A Comparison of SQL and NoSQL Database Management Systems (DBMS)

SWEN 432 - Essay

Zoltan Debre - 300360191

1. Schemas and flexibility

Schemas in RDBMS:

- We clearly declare the database schema for RDBMS. It is an abstract description of our real system, rigidly setup the structure and constraints of our database. We determine tables, relations among tables.
- We have to set up RDBMS schemas before adding data to the database. SQL database needs to know in advance what we would like to store. (In RDBMS writes are less efficient, so we usually try to minimize it, for this reason, we try to plan our database structure thoroughly and database usage in advance.)

Schemas in NoSQL:

- NoSQL databases does not require a predefined schema. The data structure can dynamically change.
- Data duplication and denormalization is totally acceptable in Cassandra. To get the most efficient reads we often have to duplicate our data. (In Cassandra, we do not have to worry about writes, writes are cheap and if we can speed up reading with extra writes, we should add more row to our database.)
- MongoDB has flexible schema. Instead of "tables", Mongo has "collections" which contain "documents". The structure of a document or a collection is not enforced.

SQL (RDBMS)	NoSQL
● rigidly defined schema ● fix schema structure before we start using the database ● relationships between tables are built in the schema, so it is advance determined also ● changing schema and relationship between tables need migration ● require normalized data ● no duplications	● support dynamic schema design ● increased flexibility ● denormalized data ● duplications ● ideal for for non-uniform data ● ideal for storing data which requiring frequent updates

When we would like to change our data structure in RDBMS, we have to write a migration tool, and update the whole database to the new structure. In NoSQL systems, we can just start inserting new documents and records in our Cassandra or MongoDB cluster, we do not have to change the structure of our previous collections.

In modern agile development culture the requirement of changing database structure is more common. In this case, migrating the database often is inefficient. A more flexible NoSQL strategy would be a better fit in a modern agile environment.

Running a unique query on RDBMS database, which was not considered in design phase, requires more work, and resources for our database system. We can compose an SQL query dynamically and satisfy the request. We can use materialized views, cache methods to speed up our process. Common practice is adding new indexes to our tables in order to optimize search, however this can be done by using migration process.

On the other hand, in NoSQL world, we can simply create new collections with new keys, and we can deliver the requested data rapidly. Adding new optimized collections to our MongoDB or Cassandra database is simple, and it requires minimum effort from our database administrator and from our system.

The SQL API is strict. If we would like to relax our ACID requirements, we cannot achieve it with SQL databases, because the SQL API does not allow. (Please find more about ACID in the next section.)

Glossary:

Schema: collection of database objects, the organization of data, a blueprint of how the database is constructed.

Relational database management system (RDBMS): a program where you can create, update and manage a relational database. It uses SQL language to manage data. (Popular products: Oracle, DB2, MS SQL Server, MySQL, PostgreSQL)

NoSQL database systems: ("non SQL", "non relational", not only SQL") Types: document databases (MongoDB), graph stores (Neo4J, Giraph), key-value stores (Riak, Redis), wide-column stores (Cassandra, HBase)

Sources:

- http://searchdatamanagement.techtarget.com/essentialguide/Guide-to-NoSQL-databases-How-they-can-help-users-meet-big-data-needs
- https://en.wikipedia.org/wiki/Database_schema
- https://www.mongodb.com/nosql-explained
- https://en.wikipedia.org/wiki/NoSQL

2. ACID compliancy

Relation databases guarantee ACID properties. This true for reading and writing data. NoSQL databases are not ACID compliant. This is a trade off that we pay for speed and 100% availability.

Atomicity in MongoDB: MongoDB never writes a document partially, it is always document-wide transaction. Writing document is atomic.

Atomicity in Cassandra: a write operation is atomic at the partition level, insertion or updates of two or more rows managed as one write operation in the same partition. Delete operation is also atomic at the partition level. Cassandra uses client-side timestamps to determine the most recent update to a column.

Consistency in MongoDB: primary Mongo server will get all the write request. Single server consistency is guaranteed. Secondary servers can guarantee only eventual consistency. The CAP theorem says that consistency and availability are incompatible, so if we allow reads from secondary server to increase availability, the strict consistency cannot achieved.

