Fundamentals of Database Systems [NoSQL]

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

October, 2019

Limitations of relational databases

Relational databases are entirely dependent on relations

- Relational databases cannot deal with data that are infeasible of fitting to relations.
- They cannot handle unpredictable, unstructured or messy data.
- They may require vertical and horizontal expansion of servers, as the data or processing requirements grow.

What are beyond relational databases?

Non-relational distributable databases depending on unstructured data. It's high performance with high availability, and offers rich query language and easy scalability.

CAP Theorem: Consistency, availability and partition tolerance.

NoSQL

Examples include - MongoDB, Hadoop, etc.

Note: Non-relational databases may not provide full ACID (atomicity, consistency, isolation, durability) guarantees, but still has a distributed and fault tolerant architecture.

Basics of NoSQL

NoSQL (often referred to as Not only SQL) is a broad class of data management systems where the data is partitioned across a set of servers, where no server plays a privileged role.

– NoSQL database management systems are useful when working with a huge quantity of data (especially big data), when the data's nature does not require a relational model
– The data can be structured, but NoSQL is used when what really matters is the ability to store and retrieve great quantities of data (e.g., millions of keyvalue in one or a few associative arrays)
– This is particularly useful for statistical or real-time analysis of growing lists of elements (e.g., huge number of Twitter posts, Internet server logs from a large group of users)
– It provides flexibility of the data model (e.g., ERP and BPM)

History

"It's not about BASE vs. ACID" – Emin Gün Sirer.

1998: Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface.
2009: Eric Evans, a Rackspace employee, reintroduced the term NoSQL to label the emergence of a growing number of non-relational, distributed data stores.
2011: Work began on UnQL (Unstructured Query Language), a specification for a query language for NoSQL databases, designed to query collections (versus tables) of documents (versus rows) with loosely defined fields (versus columns).

Note: BASE refers to basically available, soft-state and eventual consistency.

NoSQL – Advantages

- Elastic scaling – database can be distributed across multiple hosts as load increases (scale out)
- Handling big data – massive volume of data and transactions can be managed
- No DBA – the mechanism of automatic repair, data distribution, and simpler data models lead to lower administration and tuning requirements
- Low cost – clusters of cheap commodity servers are typically used to manage the exploding data and transaction volumes
- Flexible data models – schemaless relaxed (or even nonexistent) data model restrictions are used
- Caching layer – employs a useful cache layer
- Real time analysis and streaming model – distributed processing makes it faster

NoSQL – Challenges

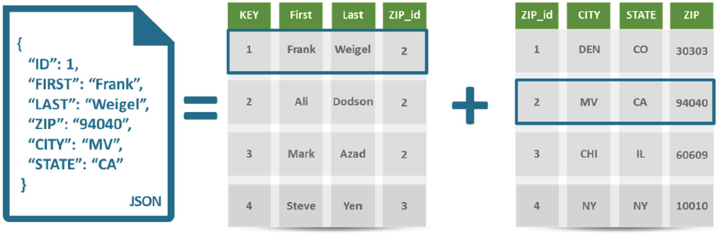
- No normalization, referential integrity, ACID – basic criteria of relational databases are not satisfied
- Poor data security – distributed data becomes less secure and might get lost easily
- Immaturity – most NoSQL alternatives are in pre-production versions with many key features yet to be implemented
- Poor support – NoSQL systems are mostly open source projects without the global reach, support resources, or credibility
- Weakness in analytics and business intelligence – commonly used BI tools do not provide connectivity to NoSQL
- Targeting no administration – the goal of providing a zero-admin solution still has not been fulfilled
- Limited expertise – almost every NoSQL developer is in a learning mode

Taxonomy of NoSQL

- **Column:** Accumulo, Cassandra, HBase, etc.
- **Document:** Clusterpoint, Couchbase, MarkLogic, MongoDB, etc.
- **Key-value:** Dynamo, MemcacheDB, Project Voldemort, Redis, Riak, etc.
- **Graph:** Allegro, Neo4J, OrientDB, Virtuoso, etc.

JSON format

JSON (stands for JavaScript Object Notation) is a binary and typed data model that supports the data types list, map, date, Boolean, and numbers (with different precisions). It can be serialized to and deserialized from bytes as well as strings.



```
{
  "ID": 1,
  "FIRST": "Frank",
  "LAST": "Weigel",
  "ZIP": "94040",
  "CITY": "MV",
  "STATE": "CA"
}
```

KEY	First	Last	ZIP_id
1	Frank	Weigel	2
2	Ali	Dodson	2
3	Mark	Azad	2
4	Steve	Yen	3

ZIP_id	CITY	STATE	ZIP
1	DEN	CO	30303
2	MV	CA	94040
3	CHI	IL	60609
4	NY	NY	10010

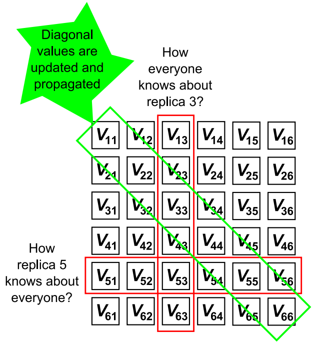
Some concepts in NoSQL – Versioning of datasets

To process “concurrent” modifications and versions of datasets in a distributed environment, different schemes can be employed. Some of these are listed below.

