Data-intensive computing systems

NoSQL systems

University of Verona
Computer Science Department

Damiano Carra

Acknowledgements

- *Credits*
  - *Part of the course material is based on slides provided by the following authors*
    - *Willem Visser, Firat Atagun*

- For a fairly complete overview of the topic, see
  - Christof Strauch, “NoSQL Databases”
    - www.christof-strauch.de/nosqldbs.pdf

The NoSQL Movement

- ❑ Not Only SQL
  - It is not No SQL
  - Not only relational would have been better
- ❑ Use the right tools (DBs) for the job
- ❑ It is more like a feature set, or even the not of a feature set

What is Wrong With RDBMS?

- ❑ One size fits all?
- ❑ Rigid schema design
- ❑ Harder to scale
  - How does RDMS handle data growth?
  - Joins across multiple nodes?
- ❑ Replication
  - Difficult to manage
- ❑ But..
  - Many programmers are already familiar with it…
  - Transactions and ACID make development easy
  - Lots of tools to use

Definition from nosql-databases.org

- ❑ Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: schema-free, easy replication support, simple API, eventually consistent /BASE (not ACID), a huge data amount, and more. So the misleading term "nosql" (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above.

Use Cases

- ❑ Massive write performance
- ❑ Fast key value lookups
- ❑ Flexible schema and data types
- ❑ No single point of failure
- ❑ Fast prototyping and development
- ❑ Out of the box scalability
- ❑ Easy maintenance

Advantages of NoSQL

- ❑ Cheap, easy to implement
- ❑ Data are replicated and can be partitioned
  - Easy to distribute
- ❑ Don't require a schema
- ❑ Can scale up and down
  - Quickly process large amounts of data
  - Can handle web-scale data
- ❑ Relax the data consistency requirement (CAP)

Disadvantages of NoSQL

- ❑ New and sometimes buggy
- ❑ Data is generally duplicated, potential for inconsistency
- ❑ No standardized schema
- ❑ No standard format for queries
- ❑ No standard language
- ❑ Difficult to impose complicated structures
- ❑ Depend on the application layer to enforce data integrity
- ❑ No guarantee of support
- ❑ Too many options
  - Which one, or ones to pick?

CAP Theorem

- ❑ Also known as Brewer’s Theorem by Prof. Eric Brewer
  - Published in 2000 at University of Berkeley

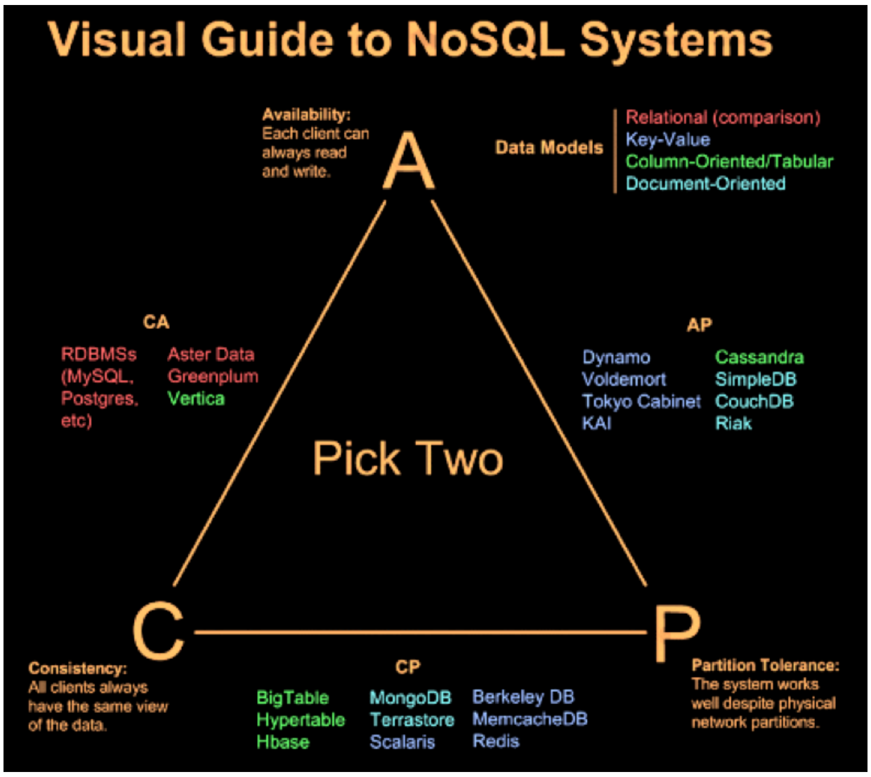
“Of three properties of a shared data system: data ***consistency***, system ***availability*** and tolerance to network ***partitions***, only two can be achieved at any given moment.”

