```

Document Oriented Databases

- They store complex structures like:
  - JSON
  - XML
  - YAML
- These work really well when most queries are for one item instead of aggregations
- Typically provide their own unique query languages
- These extend the idea of key value stores to more complex types


Relational
Articles
1 Title
2 Title
Comments
1 Comment 1
2 Comment 1
3 Comment 1
4 Comment 2
Document
Articles
1 Title
Comment
Comment
Comment
2 Title
Comment

Document Vs Key Value Databases

Both address objects with a key:

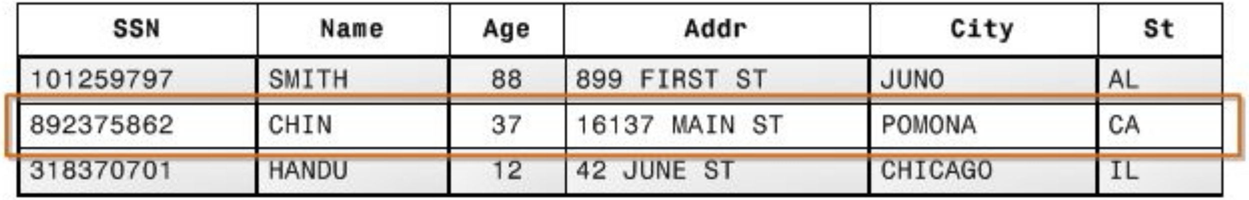
- Document DBs cluster documents within collections
- Key value stores mainly have only one collection
- Key value stores are faster (Smaller values and less structure)
- Document DBs support more extensive query languages in general

- If you do not need complex objects, use a key value store

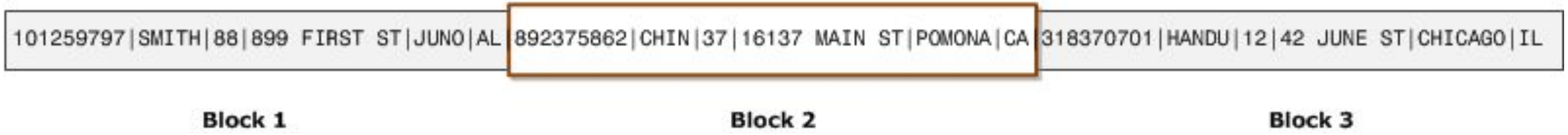
Columnar/Column based Databases

- Columns are stored together instead of rows
- A row is can be split amongst many machines
- Makes aggregations really fast since a single column normally resides on one machine
- Usually does not support joins (or joins are very slow)

Row based databases

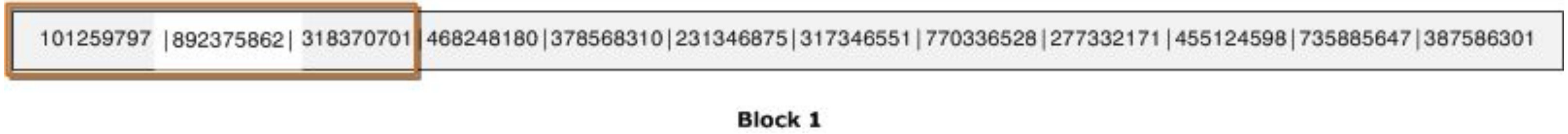


SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



101259797|SMITH|88|899 FIRST ST|JUNO|AL|892375862|CHIN|37|16137 MAIN ST|POMONA|CA|318370701|HANDU|12|42 JUNE ST|CHICAGO|IL
