Topics

1. What is NoSQL?
2. Brewer’s CAP Theorem
3. ACID vs. BASE
4. Eventual Consistency
5. Advantages of NoSQL
6. Disadvantages of NoSQL
7. NoSQL Database Taxonomy

What is NoSQL?

“NoSQL is the term used to designate database management systems that differ from classic relational database management systems in some way. These data stores may not require fixed table schemas, and usually avoid join operations and typically scale horizontally.”

-- Wikipedia

Non-relational may be more accurate than NoSQL, as some NoSQL DBs support a subset of SQL.
Why not stick to relational DBs?

Limited scalability
- Master/slave clusters are limited by write bandwidth and have a SPOF.
- Partitioning requires that you rewrite your application to find your data, and does not scale joins.

Availability is more important than consistency
- A RDBMS will make data unavailable until it is consistent on each node.
- In large clusters or if a node is down, the period of unavailability can be too long.

What problem do you have?

What data problem are you trying to solve?
- Fault tolerance
- High availability
- Consistency
- Scalability

Why not use a database that solves all of these problems at once?
- It’s impossible to make such a database!

You Still Need RDBMS
Brewer’s CAP Theorem

Web services can at most ensure 2 of the 3 following properties:

– **Strong consistency.** All clients perceive that a set of operations has occurred completely or not at all.
– **Availability.** All clients can read or write to some replica of the data, even if some nodes fail.
– **Partition tolerance.** Operations will complete, even if individual components are unavailable.

Distributed systems must be partition tolerant, so we have to choose between Consistency and Availability.

Network Partitions

A **network partition** occurs when a network failure causes a cluster to be divided into two subclusters that are isolated from each other. Partitions may occur within or btw data centers.

Brewer’s CAP Theorem

- Consistency
- Availability
- Partition Tolerance

Gossip protocol for LLC consistency

Eventual consistency
Business Perspective

Availability is more valuable than Consistency
– When in doubt, take the customer’s order.
– Apologize, fix, and compensate later.

Don’t say
– “Sorry, we can’t take your money. Our computers are down.”

ACID Properties

Atomicity—All data modifications within a transaction must happen completely or not at all. No partial transactions can be recorded.

Consistency—All changes to an instance of data must be reflected in all instances of that data.

Isolation—The Elements of a transaction should be isolated to the user performing the transaction until it is completed.

Durability—When a system failure occurs, the data in the DB must be accurate up to the last committed transaction before failure.

BASE

Basically Available
– System provides availability according to CAP.

Soft-state
– State of system changes over time even without new user inputs due to asynchronous updates.

Eventually Consistent
– Some nodes have current data
– Other nodes have previous data
– If no user updates are made, all nodes will eventually have identical current data.
**ACID vs. BASE**

**ACID**
- Consistency most important
- Strong consistency
- Pessimistic
- Data unavailable until correct answer on all nodes
- Slower
- Less scalable

**BASE**
- Availability most important
- Weak consistency
- Optimistic
- Approximate answers OK
- Faster
- Scalable

---

**Weak and Strong Consistency**

**Strong consistency**
- After an update occurs, any subsequent access will return the updated value.
- Data is unavailable until it is updated on all replicas and known to be correct.
- In CAP, consistency refers to strong consistency.

**Weak consistency**
- After an update occurs, subsequent accesses may see the old value or the updated value during a period of time called the *inconsistency window*.  

---

**Eventual Consistency**

*Eventual consistency* is a specific form of weak consistency, which guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value.

**Example: DNS**
- Owner makes DNS changes on local server
- Servers update name after old name TTL expires
- Clients see updated name after their caches expire
Eventual Consistency Variants

Causal consistency
— If a process A has communicated to process B that it has updated a data item, subsequent accesses by process B will return the updated value.

Read-your-writes consistency
— If process A has updated a data item, it will always receive the updated value when it accesses the data item in the future.

Session consistency
— As long as the session exists, system guarantees read-your-writes consistency. If session terminated because of an error, the guarantees do not apply to new sessions.

Eventual Consistency Variants

Monotonic read consistency
— If a process has seen a value for a data item, subsequent accesses will never return earlier values for that data item.

Monotonic write consistency
— Systems guarantees to serialize writes by the same process.

Combined variants
— Example: Monotonic reads plus read-your writes

Reality of Eventual Consistency in RDBMS

Whoops, not anymore!

