INTRODUCTION

As data grows large, retrieval and insertion become a concern. This is particularly common with relational databases, as large data volumes might need to be archived at specific intervals to reduce the size of the tables and make data insertion/retrieval easier. The concept of indexing is common with relational databases (which uses a binary tree structure) to aid faster retrieval of data, however, insertion takes longer as the database grows larger.

SOLUTION 1 - MongoDB

This problem has led to the more frequent use of NoSQL document-oriented (JSON style) databases when dealing with big data. Examples of which include MongoDB (in focus), Apache CouchDB, OrientDB and IBM Cloudant to mention a few. One key advantage of document-oriented databases is the capacity to handle humongous data and the object-related structure which allows programmers to easily access data as they would access objects in a Software program.

Considering the increasing size of the Twitter DB and YouTube comments table, one of the recommended solutions is a document-oriented database - MongoDB. This database is a NoSQL database with full indexing capabilities, replication and high availability which can handle the large amount of data in question.

MongoDB is also valuable when there is no predefined data structure and the data structure can change over time. This could be useful in the case where new data is being collected from these social media platforms and is required to be integrated with the current data store.

Being one of the most popular document oriented databases available, it is compatible with Windows, Linux, OS X and Solaris operating systems. Companies using MongoDB include Uber, Lyft, Comcast, Vivint(emphasized here as they collect up to 500m IOT data points per day on a geo-distributed MongoDB cluster) and 3700+ other companies.

The next paragraphs will cover key database features and how MongoDB performs with regards to these features, as well as the disadvantages of using MongoDB.

ACID (Atomicity, Consistency, Isolation and Durability)

For transactional databases, being ACID compliant is required to ensure validity of data in the database. Transactions are groups of read/write operations that need to succeed or fail together, such that operations are half-executed, but rather either executed completely or not.

MongoDB v4 supports ACID transactions and while this is required for coordinated systems and systems that require consistency across data (financial systems for example), all of the major relational databases comply with ACID principles (with MySQL only supporting atomicity out of the box).

Other advantages of using MongoDB include:

- A schema-less database which allows users to incorporate datasets from several sources (e.g. Twitter and YouTube) into a single, unified data store. A schema-less DB also helps to speed up queries, improving database performance.
- Scalability: MongoDB has the ability to scale out(horizontally) across many servers with data sharding or replication to ensure high availability. This allows data to be stored not just within a single server but also within a cluster.
- Dynamic query support: which allows users to find data using any criteria, almost as powerful as you would be using SQL queries.
- Internal memory(RAM) is used for storing a working set of data to ensure quicker retrieval of data.
- Built-in support of MapReduce and aggregation operations.
- Supports relationships similar to SQL databases, however it is not as enforced as in SQL databases.

Disadvantages of MongoDB include:

- Unlike SQL, MongoDB doesn’t support joins. It however has a JOIN equivalent $lookup operator.
- Compared to MySQL which has native performance analysis tools, MongoDB doesn’t have as many performance analysis/evaluation tools. However, this has changed over time as more tools are being developed.

- The structure of RDBMS makes it easy to ensure data security unlike in NoSQL databases like MongoDB where users have to rely on application rules and logic.

In summary, MongoDB is recommended when

- Integrating large amounts of diverse(big) data into a unified data store
- High availability of data, with automatic, fast and instant data recovery is required
- Delivering data in high-performance applications
- The team does not have a dedicated Database Administrator

SOLUTION 2 - Apache Cassandra

One of Cassandra's biggest strengths is being able to handle massive amounts of unstructured data. In cases where your database needs to rapidly scale with minimal increase of administrative work, Cassandra may be a good choice.

How big can it scale? Cassandra can handle the load of applications like YouTube where the platform’s users upload more than 500 hours of fresh video per minute, YouTube revealed at recent press events. That works out to 30,000 hours of new content per hour, and 720,000 hours of new content per day.

Who Uses These Databases?

