STUDY OF MONGODB AND MYSQL IN E-COMMERCE ENVIRONMENT

#1.Ms. Charu Jain, Student of M.Tech (CSE),AITM Engg College, Palwal
#2. Mr. Mahesh Chauhan, Asstt. Prof., AITM Engg College, Palwal

Abstract:As business continue to grows, businesses continue to look for better ways to meet their evolving needs with web solutions that scale and perform adequately. Several online retailers have been able to address scaling challenges through the implementation of NoSQL databases. While architecturally different from their relational database counterparts, NoSQL databases typically achieve performance gains by relaxing one or more of the essential transaction processing attributes of atomicity, consistency, isolation, and durability (ACID).
As with any emerging technology, there are both critics and supporters of NoSQL databases. The detractors claim that NoSQL is not safe and is at a greater risk for data loss. On the other hand, its ardent defenders boast the performance gains achieved over their relational counterparts.
This thesis studies the NoSQL database known as "MongoDB," and discusses its ability to support the growing needs of e-commerce data processing. It then examines MongoDB's raw performance (compared to MySql, anopen source & light weight relational database)

Keyword:*Ecommerce, MongoDB, Mysql Key-value pair, column oriented, and document based etc.*

I Introduction:MongoDB is an open-source NoSQL-Database developed by 10gen in C++. NoSQL is a new trend in database development and refers generally to databases without fixed schema. Such databases usually have a lower transaction safety but are faster in accessing data and scale better than relational databases. MongoDB is one of the newer NoSQL databases developed in 2009. The database belongs to the category of the document-based databases.

NoSQL databases have proven to be a viable solution for some unique scaling scenarios. However, the idea of implementing a NoSQL solution (instead of a relational database) seems premature to many information technology organizations. Often, experienced professionals will decide to "live with" a process that is lengthy or slow due to an inability to scale appropriately.

1. What is MongoDB?

The MongoDB database consists of a set of databases in which each database contains multiple collections. Because MongoDB works with dynamic schemas, every collection can contain different types of objects. Every object – also called document – is represented as a JSON structure: a list of key-value pairs. The value can be of three types: a primitive value, an array of documents or again a list of key-value-pairs (document).

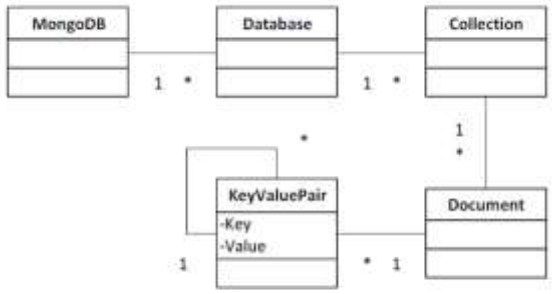


Figure 1: Model of the MongoDB system

To query these objects, the client can set filters on the collections expressed as a list of key-value pairs. It is even possible to query nested fields. The queries are also JSON structured; hence a complex query can take much more space than the same query for a relational database in SQL syntax. If the built-in queries are too limited, it is possible to send JavaScript logic to the server for much more complex queries. MongoDB requires using always the correct type: If you insert an integer value into a document, you have to query for it also with an integer value. Using its string representation does not yield the same result. 5

The database supports indexes It is possible to create ascending, descending, unique and geospatial indexes. These indexes are implemented as B-Tree indexes. The "_id" column which can be found in every root document is always indexed.

The following are some of MongoDB benefits and strengths:

(i)Speed: It gives good performance, as all the related data are in single document which eliminates the join operations.

(ii)Scalability: It reduces the workload by increasing the number of servers in your resource pool instead of relying on aindividual resource.

(iii)Manageable: It is easy to use for both developers and administrators.

(iv)Dynamic Schema: Its gives the flexibility to evolve data schema without modifying the existing data.

What Makes NoSQL Different?