There are 5 glass vases in inventory to RDBMS
Give up Partition Tolerance

Examples
- RDBMSs
- Master/slave RDBMS clusters

Traits
- All nodes must be in contact to function.
- 2-phase commit
- Cache invalidation protocols

Give up Availability

Examples
- Multi-master distributed DBs
- BigTable, Hbase, Neo4J
- MongoDB, Redis

Traits
- Shards
- Quorum/majority algorithms
- System down when transactions cannot complete across cluster.

Give up Consistency

Examples
- DNS
- Cassandra, Voldemort
- SimpleDB, CouchDB

Traits
- Eventually consistent
- Highly scalable
- Conflict resolution
- Optimistic
NoSQL Advantages

- Semi-structured data
- Alternative model paradigms
- Multi-valued properties
- Generalized analytics
- Version history
- Predictable scalability
- Scheme evolution

NoSQL Disadvantages

- Ease of Expression
- Lack of ACID Properties
- Eventual Consistency
- Normalized updates
- Standardization
- Access Control

NoSQL Taxonomy

Key/Value Store
- Access data by strings called keys.
- Data has no required format.

Document Store
- Access data by key or by search of document data.
- Document formats: XML, YAML, JSON

Graph
- Nodes (objects), properties, edges
- Object-oriented schema-less network
**Key/Value Stores**

A simple model
- Hash, map, or dictionary

Key/value stores are an old idea
- BerkeleyDB is a key/value store

Modern key value stores are different
- Non-ACID
- Highly scalable
- Highly available

---

**Cassandra: Key/Value Column Store**

---

**Dynamo: Reconciling Concurrent Writes**

---
Key/Value Stores

- BigTable
- Cassandra
- Dynamo
- Redis
- Voldemort

Document Stores

Store documents instead of rows
- No schema
- High flexibility
- Full-text and key search instead of SQL

Document structures include
- JSON
- XML
- YAML

JSON and XML

What’s JSON?

JSON, which stands for JavaScript Object Notation, is a data exchange format that serves as an alternative to XML. A basic JSON document for a contact in an address book might look like this:

```
{ "name": "John Doe", "age": 30, "phone": [ "555-1234", "555-5678" ], "email": [ "john.doe@example.com", "jdoe@example.net" ] }
```

That JSON data would likely be expressed in XML like this:

```
<contact>
  <name>John Doe</name>
  <age>30</age>
  <phone type="work">555-1234</phone>
  <phone type="home">555-5678</phone>
  <email type="work">john.doe@example.com</email>
  <email type="home">jdoe@example.net</email>
</contact>
```

JSON supports just a few data types, as opposed to the many offered by XML. These are number, Unicode string, boolean, array, object, and null.
CouchDB: JSON Documents in a B-Tree

CouchDB: Multi-Version Concurrency Control
- Writes don't have to wait for locks.
- Instead, writes create a new version of document.
- Before complete, reads retrieve old version.
- After complete, reads retrieve new version.
- System must later remove old documents.

Document Stores
- CouchDB
- MongoDB
- SimpleDB
- Terrastore
Graphs

A graph is a representation of a set of objects (nodes) connected by links (edges).

In a graph database, nodes contain data items called properties.

Social Graphs

Graph Databases
Why graph DBs over Relational DBs?

Many web applications have graph data
- Social networks, tagging systems, CMS, wikis

Graph operations are slow in RDBMS
- Graphs are recursive data structures
- Each traversal of an edge of a graph is a join

Schemas difficult to pre-conceive
- You don’t know which users will friend each other, which tags will be applied, which items in a CMS or wiki will be hyperlinked

Graph Databases
- AllegroGraph
- DEX
- FlockDB
  - Social network management from Twitter.
  - Twitter FlockDB cluster stored 13+ billion edges.
- HypergraphDB
- Neo4J
  - Dual-licensed open/commercial graph DB.
- VertexDB
Key Point: SQL and NoSQL

If SQL is your only storage tool, then all problems must look the same.

Using SQL and NoSQL storage technologies where appropriate, systems can have

- SQL for high consistency where needed
- NoSQL for high availability and scalability

Key Points

Brewer's CAP Theorem
- CA properties
- CP properties
- AP properties

Consistency
- Strong consistency
- Eventual consistency

NoSQL Taxonomy
- Key/value
- Document
- Graph

References