NoSQL

CPSC 4670/5670

NoSQL

- What does it mean?
  - Not Only SQL.

Use Cases

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

Motives Behind NoSQL

- Big data.
  - Collect.
  - Store.
  - Organize.
  - Analyze.
  - Share
- Scalability.
  - Data growth outruns the ability to manage it so we need **scalable** solutions.
- Data format.
- Manageability.

Scalability

- Scale up, Vertical scalability.
  - Increasing server capacity.
  - Adding more CPU, RAM.
  - Managing is hard.
  - Possible down times

Scalability

- Scale out, Horizontal scalability.
  - Adding servers to existing system with little effort, aka Elastically scalable.
    - Bugs, hardware errors, things fail all the time.
    - It should become cheaper. Cost efficiency.
  - Shared nothing.
  - Use of commodity/cheap hardware.
  - Heterogeneous systems.
  - Controlled Concurrency (avoid locks).
  - Service Oriented Architecture. Local states.
    - Decentralized to reduce bottlenecks.
    - Avoid Single point of failures.
  - Asynchrony.
  - Symmetry, you don’t have to know what is happening. All nodes should be symmetric.

What is Wrong With RDBMS?

- Nothing. One size fits all? Not really.
- Impedance mismatch.
  - Object Relational Mapping doesn't work quite well.
- Rigid schema design.
- Harder to scale.
- Replication.
- Joins across multiple nodes? Hard.
- How does RDMS handle data growth? Hard.
- Need for a DBA.
- Many programmers are already familiar with it.
- Transactions and ACID make development easy.
- Lots of tools to use.

ACID Semantics

- **A**tomicity: All or nothing.
- **C**onsistency: Consistent state of data and transactions.
- **I**solation: Transactions are isolated from each other.
- **D**urability: When the transaction is committed, state will be durable.

Any data store can achieve Atomicity, Isolation and Durability but do you always need consistency? No.

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

Brewer’s CAP Theorem

A distributed system can support **only two** of the following characteristics:

- Consistency
- Availability
- Partition tolerance
- Proven by Nancy Lynch et al. MIT labs.

- http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

Consistency

- **C**onsistency: Clients should read the same data. There are many levels of consistency.
  - Strict Consistency – RDBMS.
  - Tunable Consistency – Cassandra.
  - Eventual Consistency – Amazon Dynamo.
- client perceives that a set of operations has occurred all at once – Pritchett
- More like Atomic in ACID transaction properties

Availability

- **A**vailability: Data to be available.
- node failures do not prevent survivors from continuing to operate – Wikipedia
- Every operation must terminate in an intended response – Pritchett

Partition Tolerance

- **P**artial Tolerance: Data to be partitioned across network segments due to network failures.
- the system continues to operate despite arbitrary message loss – Wikipedia
- Operations will complete, even if individual components are unavailable – Pritchett

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.

A Clash of cultures

ACID:

- Strong consistency.
- Less availability.
- Pessimistic concurrency.
- Complex.

BASE:

- Availability is the most important thing. Willing to sacrifice for this (CAP).
- Weaker consistency (Eventual).
- Best effort.
- Simple and fast.
- Optimistic.