- **Timestamps:** It helps to develop a chronological order but rely on synchronized clocks and don’t capture causality.
- **Optimistic Locking:** It implies that a unique counter or clock value is saved for each piece of data. When a client tries to update a dataset it has to provide the counter/clock-value of the revision it likes to update. It does not work well in a distributed and dynamic scenario where servers show up and go away often and without prior notice.
- **Multiversion Storage:** It means storing a timestamp for each table cell, thus providing optimistic concurrency control with compare-and-swap on timestamps.
- **Vector Clocks:** It is an alternative approach to capture order and allow reasoning between updates in a distributed system.

Some concepts in NoSQL – Vector Clocks

A vector clock is defined as a tuple $V_{i0}, V_{i1}, \ldots, V_{in}$ of clock values (may be real timestamps, version/revision numbers or some other ordinal values) collected from each node $i = 1, \ldots, n$, which represent the state of itself and the other (replica) nodes' state.


Diagonal values are updated and propagated
How everyone knows about replica 3?
How replica 5 knows about everyone?
V_{11} V_{12} V_{13} V_{14} V_{15} V_{16}
V_{21} V_{22} V_{23} V_{24} V_{25} V_{26}
V_{31} V_{32} V_{33} V_{34} V_{35} V_{36}
V_{41} V_{42} V_{43} V_{44} V_{45} V_{46}
V_{51} V_{52} V_{53} V_{54} V_{55} V_{56}
V_{61} V_{62} V_{63} V_{64} V_{65} V_{66}

An overview of a vector clock managing six nodes

Some concepts in NoSQL – Vector Clocks

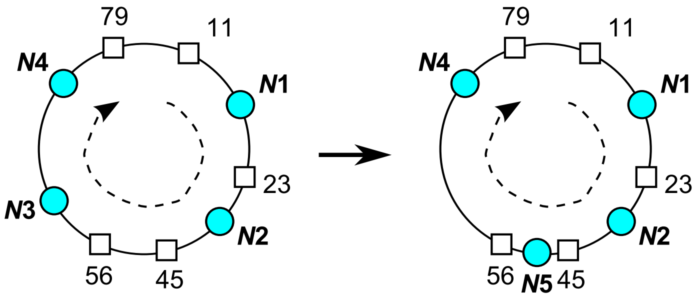
Vector clocks can be utilized “to resolve consistency between writes on multiple replicas”.

Vector clocks are updated in a way defined by the following rules:

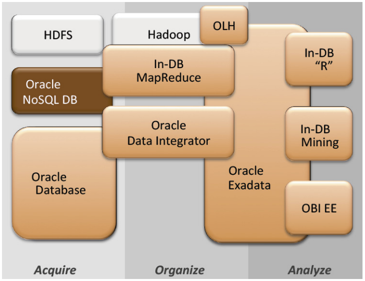
1. If an internal operation happens at node i, this node will increment its clock V_{ii} (immediate execution).
2. If node i sends a message to node k, it first advances its own clock value V_{ii} and attaches the vector clock V_i (about his internal state and his view of the other nodes) to the message sent to node k.
3. If node i receives a message from node j, it first advances its vector clock V_{ii} and then merges its own vector clock with the vector clock $V_{message}$ attached to the message from node j so that $V_i = \max(V_i, V_{message})$. To compare two vector clocks V_i and V_j in order to derive a partial ordering, the following rule is applied $V_i > V_j$, if $\forall k V_{ik} > V_{jk}$, If neither $V_i > V_j$ nor $V_i < V_j$ applies, a conflict caused by concurrent updates has occurred and needs to be resolved (e.g., by a client application).

Consistent hashing

Which object is mapped to which node is determined by moving clockwise around the ring. Therefore, if *N*3 leaves and *N*5 enters the system, the object 45 will get mapped to node *N*5.



Oracle NoSQL database



Oracle NoSQL database seamlessly integrates into an enterprise application architecture

Oracle-provided adapters allow the Oracle NoSQL database to integrate with a Hadoop MapReduce framework or with the Oracle database in-database MapReduce, data mining, R-based analytics, or whatever business needs demand.