- ❑ Proven by Nancy Lynch et al. MIT labs

CAP Semantics

- ❑ Consistency:
  - Clients should read the same data
- ❑ Availability:
  - Each client can always read / write
- ❑ Partial Tolerance:
  - The system works well despite network failures (partitions)

CAP Theorem



ACID Semantics

- ❑ Atomicity → All or nothing
- ❑ Consistency → Consistent state of data and transactions
- ❑ Isolation → Transactions are isolated from each other
- ❑ Durability → When the transaction is committed, state will be durable

- ❑ Any data store can achieve Atomicity, Isolation and Durability but do you always need consistency?
  - Not always

- ❑ By giving up ACID properties, one can achieve higher performance and scalability

BASE, an ACID Alternative

- Almost the opposite of ACID

- Basically Available
  - Nodes in the a distributed environment can go down, but the whole system shouldn't be affected

- Soft State (scalable)
  - The state of the system and data changes over time

- Eventual Consistency
  - Given enough time, data will be consistent across the distributed system

Issues: managing distributed systems

- How to maintain scalability and performance?
  - The system should be simple and flexible
  - Some properties (e.g., consistency) can be relaxed

- What are the tools that can be used?
  - How to assign data to nodes?
    - Partitioning
  - How to manage the consistency?
    - Versioning
  - How to store data on each node?
    - Row-based layout, Column-based, …

Partitioning: Consistent Hashing

- ❑ Problem: Assign portions of data to different nodes
- ❑ Requirements:
  - Support load balancing
  - Allow for dynamic nodes
- ❑ A possible solution: Hashing

  destination = hash(data) mod N

  - Intuition: Assign data to nodes uniformly at random
- ❑ Problem: What if a node fails? Or a new node is added?

  destination = hash(data) mod (N ± 1)

  - inconsistency: data is stored in different nodes…
  - all data should be redistributed

Partitioning: Consistent Hashing

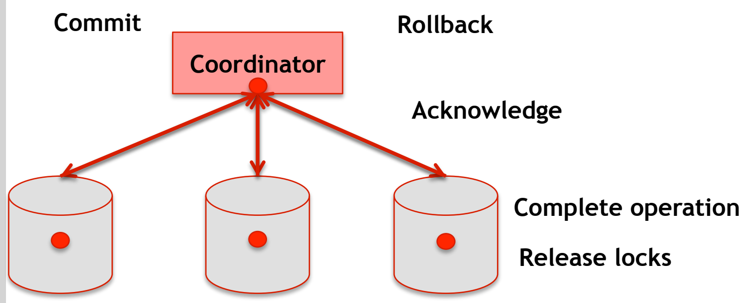
- ❑ Managing dynamic nodes
  - Hash node IDs → H_i = hash(N_i)
  - Hash data → D = hash(data)
  - Send data to the closest node in the hash space
    - select i such that the distance between H_i and D is the minimum
- ❑ Properties
  - All buckets get roughly same number of items (like standard hashing)
  - When kth node is added only a 1/k fraction of items move, and only from a few nodes
  - To handle node failures, the data is ***replicated*** into k nearest nodes

Data Consistency

- ❑ When data is replicated (for reliability reasons), then we need to manage the consistency
  - There are many levels of consistency.
    - Strict Consistency – RDBMS
    - Tunable Consistency – Cassandra
    - Eventual Consistency – Amazon Dynamo
- ❑ There are many solutions to manage consistency
  - The complexity and the performance of the solutions depend on the level of required consistency