Block 1
Block 2
Block 3

Columnar based databases



Source: AWS

Relaxing Constraints

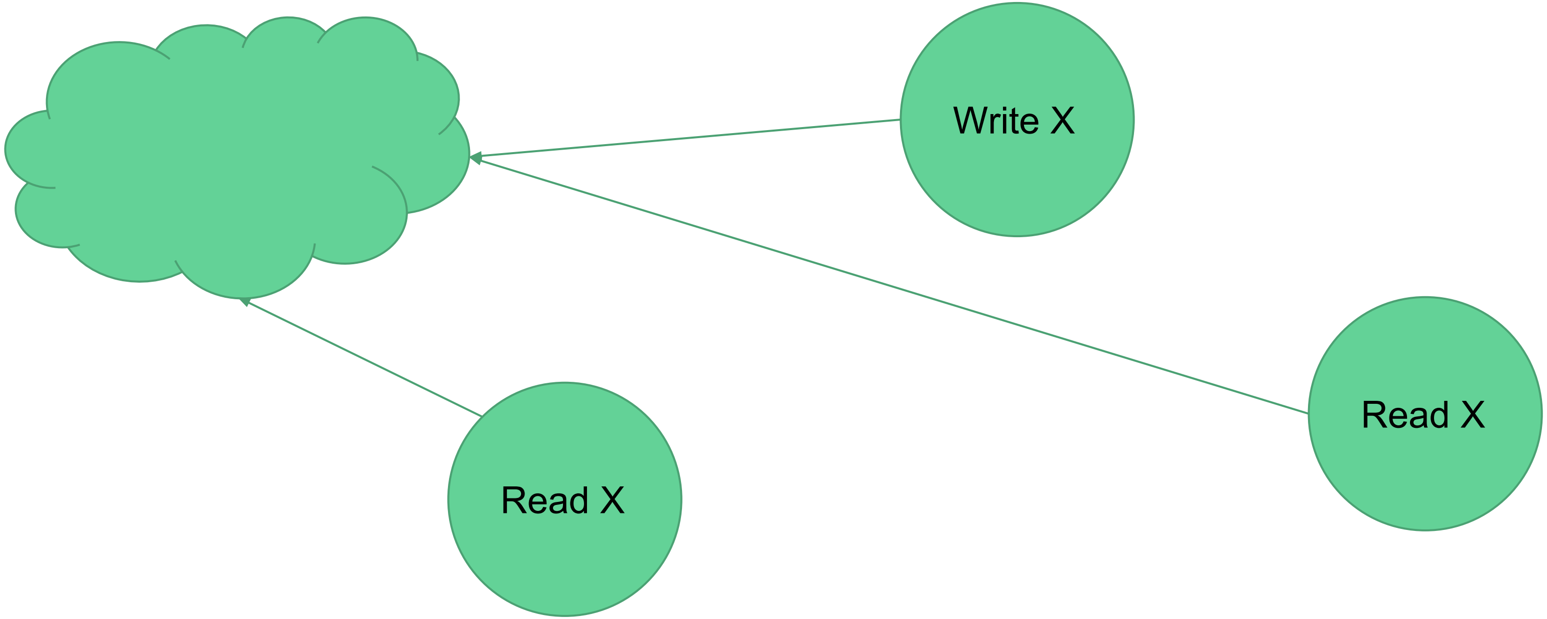
- All of the above types can be implemented using a normal SQL database a backend

- They can also be implemented as ACID databases

- What if we specified you did not need strong consistency?

Eventual Consistency

- Making everything consistent immediately means clients need to queue
- Say you have 3 clients, one writing and the other two reading



Eventual Consistency

- Making everything consistent immediately means clients need to queue
- Say you have 3 clients, one writing and the other two reading

The true ordering is

T=0	T=1	T=2
Read X	Write X	Read X

Eventual Consistency

- But you could receive this order because of network delays

T=0	T=1	T=2
Read X	Read X	Write X

Eventual Consistency

- Or if you have two servers, one could receive the true ordering and one the out of order ordering

Server 1 sees

T=0	T=1	T=2
Read X	Write X	Read X

Server 2 Sees

T=0	T=1	T=2
Read X	Read X	Write X

Eventual Consistency

- But sometimes we can afford old values being read for a little while. This means we can read and write at the same time.

Server 1 sees

T=0	T=1	T=2
Read X	Write X	Read X

Server 2 Sees

T=0	T=1	T=2
Read X	Read X	Write X

CAP Theorem

- **Consistency**
    - All reads receive the most recent write or error
- **Availability**
