INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Dharwad
ज्ञानेन विकासः

No-SQL and CAPs Theorem

Dr. Animesh Chaturvedi

Assistant Professor: IIIT Dharwad

Post Doctorate: King's College London & The Alan Turing Institute

PhD: IIT Indore MTech: IIITDM Jabalpur

KING'S College LONDON

Indian Institute of Technology Indore
भारतीय प्रौद्योगिकी संस्थान इंदौर

PDPM
Indian Institute of Information Technology, Design and Manufacturing, Jabalpur

The Alan Turing Institute

Not Only SQL (No-SQL)

No-SQL

- “Not Only SQL” i.e. database systems that works on other than the tabular relations used in relational databases.
- No-SQL is a non-relational database system, which may not necessarily use SQL.
- No-SQL databases are mostly used for real-time and big data applications.
- No-SQL systems may also support SQL-like query languages.
- No-SQL database systems has data structures different from relational databases (key-value, graph, document).
- No-SQL allow a dynamic schema for unstructured data.
- No-SQL databases allow to add new attributes and fields.

No-SQL Properties

- High usage in big data and real-time web applications.
- Mostly No-SQL databases systems follows the CAP theorem.
- Huddle in adoption of No-SQL stores are low-level query languages, lack of standardized interfaces, and huge investments in existing SQL.
- Less need to pre-plan and pre-organize data, and it's easier to make modifications.
- Some No-SQL systems do not provides all four ACID properties together (atomicity, consistency, isolation and durability).
- "Horizontal" scaling to make clusters of machines, making operations faster.
  - Means add additional servers or nodes as needed to increase load.
- "Vertical" scaling is mostly necessity of SQL based Relational Database
  - to follow ACID properties.

Categories of No-SQL databases

- **Column Oriented:** data is stored as columns instead of rows
  - data is stored in cells grouped in a virtually unlimited number of columns rather than rows.
  - Accumulo, Cassandra, Druid, HBase,Vertica
- **Document-Oriented:**
  - use documents to hold and encode data in standard formats including XML,YAML, JSON (JavaScript Object Notation) and BSON (Binary JSON).
  - documents within a single database can have different data types
  - Clusterpoint, CouchDB, Couchbase, MarkLogic, MongoDB, OrientDB
- **Key-value:** contains many different key value pairs
  - use an associative array (also known as a dictionary or map)
  - Dynamo, FoundationDB, MemcacheDB, Redis, Riak, FairCom ctreeACE, Aerospike, OrientDB
- **Graph:** used to store data related to connections or networks
  - represent data on a graph that shows how different sets of data relate to each other
  - Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog, RedisGraph
- **Multi-model:** OrientDB, FoundationDB, ArangoDB, Alchemy Database, CortexDB.

Structured Query Language (SQL) Transactions

Transaction Concept

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.
- E.g., transaction to transfer $50 from account A to account B:
  1. **read**(A)
  2. $A := A - 50$
  3. **write**(A)
  4. **read**(B)
  5. $B := B + 50$
  6. **write**(B)
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

Example of Fund Transfer (Cont.)

- **Atomicity requirement**
  - If the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state
    - Failure could be due to software or hardware
  - The system should ensure that updates of a partially executed transaction are not reflected in the database
- **Durability requirement** — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures.

Example of Fund Transfer (Cont.)

- **Consistency requirement** in above example:
  - The sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
  - Explicitly specified integrity constraints such as primary keys and foreign keys
  - Implicit integrity constraints
    - e.g., sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
  - A transaction must see a consistent database.
  - During transaction execution the database may be temporarily inconsistent.
  - When the transaction completes successfully the database must be consistent
    - Erroneous transaction logic can lead to inconsistency

Example of Fund Transfer (Cont.)