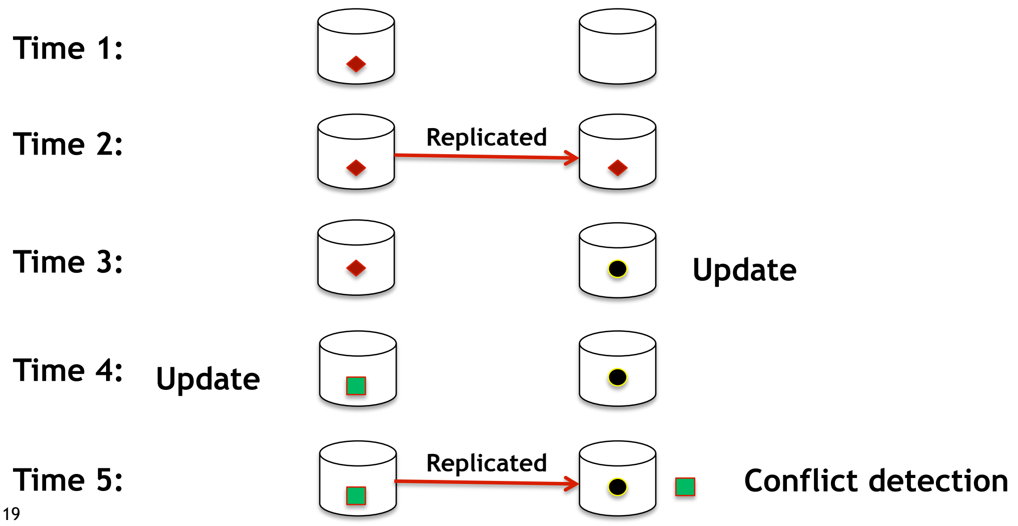
Distributed Transactions

- ❑ Two phase commit.
- ❑ Possible failures
  - Network errors.
  - Node errors.
  - Database errors.
- ❑ Problems
  - Locking the entire cluster if one node is down
  - Possible to implement timeouts
  - Possible to use Quorum
  - Quorum: in a distributed environment, if there is partition, then the nodes vote to commit or rollback