Consistency in Cassandra: not offer consistency in the ACID sense. Cassandra extends the concept of eventual consistency by offering tunable consistency for any given read or write operation, the client application decides how consistent the requested data should be. The most importants are "ALL", "QUORUM", "ANY". We can determine this level for each request.

Isolation in MongoDB: We can use "$isolated" operator to force isolation in single write operation, if it affects multiple documents, but this feature does not work with sharded clusters.

Isolation in Cassandra: write and delete operations are performed with full row-level isolation. A write to a row within a single partition on a single node is only visible to the client which performs the operation. But a batch operation is not isolated if data changes in more than one partition.

Durability in MongoDB: we can choose between different level of data durability with using "Write Concern". We can determine the level of acknowledgement of a request. Levels can be "no acknowledgement of the write operation", "write operation finished on a single mongo server", "write acknowledgement from the majority of nodes". We can also determine acknowledgement of a journal write.

Durability in Cassandra: writes are durable. Recorded in memory and in commit log on disk, so in case of server failure, Cassandra recovers any lost writes from commit log.

Glossary:

ACID (Atomicity, Consistency, Isolation, Durability) are properties of a safe database transaction.

- atomicity: "one part of the transaction fails, then the entire transaction fails"
- consistency: "any transaction will bring the database from one valid state to another"
- isolation: "concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially"
- durability: "once a transaction has been committed, it will remain"

CAP theory: Eric Brewer said that it is impossible to guarantee more than two out of the following three for distributed data stores: *consistency, availability, partition tolerance*.

Strict consistency is the strongest consistency model. Under this model, a write to a variable by any processor needs to be seen instantaneously by all processors.

Eventual Consistency is a weak consistency model in the system with the lack of simultaneous updates. It defines that if no update takes a very long time, all replicas eventually become consistent.

Sources:

- https://en.wikipedia.org/wiki/ACID
- https://en.wikipedia.org/wiki/CAP_theorem
- https://en.wikipedia.org/wiki/Consistency_model
- https://dzone.com/articles/how-acid-mongodb
- http://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlTransactionsDiffer.html
- https://docs.mongodb.com/manual/reference/write-concern/

3. Scalability

Vertical scaling: we increase the capacity of the actual hardware (more CPU, more memory)

Horizontal scaling: we add more machine to the cluster.

Good example for horizontal scaling is MongoDB and Cassandra. On the other side, RDBMS databases, like MySQL or PostgreSQL prefer vertical scaling.

We have to stop and restart the server if we scale vertically, however, we don't have to turn off our operation if we scale horizontally.

We can scale RDBMS horizontally, but it is more effort and we have to improve our application or implement a routing logic in our system. One of the common practices is sharding. In this case a routing service can help to find the appropriate shard.

Focus of NoSQL databases is scalability, all include some form of sharding or partitioning. The database can scale out across commodity hardware on-premises or in the cloud. Enabling unlimited growth, higher throughput and lower latency.

ACID-compliant databases (RDBMS) can scale read operations. We can organize our RDBMS system in a cluster with more nodes, but our capacity to scale is limited.

NoSQL databases follow the BASE model instead of the ACID model. We give up Atomicity, Consistency and/or Durability, so we can improve scalability. However we can use Cassandra or MongoDB features to opt in and choose ACID guarantees if we need.

Glossary:

BASE (basic availability, soft-state, eventually-consistency):

- basic availability: the database work most of the time
- soft-state: the write is not consistent
- eventual consistency: not immediately but later the content of replicas will be consistent

Database Sharding: a practice of using multiple servers of the same database. We can split a database based on the order of the data stored in the database and we can store the separated databases in different machines.

Sources:

- https://www.mongodb.com/nosql-explained
- https://webassets.mongodb.com/_com_assets/collateral/10gen_Top_5_NoSQL_Considerations.pdf
- https://stackoverflow.com/questions/11707879/difference-between-scaling-horizontally-and-vertically-for-databases
- https://softwareengineering.stackexchange.com/questions/194340/why-are-nosql-databases-more-scalable-than-sql
- https://neo4j.com/blog/acid-vs-base-consistency-models-explained/

4. Availability

NoSQL databases provide replication, multi-site architecture. They have cloud infrastructure support out of the box, so we can implement a robust fault-tolerant system without using any third party tool or service.

Traditional SQL databases can achieve high availability and load balancing, but they need an external service to support. (Companies, like Oracle, Microsoft provide a broad toolset with enterprise support.)