NoSQL Database Types

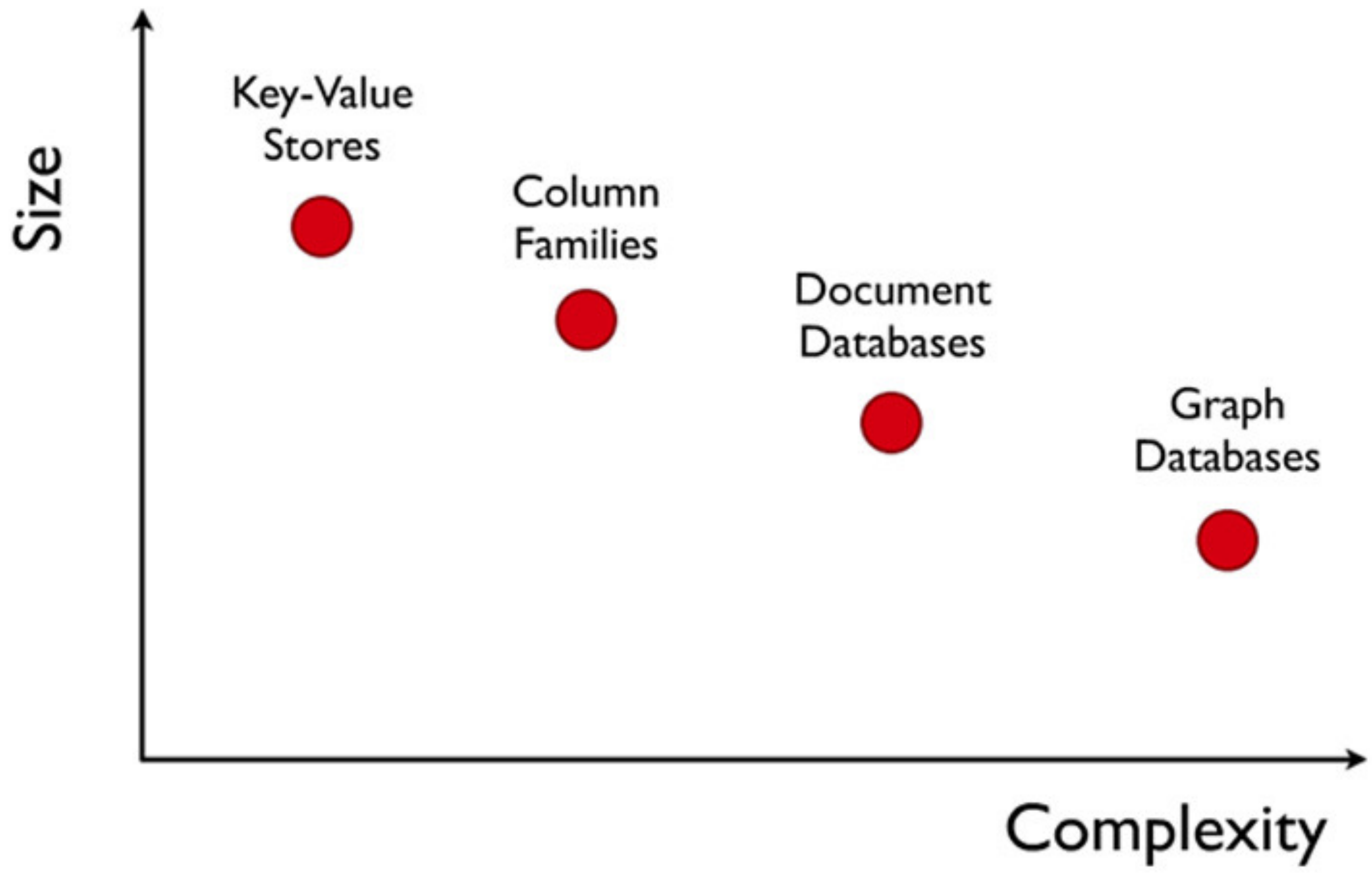
Discussing NoSQL databases is complicated because there are a variety of types:

- Key-Value Store – Hash table of keys
- Column Store – Each storage block contains data from only one column
- Document Store – stores documents made up of tagged elements
- Graph Databases

Other Non-SQL Databases

- XML Databases
- Codasyl Databases
- Object Oriented Databases
- Etc…
- Will not address these today

Complexity


Size
Key-Value Stores
Column Families
Document Databases
Graph Databases
Complexity

NoSQL Examples: Key-Value Store

- Hash tables of Keys
- Values stored with Keys
- Fast access to small data values
- Example – Project-Voldemort
  - http://www.project-voldemort.com/
  - Linkedin
- Example – MemCacheDB
  - http://memcachedb.org/
  - Backend storage is Berkeley-DB