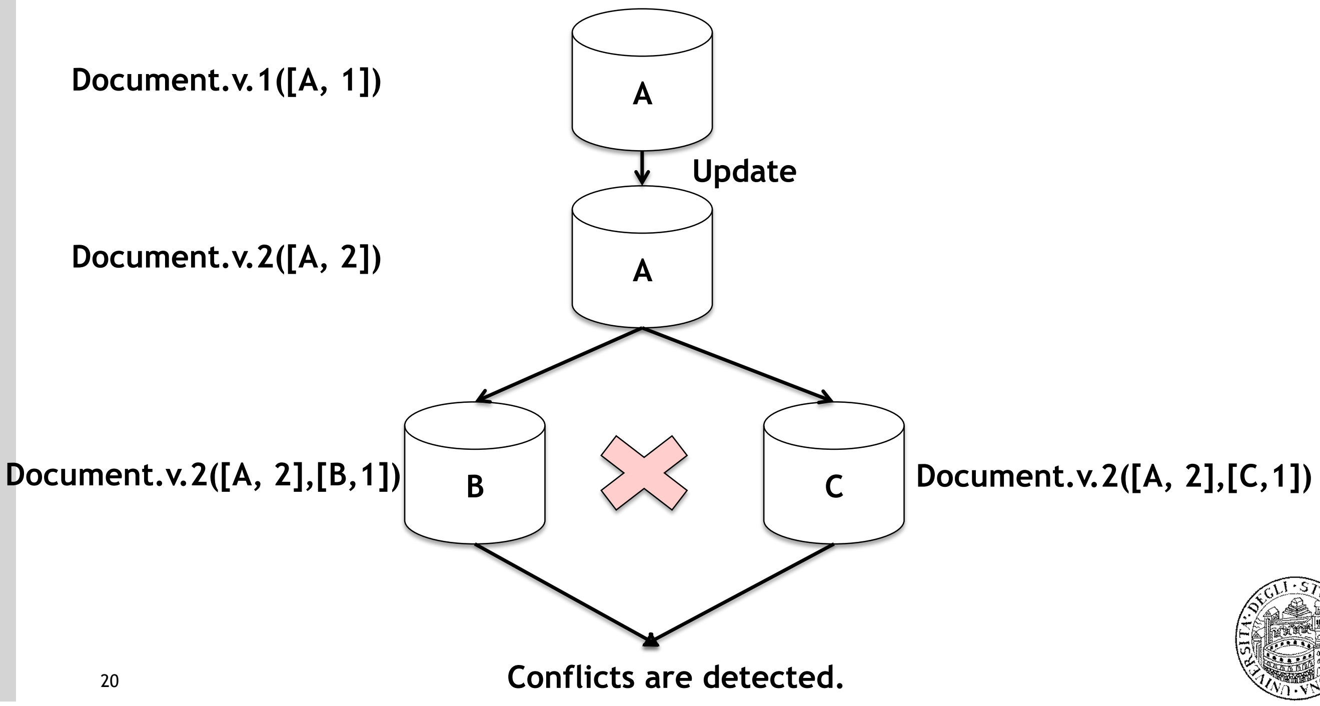


Vector Clocks

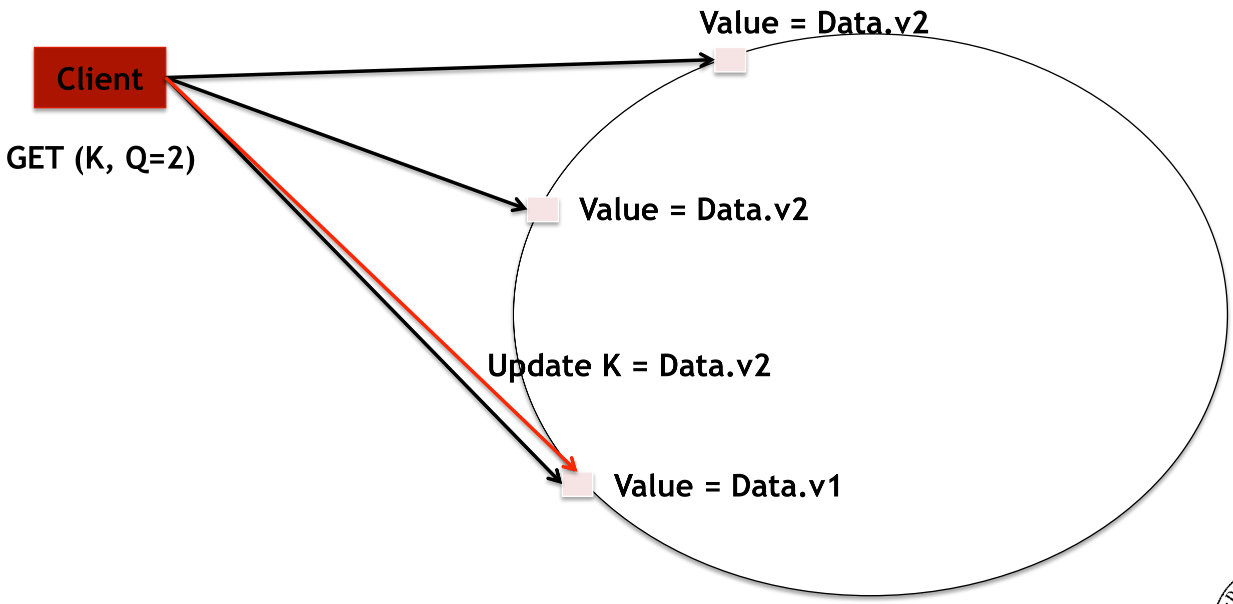
- ❑ Used for *conflict detection* of data.
- ❑ Timestamp based resolution of conflicts is not enough.



Vector Clocks

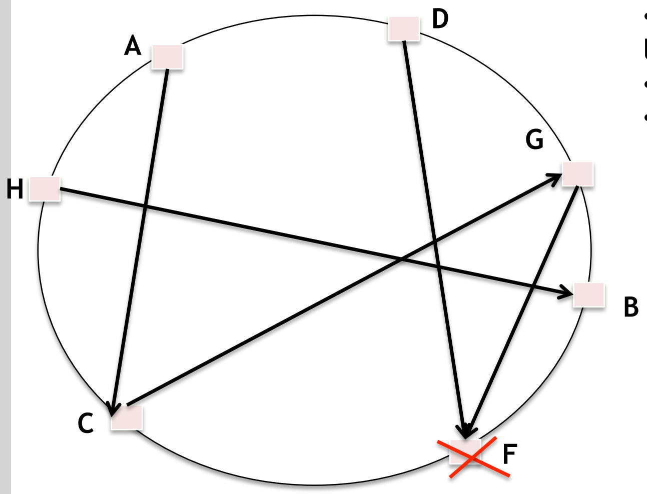


Read Repair



Gossip Protocol & Hinted Handoffs

- ❑ Most preferred communication protocol in a distributed environment is Gossip Protocol.



