Tutorial 6: Databases in Cloud Computing

Question Set:

Q1. What are the differences between SQL and NoSQL?

ANSWER:

SQL databases are primarily called Relational Databases (RDBMS), whereas NoSQL databases are primarily called non-relational or distributed databases.

SQL databases are table-based, whereas NoSQL databases are document-based, key-value pairs, graph databases, or wide-column stores. This means that SQL databases represent data in the form of tables, which consists of n number of rows of data. In contrast, NoSQL databases are the collection of key-value pair, documents, graph databases, or wide-column stores that do not have standard schema definitions that it needs to adhere to.

SQL databases have a predefined schema, whereas NoSQL databases have a dynamic schema for unstructured data.

SQL databases are vertically scalable, whereas NoSQL databases are horizontally scalable. SQL databases are scaled by increasing the horsepower of the hardware. NoSQL databases are scaled by increasing the database servers in the pool of resources to reduce the load.

SQL databases use SQL (structured query language) for defining and manipulating the data, which is very powerful. In a NoSQL database, queries are focused on the collection of documents. Sometimes, it is also called UnQL (Unstructured Query Language). The syntax of using UnQL varies from database to database.

SQL database examples: MySQL, Oracle, SQLite, Postgres, and MS-SQL. NoSQL database examples: MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j and CouchDB

SQL databases are not the best fit for the data type to be stored for hierarchical data storage. However, the NoSQL database fits better for hierarchical data storage as it follows the key-value pair way of storing data like JSON. NoSQL databases are highly preferred for large data sets (i.e., for big data). HBase is an example of this purpose.

For scalability: In most typical situations, SQL databases are vertically scalable. You can manage the increasing load by increasing the CPU, RAM, SSD, etc., on a single server. On the other hand, NoSQL databases are horizontally scalable. You can easily add a few more servers to your NoSQL database infrastructure to handle the large traffic.

For high transaction-based applications, SQL databases are the best fit for heavy-duty transactional-type applications, as they are more stable and promise the atomicity and integrity of the data. While you can use NoSQL for transaction purposes, it is still not comparable and sable enough in high load and for complex transactional applications.

For support: Excellent support is available for all SQL databases from their vendors. There are also a lot of independent consultations that can help you with SQL databases for large-scale deployments. For some NoSQL databases, you still must rely on community support, and only limited outside experts are available for you to set up and deploy your large-scale NoSQL deployments.

For properties: SQL databases emphasize ACID properties (Atomicity, Consistency, Isolation, and Durability), whereas the NoSQL database follows the Brewers CAP theorem (Consistency, Availability, and Partition tolerance)

For DB types: On a high level, we can classify SQL databases as either open-source or close-sourced from commercial vendors. NoSQL databases can be classified based on the way of storing data as graph databases, key-value store databases, document store databases, column store databases, and XML databases.

Q2. What are ACID properties?

ANSWER:

In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc. In the context of databases, a sequence of database operations that satisfies the ACID properties (and these can be perceived as a single logical operation on the data) is called a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.

Atomicity:
Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely or fails completely. If any statements constituting a transaction fail to complete, the entire transaction fails, and the database is left unchanged. An atomic system must guarantee atomicity, including power failures, errors, and crashes.

Consistency:
Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This prevents database corruption by an illegal transaction but does not guarantee that a transaction is correct. Referential integrity guarantees the primary key - foreign key relationship.

Isolation:
Transactions are often executed concurrently (e.g., multiple transactions reading and writing to a table simultaneously). Isolation ensures that the concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Isolation is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions.

Durability:
Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash). This usually means that completed transactions (or their effects) are recorded in non-volatile memory.

Q3. What are CAP theorem and BASE properties?

ANSWER:

In theoretical computer science, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

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

Eventually consistent services are often classified as providing **BASE** (Basically Available, Soft state, Eventual consistency) semantics, in contrast to traditional ACID (Atomicity, Consistency, Isolation, Durability) guarantees. In chemistry, the BASE is opposite to ACID, which helps to remember the acronym. According to the same resource, these are the rough definitions of each term in BASE:

(B)asically (A)vailable: basic reading and writing operations are available as much as possible (using all nodes of a database cluster), but without any consistency guarantees (the write may not persist after conflicts are reconciled, the read may not get the latest write).
(**S)oft state**: without consistency guarantees, after some time, we only have some probability of knowing the state since it may not yet have converged.
(E)ventually consistent: If the system is functioning and we wait long enough after any given set of inputs, we will eventually be able to know what the state of the database is, and so any further reads will be consistent with our expectations.

Q4. Which two CAP characteristics does the following NoSQL database support?

- Cassandra
- HBase
- MongoDB

ANSWER:

Cassandra focuses on Availability and Partition tolerance (AP).
HBase focuses on Consistency and Partition tolerance (CP).
MongoDB focuses on Consistency and Partition tolerance (CP).

Q5. Based on the CAP theorem, select the most suitable focus for your applications.

ANSWER:

Banking System (C & A)
Imagine a banking system that prioritizes consistency and availability. In this scenario, when clients request their account balance or process a transaction, they expect to receive the most up-to-date information. However, if a network partition occurs, the system cannot continue processing requests, potentially causing service disruption and data loss.

Social Media Platform (A & P)
On the other hand, a social media platform like Twitter may prioritize availability and partition tolerance. This ensures that the platform remains accessible even during network partitions. In this case, users may see slightly outdated content but can still interact with the platform without significant disruptions.

Search Engine (A & P)
A search engine must ensure that users can always get results for their queries, even if some are outdated or incomplete. This system would choose AP over C because availability is more important than the consistency.

Airline Seat Reservation System (C & P)
Overbooking is a costly mistake. A CP system ensures that booking is consistently reflected across the system when a seat is booked.

In reality, the distinction between AP and CP isn't always clear-cut. Systems can be tuned for different needs, and many databases offer configurable settings to lean more towards consistency or availability as needed. However, these examples provide a general sense of the trade-offs and decisions between AP and CP systems.

References

[1] Stonebraker, M., 2010. SQL databases v. NoSQL databases. *Communications of the ACM*, *53*(4), pp.10-11.
[2] Khan, H., 2014. NoSQL: A database for cloud computing. *International Journal of Computer Science and Network*, *3*(6), pp.498-501.
[3] Li, Y. and Manoharan, S., 2013, August. A performance comparison of SQL and NoSQL databases. In *Communications, computers and signal processing (PACRIM), 2013 IEEE pacific rim conference on* (pp. 15-19). IEEE
[4] ACID. https://en.wikipedia.org/wiki/ACID
[5] CAP theorem. https://en.wikipedia.org/wiki/CAP_theorem
[6] BASE properties. https://en.wikipedia.org/wiki/Eventual_consistency