Solutions for SQL databases:

- failover clustering: a failover cluster is a combination of one or more nodes, or servers, with two or more shared disks.
- database mirroring: a software solution which mirror databases. Can be synchronous in high-safety mode or asynchronous in high-performance mode.
- log shipping: warm and hot standby servers can be kept up-to-date by reading a stream of write-ahead log records.

- replication: uses publish-subscribe model where additional database servers get updates from the main server.
- scalable shared databases: scaling out read-only databases, usually for reporting purposes.

Cassandra:

- right choice if we need scalability and high availability without compromising performance
- fault-tolerance on commodity hardware or in cloud infrastructure
- supports replication across multiple datacenters

MongoDB:

- high availability, scaling from single server deployments to large, multi-site architecture
- native replication and automated failover support enterprise-grade reliability

Glossary:

Database Replication: a practice of deploying multiple servers which are clones of each other.

Sources:

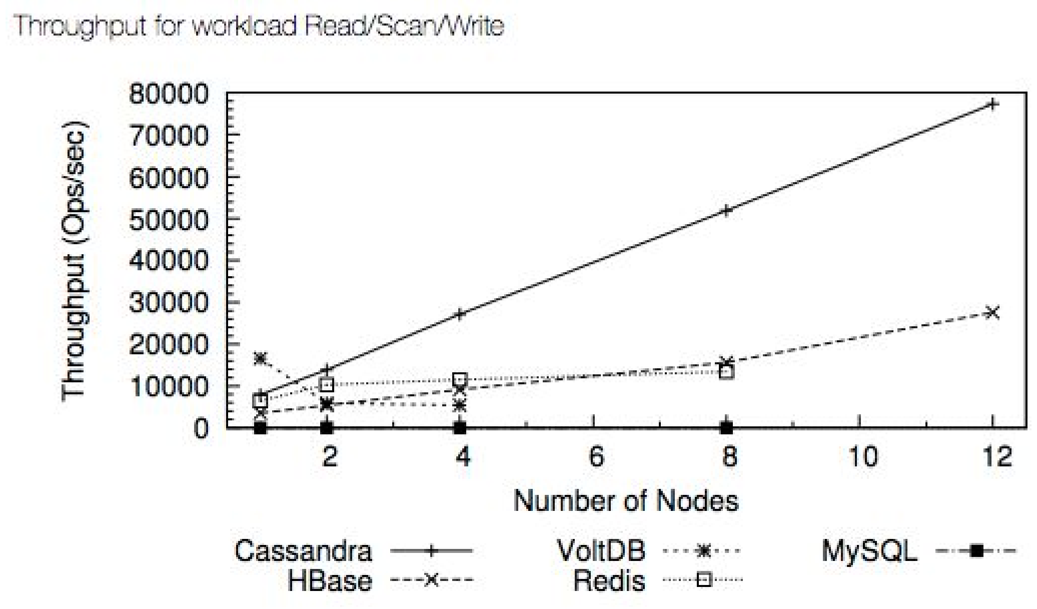
- https://msdn.microsoft.com/en-us/library/ms190202(v=sql.105).aspx
- http://bigdata-madesimple.com/a-deep-dive-into-nosql-a-complete-list-of-nosql-databases/
- https://www.postgresql.org/docs/9.5/static/different-replication-solutions.html

5. Throughput

Each solution has pros and cons as already explained in our previous paragraphs. NoSQL databases are faster in write and read than traditional databases, however the data structure is different and we give up ACID properties.

We can find more benchmarks on Internet, most of them show that NoSQL databases are more scalable and can manage higher throughput. We can increase our server and capacity with adding new nodes and clusters to our infrastructure. These clusters can be geographically distributed and separated, so the performance can be high independently of the user location.

The following benchmark is from a research of University of Toronto. It is one of the most popular comparison.



Sources:

- https://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf
- https://static.epcc.ed.ac.uk/dissertations/hpc-msc/2012-2013/RDBMS%20vs%20NoSQL%20-%20Performance%20and%20Scaling%20Comparison.pdf
- https://academy.datastax.com/planet-cassandra/nosql-performance-benchmarks
- http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf

6. Security

One of the most important assets of a modern company is its stored data. Therefore it is extremely important to keep our database secure.