Cassandra is in use at Activision, Apple, BazaarVoice, Best Buy, CERN, Constant Contact, Comcast, eBay, Fidelity, Github, Hulu, ING, Instagram, Intuit, Macy's™, Macquarie Bank, Microsoft, McDonalds, Netflix, New York Times, Outbrain, Pearson Education, Sky, Spotify, Uber, Walmart, and thousands of other companies that have large, active data sets. In fact, Cassandra is used by 40% of the Fortune 100.

What About Database Structure?

Apache Cassandra stores data in tables, with each table consisting of rows and columns. CQL (Cassandra Query Language) is used to query the data stored in tables. Apache Cassandra data model is based around and optimized for querying. Cassandra does not support relational data modeling intended for relational databases.

How Are Their Queries Different?

While cQL is similar to SQL as a query language, a key difference is how query results can be returned. In SQL, when tables with "too many" relationships are joined, the row data to the left of the join is duplicated for each joined row. cQL can return objects.

Selecting records from the customer table:
Cassandra: ‘SELECT * FROM customer;’
MySQL: ‘SELECT * FROM customer;’

Inserting records into the customer table:
Cassandra: ‘INSERT INTO customer (custid, branch, status) VALUES ('appl01', 'main', 'A');’
MySQL: ‘INSERT INTO customer (cust_id, branch, status) VALUES ('appl01', 'main', 'A');’

Updating records in the customer table:
Cassandra: ‘UPDATE customer SET branch = ‘main' WHERE custage > 2;’
MySQL: ‘UPDATE customer SET branch = 'main' WHERE custage > 2;’

What Types of Replication / Clustering Are Available?

Cassandra does replication out-of-the-box. You tell it the number of nodes it should copy your data to and it takes care of the rest of the process.

Cassandra allows for multiple masters where losing a single node still lets you write to the cluster. This can allow for better fault tolerance without the 10 to 40 second downtime required with MySQL.

MySQL supports master – slave replication and master – master replication (as of MySQL 5.7.6 and later). Multi Source replication allows you to replicate from several masters in parallel.

Key Comparisons

- Cassandra is a NoSQL, peer to peer architecture, MySQL is a RDBMS, Master / Slave.
- Cassandra is horizontally and vertically scalable, MySQL is vertically scalable, horizontal scaling possible through Master / Slave replication or sharding.
- Cassandra JOINS discouraged with the query-driven model. Fetches data from one table per query, while MySQL tables, JOINS required to query data.
- Cassandra Read Performance is 0(1), while MySQL requires reading from multiple tables using JOIN, resulting in 0(log(n)).
- Cassandra append model provides high writing performance, while MySQL requires a search first slows down write performance.
- In Cassandra, ACID properties are not provided, although it can be tuned to support ACID properties, while MySQL provides ACID transactions.

Pros

- Continuous availability: as a fully distributed database (no master nodes), we can update nodes with rolling restarts and accommodate minor outages without impacting our customer services.
- Linear scalability: for every unit of compute that you add, you get an equivalent unit of capacity. The same application can scale from a single developer's laptop to a web-scale service with billions of rows in a table.
- Amazing performance: if you design your data model correctly, bearing in mind the queries you need to answer, you can get answers in milliseconds.
- Time-series data: Cassandra excels at recording, processing, and retrieving time-series data. It's a simple matter to version everything and simply record what happens, rather than going back and editing things. Then, you can compute things from the recorded history.

Cons

- NoSQL requires thinking differently, and can be challenging for people with strong relational database backgrounds to understand. The CQL language helps with this, but it pays to understand how the engine

works under the hood. That said, the benefits outweigh the challenge of the learning curve
- Lacking the ability to relate data between sets makes querying harder, but this again is the nature of NoSQL.
- Data aggregation functions like SUM, MIN, MAX, AVG, and others are very costly even if possible. Hence Ad-hoc query or analysis is difficult.