NoSQL Overview

# Abstract

The quest for scaling in quantity of data, number of access, and redundancy has fueled a migration from files to relational database systems, and currently to NoSQL database systems. Ben Stopford's post describing the basic forces driving these transitions can give us a framework for evaluating possible NoSQL storage systems (please see reference below.)

His blog post makes detailed observations about scaling data management platforms:

* Sequential streaming access is faster than random access for both read and writes of main memory and disk.
* Indexing data increases read performance but slows write; so keep optimized index structures in memory.
* Minimize the amount read; store data by column and retrieve required columns sequentially with filtering.
* Avoid joins.
* Mind consistency vs partitioning (CAP) tradeoffs.

Our article collects attributes of many NoSQL systems to help one select an approach that correctly matches use cases of:

* Query complexity.
* Ingestion speed.
* Replication needs.
* Consistency vs partitioning.

# Modern Persistence

Much larger volumes of data are captured in modern computing and they require more scalable solutions than traditionally offered by RDBMS products. We examine relational, key/value, document, and columnar databases. These four types are collectively called NoSQL. NoSQL databases are distributed in order to handle this increased volume of data and resulting ingestion and query requests.

Our overview of current NoSQL technology examines these topics:

* Attributes of the various database system types.
* The CAP theorem – an observation about characteristics of distributed computing.
* Characteristics of individual NoSQL databases.

# High-Level Comparison

We compare RDBMS approaches to NoSQL approaches with respect to scaling, query capability, and data structure.

#### RDBMS Overview

RDBMS systems tend to store structured data, meaning the data has a schema, and offer ACID properties (Atomic, Consistent, Isolated, and Durable.) They offer excellent transactional capabilities with guaranteed preservation of order of updates and consistency of reads.

Unfortunately, providing these capabilities requires significant overhead and present challenges to scaling by partitioning. Finally, efficient table joins, the workhorse of normalized data representation, are difficult to scale.

In summary, in spite of the above scaling problems, RDBMS have several important features:

* Have a strong mathematical basis for storing and manipulating data.
* Validate structured data.
* Declarative syntax with a well-known language in Structured Query Language (SQL).
* Provide ACID transactional properties.
* Provide mature query capabilities with support for indexing to accelerate queries.

#### NoSQL Overview

NoSQL databases primarily address distributed, large-scale data storage and unstructured data types. The definition of ***NoSQL***, as taken from NoSQL.org, is:

*NOSQL* is the next Generation Database Management Systems mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable.

While achieving scaling and data format flexibility, there are associated costs for this technology:

* Less capable query languages.
* No standard or common query language.
* API and storage models differ, so there is a lack of portability between NoSQL databases.

#### Some NoSQL Systems

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **NoSQL Family** | **Name** | **Partition Type** | **Updates vs Appends** | **Index Support** | **Storage Model** | **Schema Support** |
| **Wide Column Store / Column Families** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **Document Store** | MongoDB (JSON) | Ranged & Hashed | insert, update, delete, select | Primary & secondary indices on fields. |  |  |
|  |  |  |  |  |  |  |
| **Key Value / Tuple Store** | Redis |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **In-memory Store** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **Graph Database Management Systems** |  |  |  |  |  |  |

We will elaborate on the NoSQL concepts mentioned in the table as we continue. We can now examine ***The CAP Theorem***; which is the basic theoretical framework used to evaluate distributed persistence.

# The CAP Theorem Summary

Modern distributed persistence is required to manage the large volume of data generated by 21rst century computing. Distributed computing forces network partitioning; and that adds failure modes and latency compared to single machine implementations.

CAP Theorem: A distributed database system can have at most two of these three properties: Consistency, Availability and Partition Tolerance. The property definitions are:

1. ***Consistency***: This condition states that all nodes see the same data at the same time.
2. ***Availability***: This condition states that every request gets a response on success/failure. Achieving availability in a distributed system requires that the system remains operational 100% of the time.
3. ***Partition Tolerance:*** This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes. A system that is partition-tolerant can sustain any amount of network failure that doesn’t result in a failure of the entire network.

#### Cap From Wiki

No distributed system is safe from network failures, thus network partitioning generally has to be tolerated. In the presence of a partition, one is then left with two options: consistency or availability. When choosing consistency over availability, the system will return an error or a time-out if particular information cannot be guaranteed to be up to date due to network partitioning. When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.

In the absence of network failure – that is, when the distributed system is running normally – both availability and consistency can be satisfied.

CAP is frequently misunderstood as if one has to choose to abandon one of the three guarantees at all times. In fact, the choice is really between consistency and availability only when a network partition or failure happens; at all other times, no trade-off has to be made.

Database systems designed with traditional ACID guarantees in mind such as RDBMS choose consistency over availability, whereas systems designed around the BASE philosophy, common in the NoSQL movement for example, choose availability over consistency.

The PACELC theorem builds on CAP by stating that even in the absence of partitioning, another trade-off between latency and consistency occurs.