    - Every read/write receives a non error
- **Partition Tolerance**
    - Everything keeps working if the network starts dropping messages

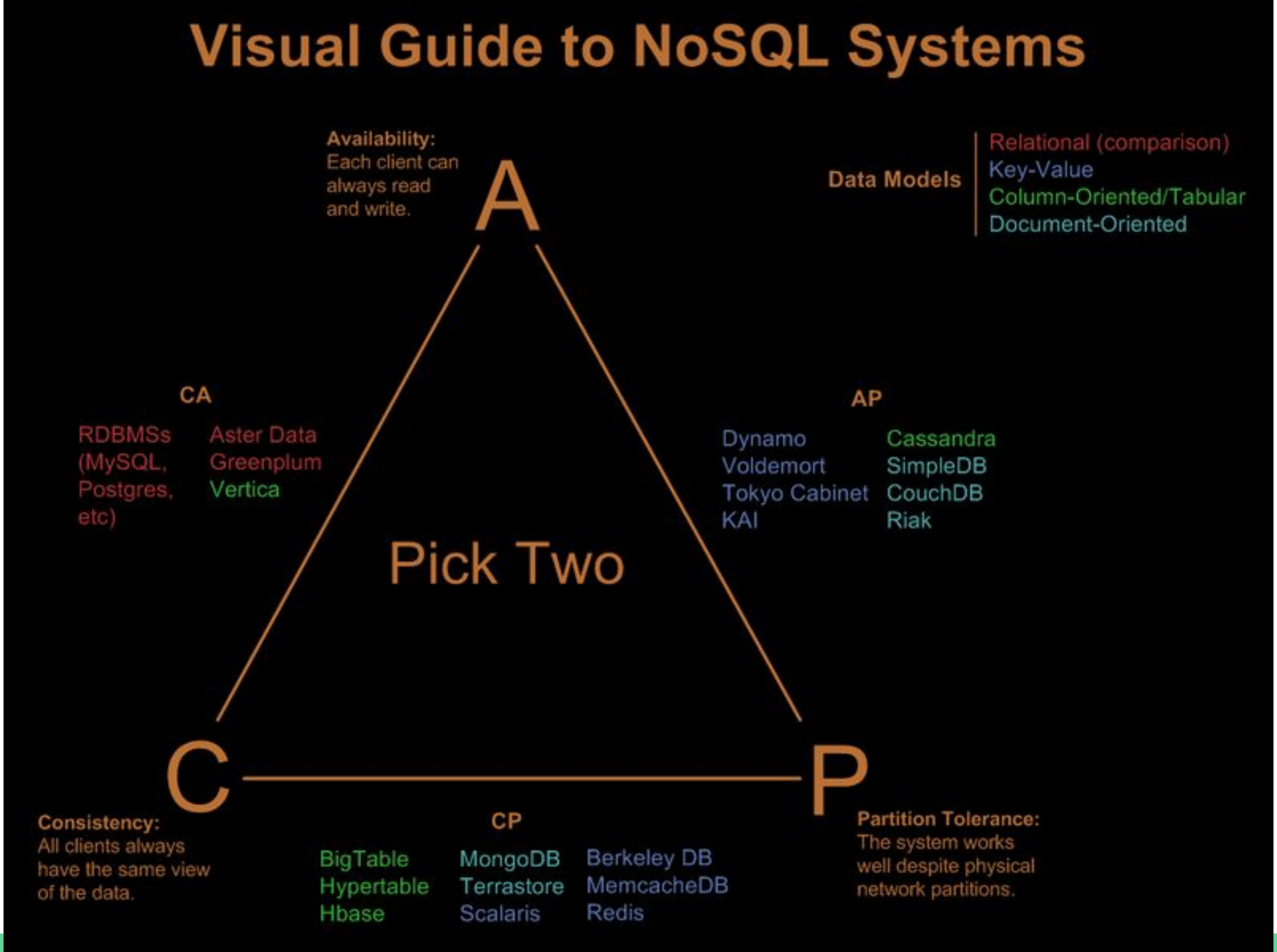
Pick 2

CAP Theorem

- **Consistency**
  - All reads receive the most recent write or error
- **Availability**
  - Every read/write recieves a non error
- **Partition Tolerance**
  - Everything keeps working if the network starts dropping messages

Pick 2

- Each of these have a non strict version
- But you cannot guarantee all 3 in all scenarios



Visual Guide to NoSQL Systems

Availability: Each client can always read and write.

Data Models
- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak

CP

BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis

Consistency: All clients always have the same view of the data.

Partition Tolerance: The system works well despite physical network partitions.

From Ofirm

CA Systems

- Consistency and availability
  - They will always respond with the latest write

- Most SQL databases are CA systems.

- SQL Systems
  - MySQL
  - MSSQL
  - SQLite
  - PostgreSQL

CP Systems

Consistency and Partition Tolerance

- Will give you the latest write or give you an error if not possible
- Can survive half the network going down
- HBase, BigTable, MongoDB

APACHE HBASE

- Is a CP system
- Used in HDFS
- Linear and modular scalability.
- Strictly consistent reads and writes.