- All the nodes talk to each other peer wise.
- There is no global state.
- No single point of coordinator.
- If one node goes down and there is a Quorum load for that node is shared among others.
- Self managing system.
- If a new node joins, load is also distributed.

Requests coming to F will be handled by the nodes who takes the load of F, lets say C with the hint that it took the requests which was for F, when F becomes available, F will get this Information from C. Self healing property.

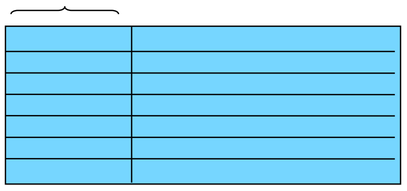
NoSQL system types

NOSQL Data Store Types

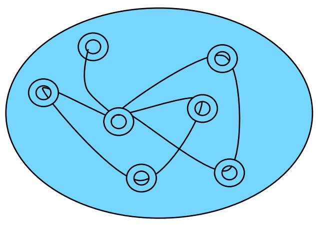
Key Value
•Distributed Hash Table

Key	Value
1	Bob
2	Sue
3	Joe
4	Jo

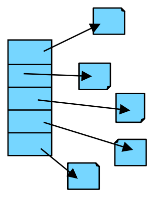
Column Based
•Semi-structured



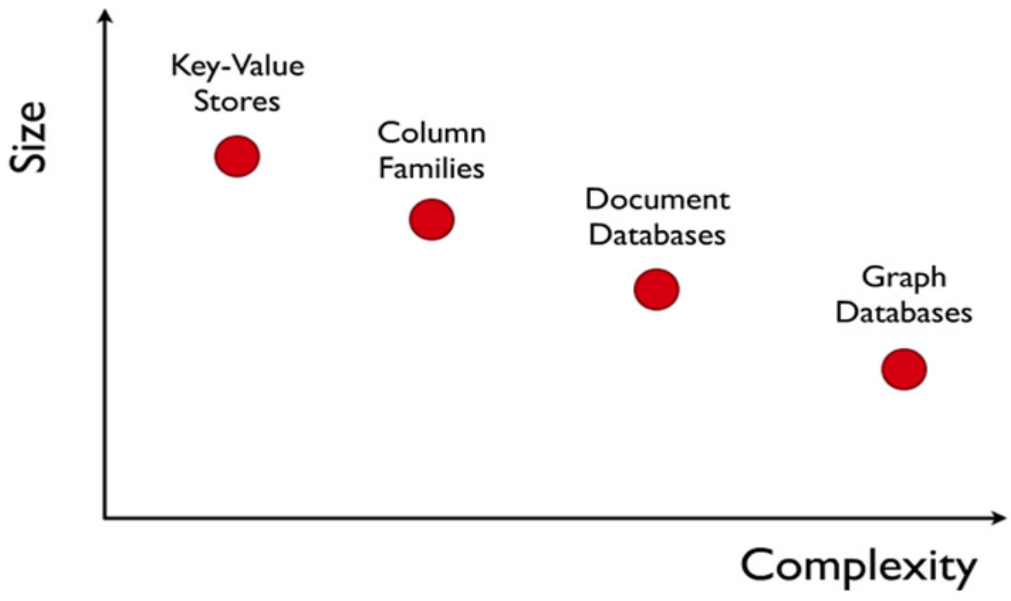
Graph
•Graph Theory



Document
•Semi-structured



Complexity



Key-Value Store

“key”

morpheus

“value”

1011101001101010011001101001001000101010111010101010110000101000110011111010110000101000111110001100000