NoSQL database technology is premised on the work done by Dr. Eric Brewer on distributed systems. In his 1999 paper (co-authored with Armando Fox) "Harvest, Yield, and Scalable Tolerant Systems", Brewer (Brewer, Fox, 1999) introduced what he called the "**CAP Principle**." Essentially, every distributed system strives to address three primary concerns:

(i)Consistency - All nodes view the same data at the same time. Data in the database remains consistent after the execution of an operation.
A system that has (Brewer, Fox 1999, p. 2) strong consistency" or "single-copy ACID consistency; by assumption a strongly-consistent system provides the ability to perform updates."

(ii)Availability - A guarantee that every request receives a response about whether it was successful or failed. High availability is achieved if a read request can always reach and return data from a replica.

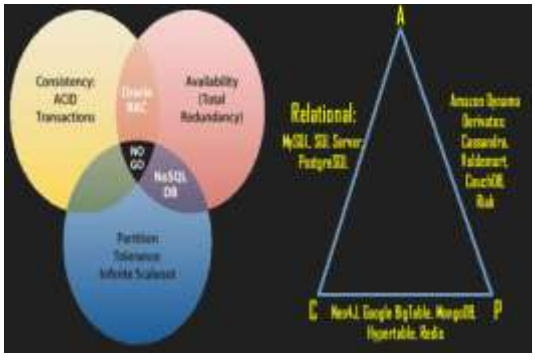
(iii)Partition Tolerance - The system continues to operate despite failure of part of the system. The servers may be partitioned into multiple groups that cannot communicate with every other group. The network can break into two or more parts, each with active systems that cannot influence other parts.

The key point of Brewer's CAP Principle (later known as both "Brewer's CAP Conjecture" and "Brewer's CAP Theorem") indicated that it was impossible for a distributed system to fully satisfy all of these conditions. MIT researchers Nancy Lynch and Seth Gilbert verified this in their 2002 paper, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services." They found that (Gilbert, Lynch, 2002, p. 11) "it is impossible to reliably provide atomic, consistent data when there are partitions in the network." They agreed with Brewer's conclusion, that it was possible to (Gilbert, Lynch, 2002, p.11) "achieve any two of the three properties: consistency, availability, and partition tolerance."

Given that a distributed system (e.g.: database) could be expected to satisfy any two of points of the CAP Principle, Brewer devised a list of characteristics hereafter referred to as "CAP Configurations." These can be used to identify the architectural strengths of various database systems:

NoSQL database follows different combinations of the CAP theorem [3]. Here is an detailed description of three such combinations:

CA - All nodes will remain in contact as a result of the single site cluster and any partition will block the system.(Brewer, Fox, 1999) "Databases that provide distributed transactional semantics can only do so in the absence of a network partition separating server peers." Relational database systems are considered to be "CA" databases, as they have the ability to provide strong ACID transactions. Partition tolerance is a secondary consideration, and istypically achieved through replication, clustering or fail-over strategies.



CP – In this, some data might not be accessible but consistency and accuracy are not compromised.

There is no need for distributed concurrency control as well.

NoSQL databases achieve ACID-like transactions to their partitions through locking strategies. These types of systems (such as MongoDB) sacrifice high-availability to ensure "strong" data consistency across all replicas/nodes, and (Brewer, Fox, 1999, p. 2) "avoid the risk of introducing merge conflicts (and thus inconsistency)."

AP - With the AP approach, the returned data might not be accurate but system will be available in spite of any partitioning. This is best suited for replication needs and fault tolerance.

This designation describes databases that are both highly available and partition tolerant. With consistency as a secondary consideration, most of these NoSQL databases (such as Cassandra, Riak, and CouchDB) operate under the assumption of optimistic or (Browne, 2009) eventual consistency. However, the performance benefits achieved by sacrificing consistency (Yu, Vahdat, 2000, pp. 1-2) "comes at the expense of an increasing probability that individual accesses will return inconsistent results, e.g., stale/dirty reads, or conflicting writes."