- Automatic and configurable sharding of table

- Everything is still a table
- Can return an error since it is CP

mongoDB®

- Is a CP system
- Extremely easy to set up
  - The defaults are insecure
- Document Oriented DB
  - Only stores JSON objects
  - No longer a simple table
- Is a key value store

redis

- CP System
- Is a key value store
- Lets you write data structures into memory and share them
- Very fast: Often used as a caching layer

DynamoDB

amazon web services™

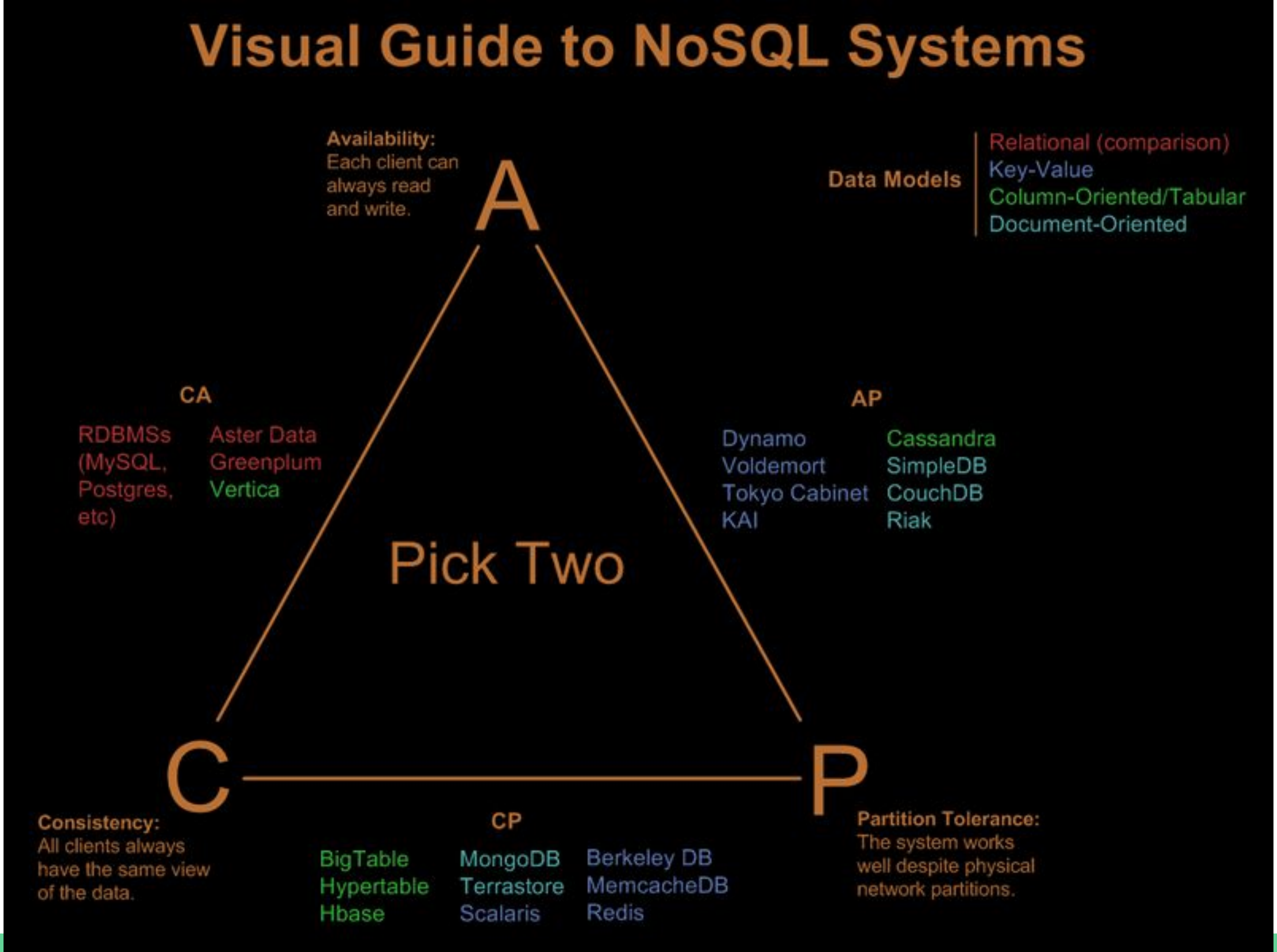
- CP System
- Resembles a key-value store
    - Allows indexing, but this involves data-replication
- Very fast reads
- Scales well

Google Spanner

- CP System
- Global SQL Database

AP Systems

- Available and partition tolerant
- Never returns an error even when half the network is dead



Visual Guide to NoSQL Systems

Data Models

- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

Availability:
Each client can always read and write.

CA

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Consistency:
All clients always have the same view of the data.

CP

BigTable
Hypertable
Hbase

MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

Partition Tolerance:
The system works well despite physical network partitions.

From Ofirm

cassandra

- AP system
- Column-Oriented
- Super high availability and super high throughput
  - Used by Reddit, Facebook and others

riak

- AP system
- Key value store
- Can be faster than MongoDB but sacrifices consistency

When to use SQL vs NoSQL

- By default use an SQL database

- Use NoSQL when you need **more than one serve**r AND you have a **super high write rate**
    - NoSQL happens when you can sacrifice CA and need some other pair from CAP

Wednesday

- Project Proposal Help
- MLlib Lab Office Hours

MP 6

Due next **Tuesday, March 13th** at 11:59pm

Topic: “Spark MLlib”

> Check Piazza for Q&A and Announcements