Key-Value Stores

- ❑ It’s a Hash
- ❑ Basic get/put/delete ops
- ❑ Very fast
- ❑ Easy to scale horizontally
- ❑ Examples:
  - Memcached
    - Key value stores
  - Membase
    - Memcached with persistence and improved consistent hashing
  - Redis
    - Data structure server
  - Project Voldemort
    - Eventual consistent key value stores, auto scaling

Memcached

- ❑ Very easy to setup and use
- ❑ Consistent hashing
- ❑ Scales very well
- ❑ In memory caching, no persistence
- ❑ LRU eviction policy
- ❑ O(1) to set/get/delete
- ❑ Atomic operations set/get/delete
- ❑ No iterators, or very difficult

Redis

- ❑ Distributed Data structure server
- ❑ Consistent hashing at client
- ❑ Non-blocking I/O, single threaded
- ❑ Values are binary safe strings: byte strings
- ❑ String : Key/Value Pair, set/get. O(1) many string operations
- ❑ Lists: lpush, lpop, rpush, rpop. You can use it as stack or queue. O(1). Publisher/ Subscriber is available
- ❑ Set: Collection of Unique elements, add, pop, union, intersection etc. set operations
- ❑ Sorted Set: Unique elements sorted by scores. O(logn). Supports range operations
- ❑ Hashes: Multiple Key/Value pairs
  - HMSET user 1 username foo password bar age 30
  - HGET user 1 age

Column Database

Name	Last Name	Age	Rank	Occupation	Version	Language
Thomas Anderson		29				
Morpheus			Captain	Total badass		
Cypher	Reagan					
Agent Smith					1.0b	C++
The Architect						

Column-oriented

- ❑ Store data in column order
- ❑ Allow key-value pairs to be stored (and retrieved on key) in a massively parallel system
  - Data model: families of attributes defined in a schema, new attributes can be added
  - Storing principle: big hashed distributed tables
  - Properties: partitioning (horizontally and/or vertically), high availability etc. completely transparent to application
- ❑ Examples:
  - Cassandra, Hbase, Bigtable

Document Store

“key”

morpheus

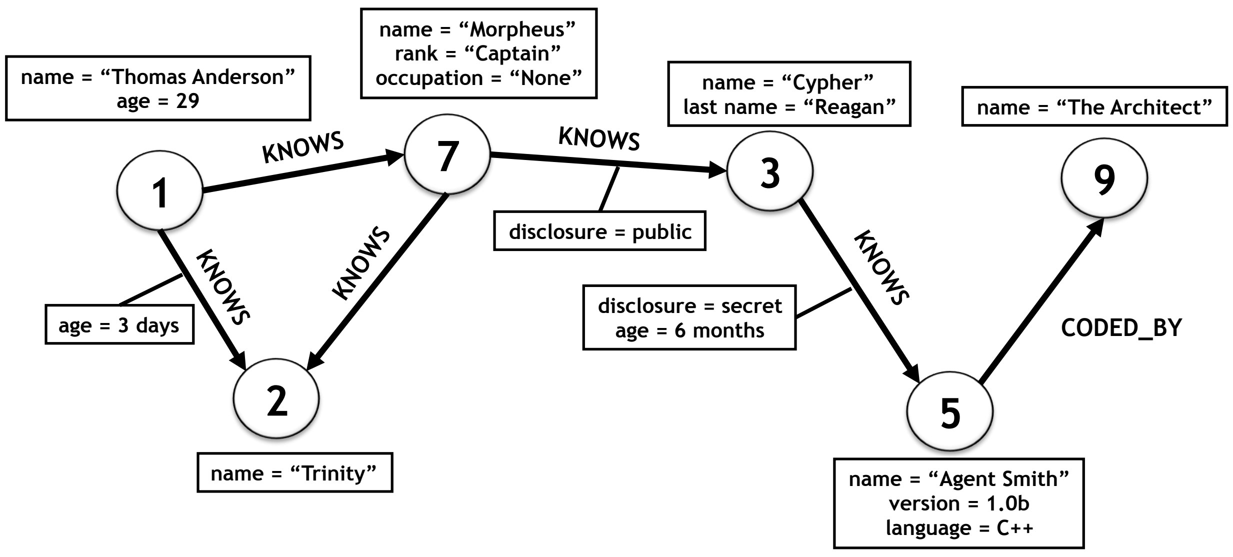
“document”

```
{
   name : “Morpheus”,
   rank : “Captain”,
   occupation: “None”
}
```