- **Isolation requirement** — if between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

	T1	T2
1.	**read**(A)	
2.	$A := A - 50$	
3.	**write**(A)	
		read(A), read(B), print(A+B)
4.	**read**(B)	
5.	$B := B + 50$	
6.	**write**(B)	

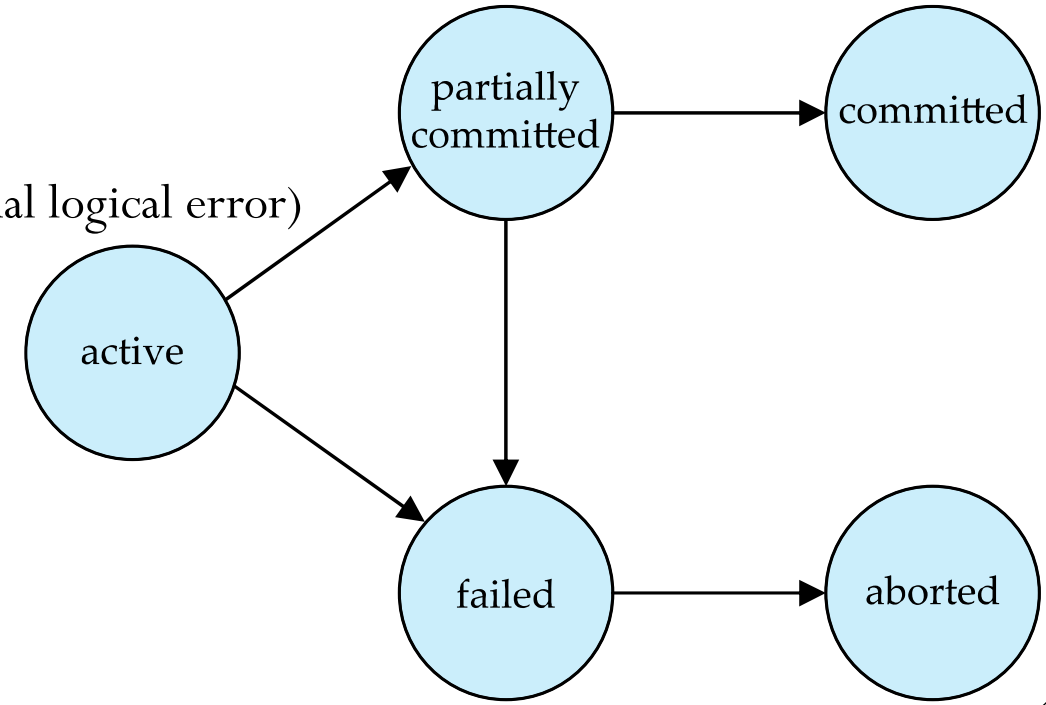
- Isolation can be ensured trivially by running transactions **serially**
  - That is, one after the other.
- However, executing multiple transactions concurrently has significant benefits.

ACID Properties

- A **transaction** is a unit of program execution that accesses and possibly updates various data items.
- To preserve the integrity of data the database system must ensure:
- **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are.
- **Consistency.** Execution of a transaction in isolation preserves the consistency of the database.
- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
  - That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.
- **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Transaction State

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction.
  - Two options after it has been aborted:
    - Restart the transaction (Can be done only if no internal logical error)
    - Kill the transaction
- **Committed** – after successful completion.



Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system.
- Advantages are:
  - **Increased processor and disk utilization**, leading to better transaction *throughput*
  - **Reduced average response time** for transactions: short transactions need not wait behind long ones.
- **Concurrency control schemes** – mechanisms to achieve isolation
  - That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database.