NoSQL Examples: Map Reduce

- Technique for indexing and searching large data volumes
- Two Phases, Map and Reduce
  - Map
    - Extract sets of Key-Value pairs from underlying data
    - Potentially in Parallel on multiple machines
  - Reduce
    - Merge and sort sets of Key-Value pairs
    - Results may be useful for other searches

Map/Reduce

- map(key, val) is run on each item in set
  - emits new-key / new-val pairs

- reduce(key, vals) is run for each unique key emitted by map()
  - emits final output

MapReduce Examples: count words in docs

– Input consists of (url, contents) pairs

– map(key=url, val=contents):
  - For each word *w* in contents, emit (w, “1”)

– reduce(key=word, values=uniq_counts):
  - Sum all “1”s in values list
  - Emit result “(word, sum)”

Count, Illustrated

```
map(key=url, val=contents):
    For each word w in contents, emit (w, "1")
reduce(key=word, values=uniq_counts):
    Sum all "1"s in values list
    Emit result "(word, sum)"
```

see bob throw
see spot run

see	1
bob	1
run	1
see	1
spot	1
throw	1

bob	1
run	1
see	2
spot	1
throw	1

MapReduce Example: Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, “1”)

- reduce(key=line, values=uniq_counts):
  - Don’t do anything; just emit line

Map Reduce

- Map Reduce techniques differ across products
- Implemented by application developers, not by underlying software

NoSQL Example: Document Store

- Schema Free.
- Usually JSON like interchange model.
- Query Model: JavaScript or custom.
- Aggregations: Map/Reduce.
- Indexes are done via B-Trees.

- Example: CouchDB
  - http://couchdb.apache.org/
  - BBC
- Example: MongoDB
  - http://www.mongodb.org/
  - Foursquare, Shutterfly
- JSON – JavaScript Object Notation

CouchDB JSON Example

```
{
  "_id": "guid goes here",
  "_rev": "314159",

  "type": "abstract",

  "author": "Keith W. Hare"

  "title": "SQL Standard and NoSQL Databases",

  "body": "NoSQL databases (either no-SQL or Not Only SQL)
           are currently a hot topic in some parts of
           computing.",
  "creation_timestamp": "2011/05/10 13:30:00 +0004"
}
```

CouchDB JSON Tags

- "_id"
  - GUID – Global Unique Identifier
  - Passed in or generated by CouchDB
- "_rev"
  - Revision number
  - Versioning mechanism
- "type", "author", "title", etc.
  - Arbitrary tags
  - Schema-less
  - Could be validated after the fact by user-written routine