Document Store

- ❑ Document = self-contained piece of data
- ❑ Semi-structured data
- ❑ Usually JSON like interchange model
  - The data model supports lists, maps, dates, Boolean with nesting
- ❑ Really: indexed semi-structured documents
- ❑ Query Model: JavaScript or custom
- ❑ Examples:
  - MongoDB, CouchDB, …

Graph Stores



Graph Stores

- ❑ Use a graph structure
  - Labeled, directed, attributed multi-graph
  - Label for each edge
  - Directed edges
  - Multiple attributes per node
  - Multiple edges between nodes
- ❑ Node adjacency instead of indices
- ❑ Relational DBs can model graphs, but an edge requires a join which is expensive
- ❑ Example
  - Neo4j, VertexDB, …

Which one to use?

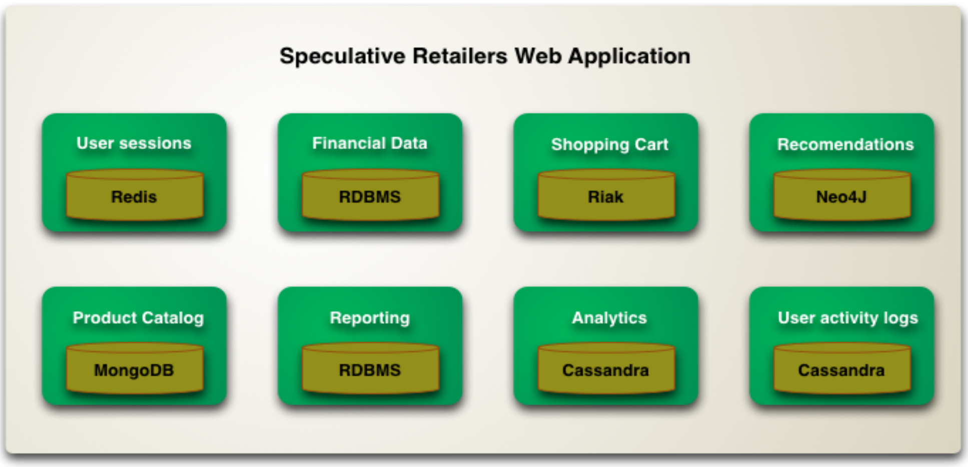
- ❑ Key-value stores:
  - Processing a constant stream of small reads and writes.
- ❑ Document databases:
  - Natural data modeling. Programmer friendly. Rapid development. Web friendly, CRUD.
- ❑ RDMBS:
  - OLTP. SQL. Transactions. Relations.
- ❑ Columnar:
  - Handles size well. Massive write loads. High availability. Multiple-data centers. MapReduce

- ❑ Want more ideas ?

http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html

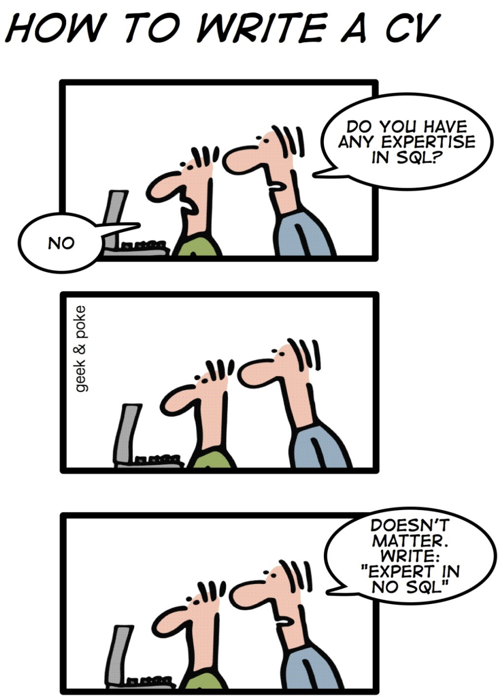
Polyglot Persistence

❑ Using different DB technologies for different storage requirements



http://martinfowler.com/bliki/PolyglotPersistence.html

Conclusion: Leverage the NoSQL boom


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"