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a commit instructions as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- **A transaction that fails to successfully complete its execution will have an abort instruction as the last statement**

CAPS Theorem

CAPS Theorem

- Also known as Brewer's theorem
- SQL follows ACID properties,
- No-SQL follows the CAP theory
- The CAP theorem says that
- “It is impossible for a distributed computer system to simultaneously provide three (Consistency, Availability, and Partition) together with guarantees in single instance”
- Although some No-SQL databases — such as IBM’s DB2, MongoDB, AWS’s DynamoDB and Apache’s CouchDB — can also integrate and follow ACID rules

CAPS Theorem

- A distributed data systems allow a trade-off that can guarantee only two of the following three properties (which form the acronym CAP) at any one time:
- **Consistency**: All nodes can view the same data at the same time. Every request receives either the most recent result or an error. MongoDB is an example of a strongly consistent system, whereas others such as Cassandra offer eventual consistency.
- **Availability**: A guarantee that every request will receives response for success or failure. Every request has a non-error result.
- **Partition**: The system continues to operate irrespective of the loss or failure of a node. Try to achieve Partition tolerance means any delays or losses between nodes do not interrupt the system operation.

When to use SQL and When to use No-SQL

When to use SQL

- SQL is a good choice when working with related data.
- Relational databases are efficient, flexible and easily accessed by any application.
- Benefit of a relational database: when one user updates a specific record, every instance of the database automatically refreshes, and that information is provided in real-time.
- SQL and a relational database make it easy to handle a great deal of information, scale as necessary and allow flexible access to data — only needing to update data once instead of changing multiple files, for instance.
- SQL is best for assessing data integrity. Each piece of information is stored in a single place, there is no problem with former versions confusing the picture.
- Tech companies use SQL, including Uber, Netflix and Airbnb.
- Companies like Google, Facebook and Amazon, which build their own database systems, also use SQL to query and analyze data.

When to use No-SQL

- While SQL is valued for ensuring data validity, No-SQL is good when it's more important that the availability of big data is fast.
- No-SQL is easy-to-use, flexible and offers high performance. It's also a good choice when a company will need to scale because of changing requirements.
- No-SQL is also a good choice when there are large amounts of (or ever-changing) data sets or when working with flexible data models or needs that do not fit into a relational model.
- No-SQL are a good fit for quick access to a key-value store without strong integrity guarantees.
- When a complex or flexible search across a lot of data is needed, then Elastic Search with No-SQL is a good choice.

When to use No-SQL

- Scalability is a significant benefit of No-SQL databases.
- Unlike with SQL, their built-in sharding and high availability requirements allow horizontal scaling.
- No-SQL databases like Cassandra, developed by Facebook, handle massive amounts of data spread across many servers, having no single points of failure and providing maximum availability.
- Companies (include Amazon, Google and Netflix) use No-SQL systems because they are dependent on large volumes of data not suited to a relational database.
- The more extensive the dataset, the more likely that No-SQL is a better choice.

SQL vs No-SQL

- The difference between SQL and No-SQL databases is really just a comparison of relational vs. non-relational databases.
- With the rise of social media, Ecommerce, search, and the explosion of data, SQL was struggling to manage all the requests, transactions, and activity occurring online. No-SQL is designed to manage lots of traffic and data.
- Each type of No-SQL database stores data differently and is selected and used in different contexts.

Data structure and Language

	SQL databases	No-SQL databases
Data structure	The SQL data structure is based on a relational model that normalizes data across strictly defined tables and standardizes the relationships between those tables, making SQL databases well suited to highly structured data.	The No-SQL data structure does not require a normalized configuration or adhere to a relational model but is instead flexible enough to accommodate different models, including key-value, document, column-oriented, and graph.
Language	SQL databases are all about the SQL language. Some relational database products support pure SQL. However, all SQL databases support the core ANSI/ISO language elements.	No-SQL databases are not locked into one language. The language used depends on the type of No-SQL database, the individual implementation, and the specific operation. For example, MongoDB stores all documents in a JSON format, with queries based on the JavaScript programming language.

Schemas and Data integrity

	SQL databases	No-SQL databases
Schemas	An SQL database requires a predefined schema that determines how tables are configured and data is stored, resulting in a rigid structure that helps to optimize storage and ensure data integrity, but limits flexibility.	A No-SQL database uses a dynamic schema that requires no predefined data structure, resulting in a high degree of flexibility, such as being able to add documents with different fields to the same database.
Data integrity	SQL databases deliver a high degree of data integrity, adhering to the principles of atomicity, consistency, isolation, and durability (ACID), which are essential when supporting workloads such as financial transactions.	It can be difficult for No-SQL databases to deliver the same level of data integrity as SQL databases, with most adhering to BASE principles (basic availability, soft state, and eventual consistency), which means data in a distributed environment might be temporarily inconsistent.