When we set up or manage a database system, either RDBMS or NoSQL, controlling access is a crucial task. The following aspects should be considered: firewall, authentication, authorization, auditing, encryption, data masking.

Firewall: firewalls prevent all access to the database, we can control that from which computer and IP address has access to a database instance.

Authentication: it is about how can we prove our identity when we connect to the database. It is mainly managed by with username and password.

Authorization: it refers to what a user can do with a database, managing permissions. Best practice, we should grant users the least privileges necessary.

Auditing: tracking database activities, recording database events to an audit log.

Encryption: is the process of obfuscating data by the use of a key or password. The data is useless without the decryption key or password.

Data masking: this limits sensitive data exposure by masking it to non-privileged users. Data masking mechanism obfuscating the sensitive data sets, however the data in the database is not changed.

Traditional database systems are famous about their security. MS SQL Server, Oracle, DB2 provide more security layer out of the box. On the other side NoSQL databases reputation is not brilliant in this area. Week security, unsafe default settings, just to name a few.

Originally the following problems are existed with NoSQL databases:

- authentication, administrative user was not enabled by default
- weak password storage
- client-server communication was not secure
- was not able to use external encryption tools like LDAP or Kerberos
- no data encryption features
- sql injection vulnerability
- no protection against denial of service attacks.

Using cloud based database services, we have to trust in our service provider. We assume that they keep our data safe. Using external cloud based services is not always acceptable. For example, in government regulated industries, for example insurance and banking, have to store customer related data in dedicated servers which managed by the company instead of a third party provider, so cloud based solutions are not ideal.

If we can store our data in the cloud, we have to store the data in fully encrypted format. Communications protocol has to be secure (https, ssh).

Sources:

- https://www.mongodb.com/scale/nosql-database-security
- https://docs.microsoft.com/en-us/azure/sql-database/sql-database-security-overview

- http://blogs.perficient.com/dataanalytics/2015/06/22/nosql-nosecuity-security-issues-with-nosql-database/

7. Typical applications

- OLTP, where ACID properties, strict consistency, and strict security are of a primary consideration (like in banking, insurance, sales, and manufacturing),
- OLAP, featuring use of huge volumes of historical data, to run complex aggregate queries, and hence producing mostly non personalized answers supporting: business planning, and decision support.
- Big Data, found in: Internet search, Meteorology, Genomics, and Biology.
- Time Series Data
- Social Networks data.

	OLTP (for example a bank like ANZ or BNZ, managing bank transactions)	**OLAP (BI systems in a company, or tracking website visitors and user behaviour)**	**Big Data (for example GeoNet collecting seismic data)**	**Time Series Data (any audit or log system, for example collecting road usage traffic data)**	**Social Networks data (for example Twitter, Facebook for posting and showing messages)**
Updatability	Critical. Creating, updating transactions. Have to be real time and atomic with a strict confirmation and logging.	Acceptable, but not so important, critical feature. Usually we update our database from a journaled, logged data once or a few times a day.	Scalability is important, because huge amount of data can arrive in short peak period. Writing data is important in that moment.	Fast writing is the most important. Common practice to write the log in a text file and process later, after midnight.	Creating new entries are important, but update almost negligible.
Consistency	Critical, ACID guaranteed transactions required. Have to be strictly consistent.	Eventually consistency enough.	Eventually consistency accepted. Usually data consumption is timely separated.	Not important. Geographically separated servers can be synchronized later.	Not important, latency acceptable.
Security	Critical. Regulated industry cannot store data in the cloud. Accessibility and authentication are strictly journaled. Data has to be encrypted.	The prefered solution is to keep these system behind the company firewall, so we can limit accessibility. In this case the protection guaranteed by the company IT infrastructure than the database itself.	Most of the research data is publicly accessible, encryption is not a requirement.	Access is very limited, only database administrator or in-house analytical software should access these data.	Need only limited authorization management, because only the message creator can delete or update a social network post.
Availability	Critical. Security, consistency more important than 100% availability. During migration, database update, or horizontal server capacity upgrade, the service is not available. Usually after midnight for a few hours.	Important, especially in business hours, when users need access to these systems. However, we can clearly determine maintenance time outside of business hours, when the access is limited.	Number of users are limited and the response time is not critical. This type of information usually accessible by a web based API.	Have to be available for writing. It is cheap to redirect or cache writing parallel when the main log service is not available.	Geographically distributed replications. Definitely a good candidate for cloud based solution.
Throughput	Important, but if the service get high load, the safety and consistency get priority, so the response time can increase.	Our users run massive big queries on these system, so they have to support aggregation, analysis in huge scale.	Response time is not critical, can be limited and slower response is acceptable for huge queries.	We have to be ready for fast writing. Usually appending plain text is the best solution.	High performance read capability is a must, especially if the service is popular.
Other database features.	Optimized for read, write, update and delete.	Mostly optimized for read.	Saving and writing data is important in peak period, however, it can be managed by a journal or a log, so we can update and structure our data later.	Data format can be various. Using NoSQL databases for managing this type of data requires less effort on "extract", "transform" and "load" (ETL) processes.	Social networks store incredible amount of data. They archive older informations that are kept on a slower cheaper infrastructure.
Suggested Databases	Typical RDBMS users. Suggested databases: Oracle, MS SQL and industry specific mainframe solutions, like IBM DataPower.	Could be a mix of RDBMS and NoSQL solutions. Cassandra, Elasticsearch. PostgreSQL, MongoDB	Cassandra is ideal for managing this type of information.	Combination of Amazon S3 static file store with Apache Kafka is a great solution.	Most of the social network uses a combination of RDBMS and NoSQL solutions. PostgreSQL, Cassandra, MongodDB are popular. choices.