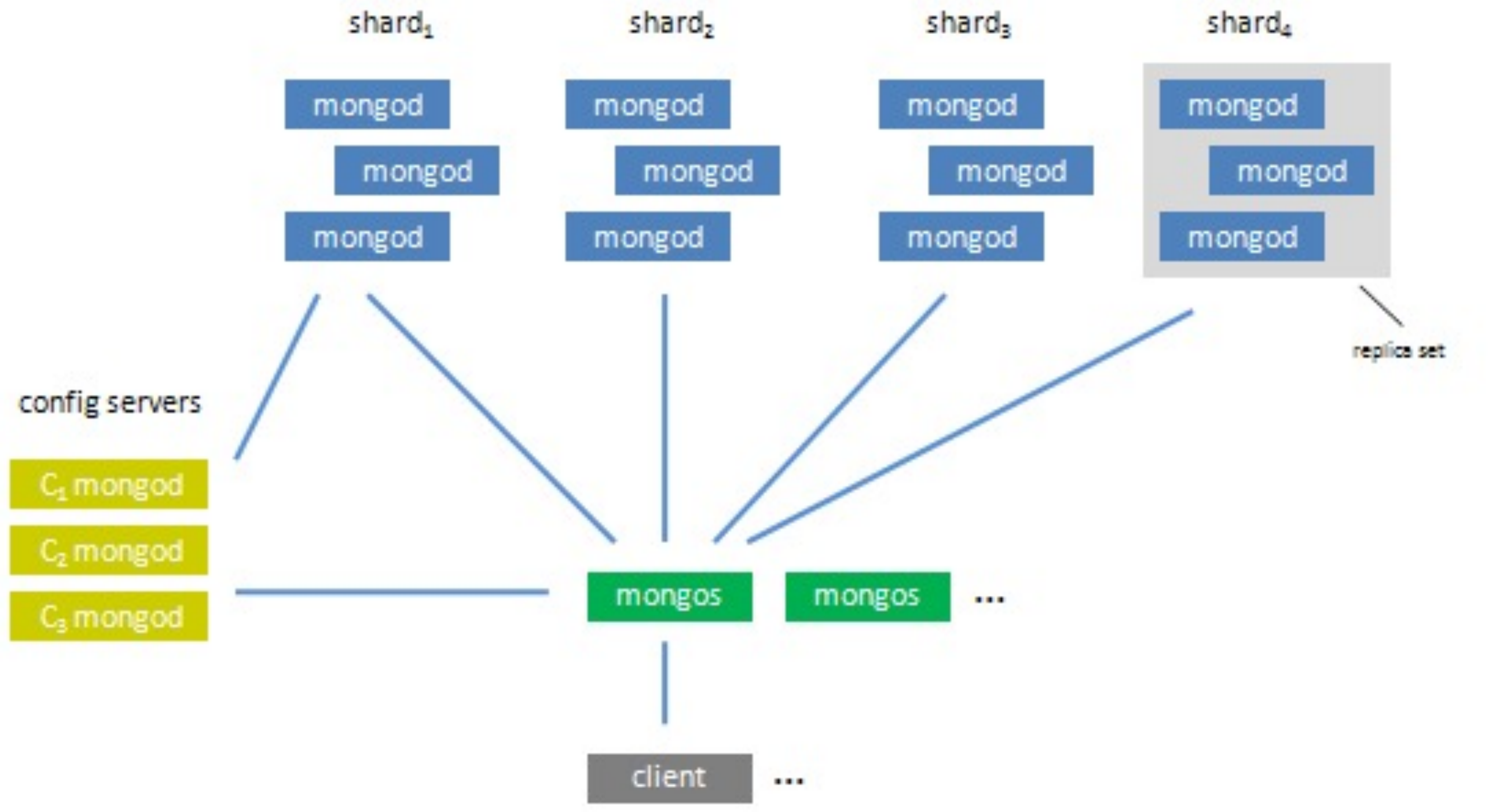
Mongodb

- Data types: bool, int, double, string, object(bson), oid, array, null, date.
- Database and collections are created automatically.
- Lots of Language Drivers.
- Capped collections are fixed size collections, buffers, very fast, FIFO, good for logs. No indexes.
- Object id are generated by client, 12 bytes packed data. 4 byte time, 3 byte machine, 2 byte pid, 3 byte counter.
- Possible to refer other documents in different collections but more efficient to embed documents.
- Replication is very easy to setup. You can read from slaves.

Mongodb

- Supports aggregation.
  - Map Reduce with JavaScript.
- You have indexes, B-Trees. Ids are always indexed.
- Updates are atomic. Low contention locks.
- Querying mongo done with a document:
  - Lazy, returns a cursor.
  - Reduceable to SQL, select, insert, update limit, sort etc.
    - There is more: upsert (either inserts of updates)
  - Several operators:
    - $ne, $and, $or, $lt, $gt, $incr,$decr and so on.
- Repository Pattern makes development very easy.

Mongodb - Sharding



Config servers: Keeps mapping
Mongos: Routing servers
Mongod: master-slave replicas

Graph Stores

- Based on Graph Theory.
- Scale vertically, no clustering.
- You can use graph algorithms easily.