Scalability, Querying, and Maturity

	SQL databases	No-SQL databases
Scalability	SQL databases primarily scale vertically, which means they can be easily scaled up by adding resources such as CPUs or memory, but SQL databases are not very efficient at scaling horizontally, making them ill-suited for large, distributed data sets.	No-SQL databases can scale horizontally very efficiently across systems and locations, making it possible to accommodate large stores of distributed data, while supporting increased levels of traffic.
Querying	SQL databases are efficient at processing queries and joining data across tables, making it easier to perform complex queries against structured data, including ad hoc requests.	No-SQL databases lack consistency across products and typically require more work to query data, particular as query complexity increases.
Maturity	SQL databases are built on mature technologies that are well known and supported by large developer communities.	No-SQL technologies are making fast inroads into the industry, with developer communities constantly growing.

SQL vs No-SQL

Consider SQL databases when…	**Consider No-SQL databases when…**
SQL came first. SQL is used to communicate with relational databases. Relational databases store data in a very organized, but also rigid way.	No-SQL, earning it's name by being "not only SQL" makes it easier to store all different types of data together. It's used for its flexibility and therefore speed and scalability in managing large volumes of data.
Data is highly structured, and that structure does not change frequently	Working with large amounts of unstructured or semi-structured data that does not fit the relational model
Support transaction-oriented systems such as accounting or financial applications	Require the flexibility of a dynamic schema or want more choice over the data model
Require a high degree of data integrity and security	Require a database system that can be scaled horizontally, perhaps across multiple geographic locations
Routinely perform complex queries, including ad-hoc requests	Want to streamline development and avoid the overhead of a more structured approach
Used for applications require the level of data integrity offered by SQL databases	Used for application require the scale-out capabilities that No-SQL offers

Database Vendors

- Vendors have been steadily incorporating features into their products to make them more universal.
- For example, MongoDB now supports multi-document ACID transactions, and MySQL now includes a native JSON data type for storing and validating JSON documents.
- Graph databases are designed for data whose relations are well represented as a graph consisting of elements connected by a finite number of relations. Examples of data include social networks, public transport links, road maps, network topologies, etc.

Benchmarking

Data model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–value store	high	high	high	none	variable (none)
Column-oriented store	high	high	moderate	low	minimal
Document-oriented store	high	variable (high)	high	low	variable (low)
Graph database	variable	variable	high	high	graph theory
Relational database	variable	variable	low	moderate	relational algebra

Reference

- Avi Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, Seventh Edition, McGraw-Hill, ISBN 9780078022159
- Wikipedia https://en.wikipedia.org/wiki/NoSQL
- Robert Sheldon, 13 April 2021 https://www.red-gate.com/simple-talk/databases/nosql/how-to-choose-between-sql-and-nosql-databases/
- Benjamin Anderson “STSM, IBM Cloud Databases”, Brad Nicholson “Senior Database Engineer, IBM Cloud Databases”, https://www.ibm.com/cloud/blog/sql-vs-nosql

ขอบคุณ
Thai

Grazie
Italian

תודה רבה
Hebrew

धन्यवादः
Sanskrit

ಧನ್ಯವಾದಗಳು
Kannada

Ευχαριστώ
Greek

Thank You
English

Gracias
Spanish

Спасибо
Russian

Obrigado
Portuguese

شكراً
Arabic

https://sites.google.com/site/animeshchaturvedi07

Merci
French

多謝
Traditional
Chinese

धन्यवाद
Hindi

Danke
German

多谢
Simplified
Chinese

நன்றி
Tamil
Tamil

ありがとうございました
Japanese

감사합니다
Korean