Essentially, NoSQL databases are architecturally different from relational databases because they are designed to reap the read and write performance benefits of partition tolerance (horizontal scaling), while leaving either consistency or availability up for negotiation. Knowing this difference is paramount to understanding the situations where a NoSQL database may be a better fit than a RDBMS.

Why is ACID Important?

One common way to describe relational database transactions is to say that they contain properties of atomicity, consistency, isolation, and durability. These transaction properties are collectively known as ACID. The ability to provide ACID transactions is one of the main reasons many businesses utilize RDBMSs. As author Joan Casteel illustrates in her 2007 book "My SQL," an example that makes a good case for ACID transactions (Casteel, 2007) is a banking transaction.

ACID TRANSACTION

(i)**Atomicity**: Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen.

(ii)**Consistency:**The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined rules.

(iii)Isolation: The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. Providing isolation is the main goal of concurrency control. Depending on concurrency control method (i.e. if it uses strict - as opposed to relaxed - serializability), the effects of an incomplete transaction might not even be visible to another transaction.

(iv) **Durability :**Durability means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.

Assume a bank transaction in which $500 is withdrawn from one account, split in half and $250 is deposited into two other accounts. If the bank's data center should suffer a power failure after the money is withdrawn, but before the other two transactions reach completion, is that $500 lost? If the transactions were completed on a database that supports ACID transactions, then (Casteel, 2007, p. 142) "the save will not occur until all three actions are entered."

It should be noted that (Banker, 2012, p. 256) "MongoDB doesn't provide ACID guarantees over a series of operations, and no equivalent of RDBMSs' BEGIN, COMMIT, and ROLLBACK semantics exists." However, MongoDB does support (Banker, 2012, p. 256) "atomic, durable updates on individual documents and consistent reads." The take-away from this: if you have a multi-operation transaction that decrements from one property and increments another (on a single document), then MongoDB can ensure atomicity and durability in this case.

On the other hand, MongoDB cannot ensure atomicity (or durability) on a multi-operation transaction that updates properties on separate or multiple documents.

With ACID transaction support defined, it is time to ask next logical question. "Why would anyone choose to not implement ACID transactions?" After all, the reasons to use ACID transactions seem to be grounded in data safety and common sense.

WHY MONGO IS CHOSEN OVER MYSQL DESPITE LACK OF ACIDITY

In some scenarios, e-commerce retailers may allow certain non-critical data to go inconsistent or "stale" with the majority of nodes, for short periods of time. Assume a scenario where a customer places an order on a website, and it makes its way through their ERP system. After a while, it may be sent back to the order history database (in batch), and made available for query by a web user. If the order history database nodes are updated every hour, then a partial or failed update is not critical to a business transaction. In this case, an ACID transaction is not required as there is a good chance (assuming that the issue which caused the original failure has been rectified) that the order history data will re-update successfully within the next hour.

The benefit to be gained (in the previous scenario) is that of raw performance. Large internet retailers may have hundreds of thousands (or even millions) of customers. A database containing order history records for prior years may grow big rather quickly. The absence of ACID transactions allows NoSQL databases to quickly serve this type of data across multiple nodes.

Other times, going without ACID guarantees is not so much a choice as it is a way to deal with system failure. In the case of Amazon, which has (Decandia et al, 2007, p. 205) "tens of thousands of servers" spread across the world; there is almost always some component in a state of failure. To counter that, Amazon's Dynamo NoSQL database (Decandia et al, 2007, p. 205) needed "to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance." In this case, as long as the data is consistent across a majority (or preconfigured quorum) of nodes, the transaction is considered to be successful.

ELEMENTS TO BE CONSIDERED IN THE SELECTION OF DATABASE

Performance and adherence to ACID are central concerns in choosing an e-commerce database. However, there are additional aspects that may influence the choice of database implementation. Some of the important aspects to consider are data security, delivered toolset, and backup/recovery tools

(i) **Security:** While not an ACID property, security is an aspect of database processing that is on the forefront of everyone's mind. Given how sensitive data may need to be stored in an e-commerce database, a discussion on security is warranted. Depending on the sensitivity of the data to be stored and the location (both physical and network) of the database server, available security options may make or break a decision to implement a specific database.