Glossary:

OLAP (online analytical processing): part of business intelligence, it is an approach to reply for multi-dimensional analytical queries. (for example: report writing or data mining applications)

OLTP (online transaction processing): information systems which manage transaction-oriented applications, typically for data entry and retrieval transaction processing on a DBMS. (for example a bank ATM machine)

ORM (Object Relational Mapping): programming technique to convert data between the limited types of database models and schema to the object-oriented programming language strictly typed object.

ODM (Object Document Mapping): a software package or library which map our object oriented programming model to our document store database structure. It is used for example managing MongoDB databases.

Sources:

- https://en.wikipedia.org/wiki/Online_transaction_processing
- https://en.wikipedia.org/wiki/Online_analytical_processing
- https://en.wikipedia.org/wiki/Object-relational_mapping
- http://www.dummies.com/programming/big-data/10-advantages-of-nosql-over-rdbms/

Appendix 1. Cassandra vs MongoDB

I found a useful comparison about Cassandra and MongoDB. A few facts from this essay:

	Cassandra	MongoDB
Users:	AppScale, Constant Contact, Digg, Facebook, IBM, Instagram, Spotify, Netflix, Reddit	Citrix, Foursquare, HootSuite, Klout, SurveyMonkey, Sony, Twitter
Database Structure:	- able to handle large amounts of unstructured data - uses wide column stores, but allows the name and format of columns to change - tables can be created, altered and dropped while the database is running and processing queries	- JSON-like documents - schema free - documents are like records in RDBMS, but we can easily modify with adding, deleting fields
Indexes:	we can only query using the primary key	indexes are important and prefered, without index every document will be searched
Replication, clustering:	- replication out of the box - allows multiple masters, better fault tolerance	- built in replication with auto-election - has single master - 10 to 40 second downtime for selecting a new master for writing when the original master is down

Development	- first release: July 2008 - developed by Facebook, it is managed by Apache Software Foundation nowadays.	- first release: 2009 - developed by 10gen, their name is changed to MongoDB Inc.
Use cases	real-time analytics, ecommerce, fraud detection, music catalogs, IoT, online courses, streaming data	real-time analytics, ecommerce, content management systems, mobile, IoT

Source:

- http://blog.panoply.io/cassandra-vs-mongodb

Appendix 2. DB Engines Ranking List in Jun 2017

Rank	**DBMS**	**Database Model**
1.	Oracle	Relational DBMS
2.	MySQL	Relational DBMS
3.	Microsoft SQL Server	Relational DBMS
4.	PostgreSQL	Relational DBMS
5.	MongoDB	Document store
6.	DB2	Relational DBMS
7.	Microsoft Access	Relational DBMS
8.	Cassandra	Wide column store
9.	Redis	Key-value store
10.	SQLite	Relational DBMS
11.	Elasticsearch	Search
12.	Teradata	Relational DBMS

Source:

- https://db-engines.com/en/ranking