SPECIAL TOPICS IN DATABASE
FINAL EXAM

ON

Cloud Database comparison for Problem statement

STUDENTS | Pragya Bhandari

ID | 16436198

INSTRUCTOR | Prof. Ramon Lawrence

IRVING K. BARBER SCHOOL OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
OKANAGAN CAMPUS

E-mail: pragya18@mail.ubc.ca

1 Problem Statement

Our company is creating the next top mobile game with the goal of supporting millions of users and making billions of dollars. Although the game may start off small, the system should scale to support millions of customers. Your goal is to design the database to support this mobile game.

Each user has a GameState that tracks their current status in the game. When the mobile game app is launched, the app immediately queries the database to retrieve this game state.

When the user interacts with the game, an event is generated. The event is sent from the app to the server. The server processes the event to make sure it is valid. If it is valid, it updates the game state and returns it to the app. The server stores the updated game state and the GameEvent in the database. Game events will not be changed once they are stored in the database.

The most important database operation is extremely fast read of the game state to send it to the player when requested. The database must also process game state updates and game event inserts quickly.

One common query is to show the top 10 players in a region. This should be near real-time but does not always have to reflect the most recent information.

For marketing to players, the system will periodically execute queries to determine players that have high activity (lots of events) in a given time. This information will be used to market bundles for players to purchase. This query does not have to be as fast as it does not effect the immediate game play for users.

2 Parameters

Upon reading the entire problem statement the following points can be considered to summarize the need of the company:

- *Scalability* : The future goals of the company are clear and they want to be prepared for the potential the game has on the market.
- *High availability*: It is required of the database to have high availability to provide the details of the game state and make necessary updates respective to the game events taking place when demanded.
- *Fast reads/writes*: The game state reads need to be speedy to avoid any lagging in information retrieval or updates.
- *Asynchronous*: Information can be asynchronous sometimes.
- *Complex queries*: Some complex queries need not be too fast.

3 Recommended Cloud Database

For the given problem statement I would recommend Redis NoSQL and Neo4J. Whereas I would consider Cassandra to be the worst choice. Finally I will mention some other notable databases that I rejected because of some concerns. I have considered the parameters mentioned above but in addition to them I have also assessed the databases on their flexibility with changing or evolving needs.

3.1 Redis

Redis is a scalable, in-memory database option which has various data structures to model the database on. In terms of scalability, Redis Enterprise allows you to scale your read operations by creating another database (on the same cluster) that will be served as a read-replica of your original database using the 'replica-of' feature. Your read-replica is treated as a completely different database, and can be configured with a different number of shards and different availability or durability characteristics.

Redis Enterprise's high availability is built around replication, but automatic failover, backup, and recovery also impact the ability to meet the application's high availability service level agreements. When it comes to performance, Redis Enterprise has been benchmarked to handle more than 200 million read/write operations per second, at sub-millisecond latencies with only a 40-node cluster on AWS. This makes Redis Enterprise the most resource-efficient NoSQL database in the market. Finally, Redis uses by default asynchronous replication, which being low latency and high performance, is the natural replication mode for the vast majority of Redis use cases.

However, the main reason I selected Redis for this problem was because of the client side caching technique it uses which I believe fits perfectly to the need of repetitive general queries and can give more efficient results. Another important thing that I considered was the fact the fact that Redis is NoSQL and it does not demand a schema. It gives the option to select from multiple data structures to implement the overall modelling of data from our problem statement. Given all the advantages, there is one concern with using Redis, is the volatility of data. The asynchronous replication and possibility of failures between data transactions make it possible to lose data, but I believe it is not crucial to lose certain data in the gaming point of view and that if it does not occur often, it will not cause hindrances to the users in any way whatsoever.

3.2 Neo4J

Neo4J is a graph based database that offers NoSQL. Neo4j's high-performance distributed cluster architecture scales with the data in real-world situations, minimizing cost and hardware while maximizing performance across connected datasets. With Neo4j, you can achieve unlimited horizontal scalability via sharding for mission-critical applications with a minutes-to-milliseconds performance advantage. Neo4J has causal consistency which means that the client applications are guaranteed to read their own writes. This, however, may not be suitable for a multi-player game setting such as the one the problem specifies. Neo4j provides options of indexing the data for fast reads/writes and has been claimed to be faster than MySQL on many complex data relation analysis.

A major advantage Neo4j might have over other databases is its graph data modelling. This can benefit our company in two ways - future developments and large scale graph analaysis. The game might expand into multiple parts with various kinds of user or game event data that might need to be persistent. This may not be visualized as of now, but is highly likely to happen due to the scalability expectations. This means that having a flexible NoSQL graph data structure will be advantageous in the future developments that the game will morph into. Secondly, the graph database will be extremely useful from the marketing perspective in proper analysis and targeting the users for in-game purchases or game event recommendations. This will enable an easier visualization as well as access of huge data which can all take place within the Neo4J stack and developer tools.

3.3 Cassandra

Let us assume that every other aspect of Cassandra is perfect for this problem statement and begin the discussion with that assumption. Suppose Cassandra has high availability, extremely fast read/writes and perfectly scalable with virtually no downtime. I am not saying that it is, but I want to argue on why it is not suitable for this problem statement with that supposition. Cassandra uses its own CQL which is Cassandra query language and can perform fast queries based on the indexing or partitioning that we do. CQL has concepts of partition key, primary key and clustering key which each have a purpose in defining the indices and later query formulations. Primary key will have to be a defining attribute which is used to query most data, partition key corresponds to which attribute we would need to group by and finally clustering key is the sorting order. A major disadvantage of CQL is that the three keys need to be balanced perfectly during the data modelling process itself and will be immutable later unless the company would like to create new tables each time their requirement changes.

If the developers recognize the keys to be attribute 1 and 2, but later are required to fetch data that corresponds to information on attribute 3,4 whether in ordering or grouping, they will have to rethink the entire data modelling. This directly implies least flexibility for future developments. Personally, I believe that gaming industry is an ever expanding in any multiplayer game scenario, there are new maps, characters, points and clan systems, and any experimental side-quests. Cassandra's query language however is extremely limited and will end up costing the developer more resources to compensate for its limitations of query options through code level calculations and data structure manipulations. Therefore, I would not recommend Cassandra for such a problem statement at all.

3.4 Notable mention

The most familiar and established kind of database RDS was in my thoughts during this analysis. Although it fits in most scenarios, I decided to leave it from my recommendations in this case because of the constant theme of my analysis which is future developments. All RDS are schema based which means there is rigid relation definition involved when starting the data modelling. Due to the unknown nature of game expansion (at least for me in this case), and the potential issues with foreign keys while editing relations, I would rather go with NoSQL options.