Security is something that Relational Database, Mysql does very well. In fact, assigning passwords to the initial system users is part of the installation process. It has a complex system where users are granted permissions to their Oracle schema and other specific database objects. Mysql also has roles that can be created and assigned permissions, then a role can be granted to a user to convey those permissions.
However, MongoDB does not provide the same granularity of security, and does not enable any type of security or authentication by default. MongoDB has security options that can be enabled, but users are recommended to (Banker, 2012, p.226) "take advantage of security features of modern operating systems to ensure the safety of their data." Essentially, this means MongoDB servers should use a local software firewall (typically provided by the operating system) to guard unused ports, and limit both physical and remote access only to verified users of the host system.

One important point to note is that data between MongoDB machines (replica set members, shards, etc…) is sent (Banker, 2012, p. 226) "in the clear." As of MongoDB 2.2, SSL encryption can be enabled (Merriman, 2012) with a slight degradation in performance. Therefore, running MongoDB in a secure (Chodorow, Dirolf, 2010) trusted environment behind a (hardware) firewall, is highly recommended.

To enable authentication in MongoDB, an administrative user must be created and (Banker, 2012) added to the "admin" database. Then the mongod processes can be started with the "--auth" option to require all connections to have an authenticated user. Authenticated users can then be added to each database, and may be designated as "read only" if required.

(ii) **Backup and Recovery:**For backup and recovery tools, there are few products capable of competing with Mysql Recovery Manager (RMAN). Introduced (Kuhn, 2010), RMAN offers a robust, feature-rich toolset that is included by default with both Oracle 11g R2 Standard Edition and Enterprise Edition. Some of the main features of RMAN include but are not limited to (Kuhn, 2010, p. 457):

- Management tools for the tracking, deletion and obsoleting of backup files.
- Incremental backups.
- Blocklevel recovery.
- Compression and Encryption.
- Validation and testing of backup files.
- Data conversion between multiple platforms.
- Recovery solution assistance from the Data Recovery Advisor.

One method of performing a MongoDB backup (Chodorow, Dirolf, 2010, p. 121) is to create "a copy of all files in the data directory." However, copying the data directory of a running instance of MongoDB will likely produce a corrupted backup containing errors. Of course, (Chodorow, Dirolf, 2010) the server can be shut-down before taking the backup copy, but this is certainly not preferred.

MongoDB also has the "mongodump" and "mongorestore" tools. For comparison, it helps to think of these tools as the rough equivalents of Oracle's Data Pump. While not as feature-rich as Oracle's RMAN, mongodump and mongorestore can be used (Membry, et-al, 2010) to backup and restore entire databases or even individual collections. The MongoDB backups can also be automated (Membry, et-al, 2010) through the use of shell and JSON scripts.

As an additional way to fulfill this need, 10gen announced in April of 2013 the addition of the MongoDB Backup Service. The MongoDB Backup Service is "a cloud-based solution geared toward its customers who use large data sets" (Backaitis, 2013, para. 1). This solution is billed as being fast,

convenient and cost-effective, while providing (among other features) security, high-availability and point-in-time recovery. While this service was announced too close to the conclusion of this project to be able to ascertain its effectiveness, it does show that 10gen is serious about providing its users with better options.

(iii) **Delivered Toolset:** As most NoSQL database packages tend to concentrate on the database product itself, there is typically little in the way of delivered toolsets. However, MongoDB does provide tools that allow users to perform complex Map/Reduce aggregations, in addition to geospatial indexes for location-based data. Other tools such as text searching and regular expression matching (which are not features in many other NoSQL databases) are also included with MongoDB.