#### Consistency vs Availability

Relational databases stress the “consistency” of data in a database after a transaction is complete. During a transaction, the degree of consistency desired is specified by the transaction isolation level. In the extreme, this can be “serialized”, meaning that transactions are ordered and one completes before the next is undertaken.

A Wikipedia reference to ***Consistency*** in a (NoSQL) database is given below, and is complicated. Here is a simplified explanation taken from Book #1.

A fully consistent database has reads return the value of the last write, and all reads from a given time epoch return the same value from all partitions in the database.

***Consistency*** is not “all or nothing”; so the major consistency models are:

* ***Strong consistency***: A database is strongly consistent if updates to a given key are ordered, and reads reflect the latest update that has been accepted.
* ***Timeline consistency***: A is timeline consistent if updates to a given key are ordered in all partitions, but reads at a given partition may not reflect the latest accepted update.
* ***Eventual consistency***: An eventually consistent database makes no guarantees about whether updates will be applied in order in all partitions, nor does it make guarantees about when a read would reflect a prior update.

A database's ***availability*** refers to the system's ability to complete a certain operation. Again, we use a definition from book #1:

An available database is generally available for both reads and writes, but may become unavailable for:

* Writes while being available for reads.
* Reads while being available for writes.
* Admin operations while being available for data operations.

## From *Seven NoSQL Databases in a Week* – NoSQL Overview

Persistence systems may be categorized as: relational, key/value, document, columnar, and graph. The last four types are collectively called NoSQL. It is ironic that all NoSQL databases are developing SQL-like interfaces to leverage the wide-spread experience of data users with SQL.

NoSQL database distinguishing characteristics are:

* Consistency versus availability tradeoffs.
* ACID guarantees.
* Hash versus range partition strategy.
* In-place updates versus appends.
* Row versus column versus column-family storage models.
* Strongly versus loosely enforced schemas.

Amazon Dynamo uses Eventual Consistency to come close to getting all three properties. The paper ***Dynamo: Amazon’s Highly Available Key-value Store*** is worth reading when learning about NoSQL databases and distributed systems.

# DynamoDB

# CosmoDB

# References

#### Books

1. *Seven NoSQL Databases in a Week*, Aaron Ploetz, © 2018 (ISBN 978-1-78728-886-7).
2. *Making Sense of NoSQL*, Dan McCreary and Ann Kelly, © 2014 (ISBN 9781617291074).
3. *The Beginner’s Guide to NoSQL*, Dr. Micheal Crabb (see <https://devopsmates.com/free-ebook-beginners-guide-nosql/>).
4. *Guide to NoSQL with Azure Cosmos DB*, © 2018 (ISBN 978-1-78961-289-9).

#### **Original Sources Links**:

1. ***Consistency*** conceptually is a little complex in the context of databases, but a good technical article on Wiki is <https://en.wikipedia.org/wiki/Eventual_consistency>.
2. ***Wiki CAP***: <https://en.wikipedia.org/wiki/CAP_theorem>.
3. ***Amazon’s Dynamo DB***: <https://www.allthingsdistributed.com/2007/10/amazons_dynamo.html>.
4. ***Dynamo: Amazon’s Highly Available Key-value Store***: <https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>.

#### Curated Discussions:

1. ***Ben Stopford's epic post on Elements of Scale: Composing and Scaling Data Platforms***: <http://highscalability.com/blog/2015/5/4/elements-of-scale-composing-and-scaling-data-platforms.html>.
2. ***NoSQL vs. RDBMS: Apples and Oranges?***: <https://it.toolbox.com/blogs/madgreek/nosql-vs-rdbms-apples-and-oranges-032810>.
3. ***Your Ultimate Guide to the Non-Relational Universe!***: <http://nosql-database.org/>.
4. ***Cloud NoSQL DB comparison***: <https://searchcloudcomputing.techtarget.com/tip/Compare-NoSQL-database-types-in-the-cloud>.
5. ***A great NoSQL overview***: <https://medium.com/@Grigorkh/nosql-in-the-cloud-a-scalable-alternative-to-relational-databases-f5dc35555fdb>.
6. ***Good SQL vs NoSQL compariso***n: <https://blog.panoply.io/sql-or-nosql-that-is-the-question>.
7. ***Index usage in SQL vs NoSQL***: <https://sql-vs-nosql.blogspot.com/2013/11/indexes-comparison-mongodb-vs-mssqlserver.html>.

## CAP tutorials:

1. Overview: <https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e>.

## DynamoDB

1. ***DynamoDB detailed discussion***: <https://blog.yugabyte.com/11-things-you-wish-you-knew-before-starting-with-dynamodb/>.
2. ***Is DynamoDB for you?***: <https://aws.amazon.com/blogs/database/how-to-determine-if-amazon-dynamodb-is-appropriate-for-your-needs-and-then-plan-your-migration/>.
3. ***Download Local Development Version***: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.html>.
4. ***AWS NoSQL description***: <https://aws.amazon.com/nosql/>.