

NSQL Introduction: MongoDB

High-Level Overview



mongoDB

{name: "mongo", type: "db"}

- Name comes from “Humongous” & huge data
- Written in C++, developed in 2009

Goal of MongoDB

- **Goal:** Bridge gap between key-value stores (fast and scalable) and relational databases (rich functionality).




What is MongoDB?

- **Definition:** MongoDB is an open source **document-oriented** database that is:
 - Scalable
 - Easy to work with (for developers)
- Instead of storing your data in tables and rows, in MongoDB you store **JSON-like documents** with dynamic schemas (**schema-free**).

What is MongoDB? (Cont'd)

- **Document-Oriented**

- Unit object is a document

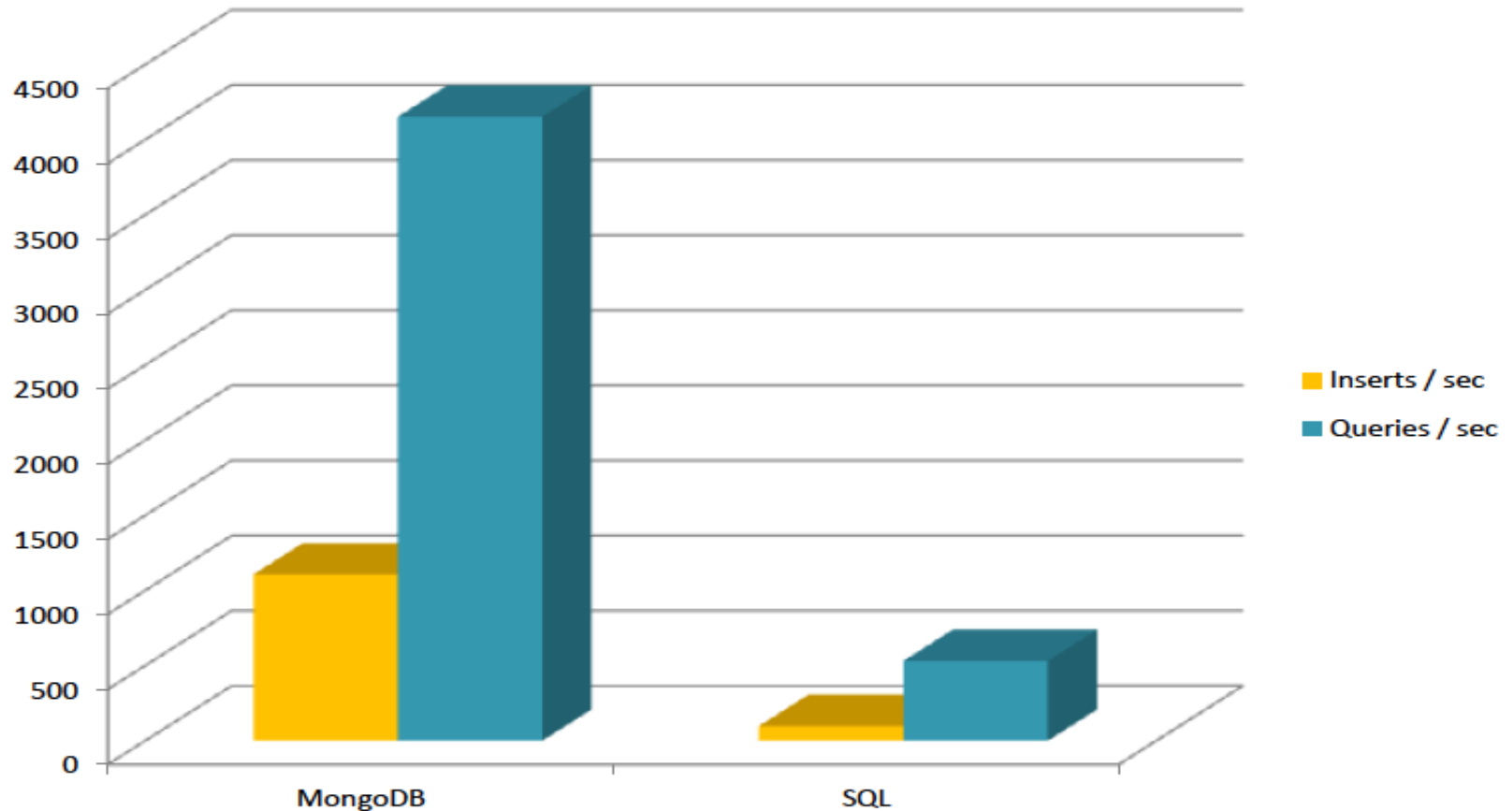


`{name: "will",
eyes: "blue",
birthplace: "NY",
aliases: ["bill", "la
ciacco"],
loc: [32.7, 63.4],
boss: "ben"}`

```
> db.user.findOne({age:39})  
{  
  "_id" : ObjectId("5114e0bd42..."),  
  "first" : "John",  
  "last" : "Doe",  
  "age" : 39,  
  "interests" : [  
    "Reading",  
    "Mountain Biking ]  
  "favorites": {  
    "color": "Blue",  
    "sport": "Soccer"}  
}
```

Is It Fast?

- For semi-structured data & relationships: Yes



Integration with Others

- [C](#)
- [C++](#)
- [Erlang](#)
- [Haskell](#)
- [Java](#)
- [Javascript](#)
- [.NET \(C# F#, PowerShell\)](#)
- [Node.js](#)
- [Perl](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)
- [Scala](#)



NoSQL DBs

What is NoSQL ?