On the other hand, Mysql comes with plenty of delivered tools. Features of MYSQL tools like SQL Yog include (but are not limited to) tools like (Nanda, 2008) Advanced Hybrid Columnar Compression, Automatic Storage Management (ASM), SQL Performance Analyzer, RMAN, and the Exadata Simulator. Additionally, Mysql delivers an extensive set of tools supporting business intelligence (BI) and analytics. Business analysts will find features like (Nanda, 2008) Cube-Organized Materialized Views, Partition Change Tracking (for materialized views), and the Analytic Workspace Manager to be quite useful. The Query Rewrite function also helps business users, as it checks for queries data that match data in existing materialized views, and utilizes that data without performing an operation on the actual table.

MongoDB does have security options available, even though they are not enabled by default. Data communications can be encrypted by SSL, and users can be required to authenticate on each database. However, a definite strength of Mysql is the granularity and level of security provided.
In dealing with a "big data" solution, whichever database you choose will require daily backups and periodic restores. The backup and recovery tools will need to be able to serve both the everyday and emergency needs of the DBA. As with security, the extra features provided with Mysql are certainly valuable options to have.
The delivered toolset that accompanies a database product is certainly not the prime reason for selecting it. However, having a good, pre-packaged toolset can certainly be an asset. Mysql has a very robust set of utilities which offer functionality to DBAs, developers, and business analysts. While MongoDB does not offer a similar quantity of features, it does provide more additional tools than many of its NoSQL competitors

When to use MongoDB

(i)**Schema-free**
Because MongoDB is schema, free every object can easily be serialized without changing the database. If you want to serialize an object, simply convert it to a JSON representation and store it in the database. No change in the schema is needed and therefore the development of the logic for persisting new domain objects can be done very fast.

(ii)**Data processing**
If you want to process a lot of data with complex queries, I recommend using a traditional optimized database. The performance results have shown that MongoDB is the slowest database in every query test. However a benchmark by Peter Bengtsson [25] shows that MongoDB is very fast for simple CRUD (create, read, update, delete) operations: If you generally use your database for simple operations and less complex queries MongoDB should be a good choice.

(iii)**Geospatial**
If you need a simple and lightweight solution with easy to understand queries MongoDB can be a solution for you.

VIII Conclusion: With the current emphasis on "Big Data", NoSQL databases have surged in popularity. These databases are claimed to perform better than SQLdatabases[4].SQL and NoSQL have been great inventions over time in order to keep data storage and retrieval optimized and smooth. Criticizing any one of them will not help the cause. If there is a buzz of NoSQL these days, it doesn't mean it is a silver bullet to all our needs. Both technologies are best in

what they do. It is up to a developer to make a better use of them depending on the situations and needs.NoSQL databases are becoming a major part of the database landscape today, and with their handful of advantages, they can be a real game changer in the enterprise arena. If our application doesn't fall into the category of the likes of Google, Yahoo, Facebook or

Wikipedia, we should reconsider our options for using NoSQL and stick with MySQL instead.

IX Reference:

[1] Clarence J M Tauroet.al."Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases", IJCA June 2012

[2] http://www.3pillarglobal.com/insights/just-say-yes-to-nosql

[3] Haihong, E. ; Guan Le ;Jian Du"Survey on NoSQL database", IEEE Conference, Oct 2011.

4**]**Yishan Li , Manoharan, S. "A performance comparison of SQL and NoSQL databases ", IEEE conference, Aug 2013.

[5]Radulescu, F. ; Agapin, L.I. et.al. "MongoDB vs Oracle -- Database Comparison " IEEE Conference, Sept 2012

[6]http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/

[7]https://www.digitalocean.com/community/tutorials/understanding-sql-and-nosql-databases-and-different-database-models

[8]http://www.networkworld.com/article/2226514/tech-debateshat-s-better-for-your-big-data-applica/tech-debates/what-s-better-for-your-big-data-application--sql-or-nosql-.html

[9]https://www.udemy.com/blog/nosql-vs-sql/

[10]http://blog.monitor.us/2013/05/cc-in-review-the-key-differences-between-mysql-and-nosql-dbs/

[11]http://sql-vs-nosql.blogspot.in/

[12] http://techledger.wordpress.com/2011/07/15/problems-of-rdbms/