Graph Stores -- Neo4J

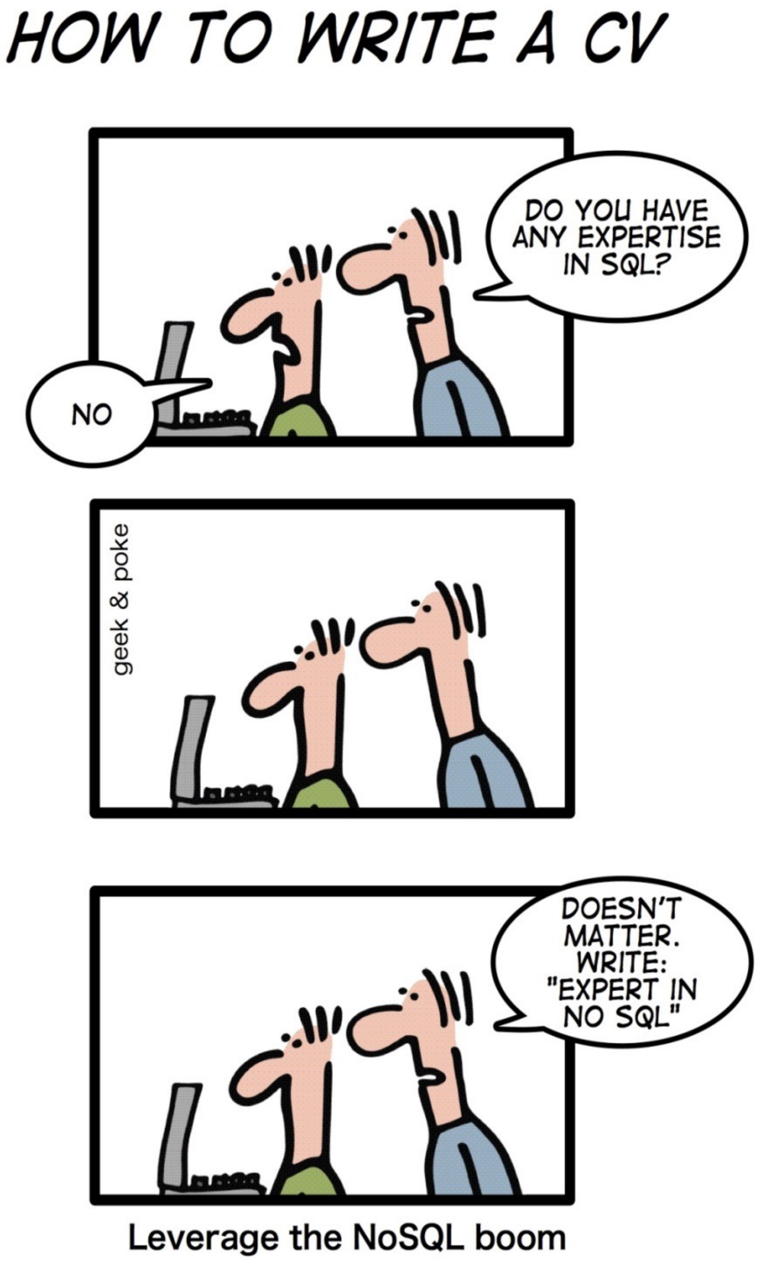
- Nodes, Relationship.
- Traversals.
- HTTP/REST.
- ACID.
- Web Admin.
- Not too much support for languages.
- Has transactions.

NoSQL Summary

- NoSQL databases reject:
  - Overhead of ACID transactions
  - “Complexity” of SQL
  - Burden of up-front schema design
  - Declarative query expression
  - Yesterday’s technology
- Programmer responsible for
  - Step-by-step procedural language
  - Navigating access path

Summary

- SQL Databases
  – Predefined Schema
  – Standard definition and interface language
  – Tight consistency
  – Well defined semantics
- NoSQL Database
  – No predefined Schema
  – Per-product definition and interface language
  – Getting an answer quickly is more important than getting a correct answer


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