- Stands for **Not Only SQL** or **None-relational** ?
- Class of **non-relational** data storage systems
 - Usually do not require a fixed table schema
 - Usually do not support concept of joins
 - Often distributed data storage systems

NoSQL: Categories

- Key-value



- Graph database



- Document-oriented



- Column family



Example: Column-Family (HBase)

ColumnFamily: Rockets		
Key	Value	
1	Name	Value
	name	Rocket
	toon	Rea
	inventoryQty	5
	brakes	
2	Name	
	name	
	toon	
	inventoryQty	
	brakes	
3	Name	Value
	name	Acme Je
	toon	Hot Ro
	inventoryQty	1
	wheels	1

MongoDB has
more flexible data
model & stronger
query interface

Typical APIs: `get(key)`, `put(key, value)`, `delete(key)`, ...

What is NoSQL

- Not **Only SQL**/None-relational
- **NoSQL systems relax ACID properties**
 - Traditionally, DB follows ACID properties:
 - Atomicity, Consistency, Isolation, and Durability
 - Now consider CAP properties & theorem

CAP Properties (by Eric Brewer)

- Three properties of a **distributed system** :
 - **Consistency** : all copies have same value
 - Each read receives the value from the most recent write, or an error
 - **Availability** : system can run even if parts have failed
 - **All** nodes still accept reads and writes and respond to these requests – but without the guarantee that it contains the most recent write
 - **Partition Tolerance** : tolerance to network being partitioned
 - Even if part is down, **others** can take over and continue to operate

<i>Consistency</i>	<i><u>Availability</u></i>	<i><u>Partition tolerance</u></i>
Every read receives the most recent write or an error	Every request receives a (non-error) response – without guarantee that it contains the most recent write	The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

What is Availability?

- Traditionally, thought of as the server/process availability being 99.999 %.
- I.e., it assumes that failures are rare
- In modern distributed systems:
 - Want a system that is resilient in the face of network disruption
 - Network partitioning has to be tolerated.
 - **Use Replication as Solution**

CAP Theorem

■ Three properties of a **distributed system** :

- ❑ **Consistency** : all copies have same value
- ❑ **Availability** : system can run even if parts have failed
 - ❑ All nodes can still accept reads and writes
- ❑ **Partition Tolerance** : tolerance to network being partitioned
 - ❑ Even if part is down, **others** can take over

■ CAP “Theorem” (by Eric Brewer)

- You can have at most two of these three properties for any system (so pick two in your system design).

CAP Theorem



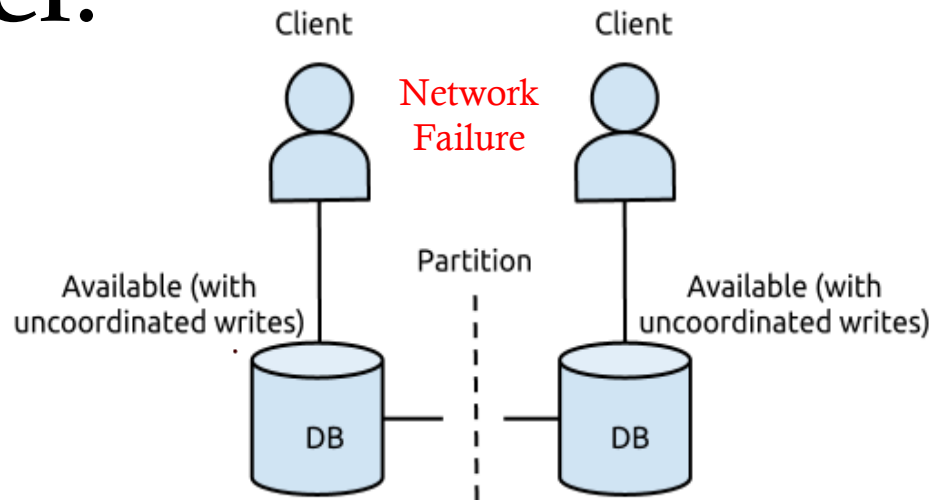
CA System

Guarantees strong consistency,
at the cost of not being able to
process requests unless all nodes
are able to talk to each other.

RDB chooses “consistency” over
“availability” when adopting
ACID principles.

Important

**In distributed systems, CA is not an option.
Either select AP or CP.**



Data divergence in an AP system during partition

- If select Availability ➔ Loosen Consistency (AP Design)
- If select Consistency ➔ Loosen Availability (CP Design)

CA vs AP

CA system :

guarantees strong consistency,
at the cost of not being able to process requests
unless all nodes are able to talk to each other.

AP system:

functions during the network split,
while providing various relaxed notions of
(eventual) consistency

Eventual Consistency

- When no updates occur for a long period of time:
 - Eventually all updates will propagate through the system and all the nodes will be consistent
- For a given accepted update and a given node:
 - Eventually either the accepted update reaches the node or the node is removed from service

Eventual Consistency: “BASE”

- **BASE** Property for AP systems :
 - Basically **A**vailable, **S**oft state, **E**ventual consistency
 - As opposed to ACID in RDBMS
 - Soft state: copies of a data item may be inconsistent
 - Eventual consistency – copies becomes consistent at some later time if there are no more updates to that data item
 - Systems designed with BASE, common for NoSQL systems, use availability over strong consistency.

What does NoSQL Not Provide?

- No ACID transactions
- No SQL
- No built-in join



Get Started with MongoDB

1. Study data examples and tutorials at:

<https://docs.mongodb.com/manual/reference/bios-example-collection/>

2. Practice with online terminal:

http://www.tutorialspoint.com/mongodb_terminal_online.php

3. Instructions to install on your own labtop, if you like:

<https://docs.mongodb.com/manual/administration/install-community>

MongoDB
will be next CS585 project
assignment.