San José State
UNIVERSITY

Project on

NoSQL

under the guidance of

Professor Rakesh Ranjan

Submitted on

05/09/2011

Submitted by

Ishaan Sutaria <ishaansutaria@gmail.com>

Gina Kang< ginak1004@gmail.com >

Batch: Monday (6:00 pm - 8:45 pm)

Table of Contents:

1. Introduction

In the computing system, there are enormous data that comes out every day from the web. A widespread system to handle those data was relational database management systems, RDBMS, but with changing of trends to various social networks, there came out another system to handle the data that changes very rapidly. All the database store system is applied by CAP theory which stands for Consistency, Availability, and Partition Tolerance. Consistency means "All clients always have the same view of the data" and Availability means "Each client can always read and write" and Partition Tolerance means "The system works well despite physical network partitions." For all of the database storage, they could take only two of those characteristics. Existing RDBMS takes Consistency and Availability but it can't be applied to Partition Tolerance. So the NoSQL takes Partition Tolerance with giving up either Consistency or Availability. To make it work, it gave up relational database and takes only things that it needs for each functions.

2. CAP Theorem:

Eric Brewer at the AMC symposium gave a lecture on principles of distributed computing proposed the CAP theorem in July 2000.

According to the CAP theorem proposed by Brewer also known as Brewer's theorem no distributed system can guarantee the following all three properties simultaneously,

- Consistency means all the nodes in a system should contain the same data.
- Availability means that the data should be available from the remaining nodes even if some nodes in a system fail.
- Partition Tolerance means that he can even if there is arbitrary message loss the systems continue to operate.

Brewer's proposed that out of these three only two of them can be guaranteed by a system. This was proved formally by Seth Gilbert and Nancy Lynch of MIT in 2002.

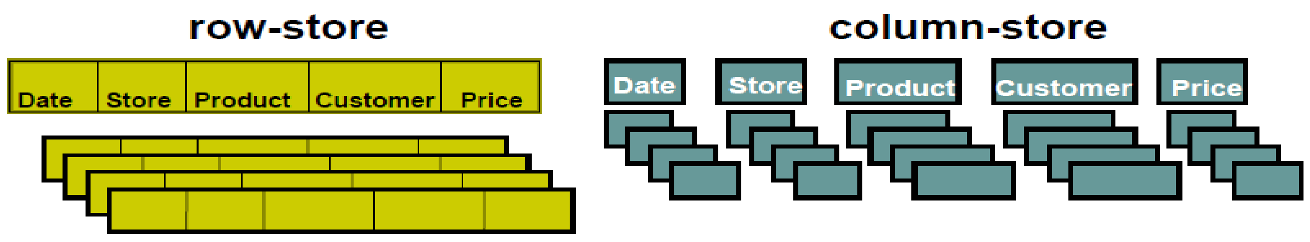
"Each node in a system should be able to make decisions purely based on local state. If you need to do something under high load with failures occurring and you need to reach agreement, you're lost. If you're concerned about scalability, any algorithm that forces you to run agreement will eventually become your bottleneck. Take that as a given."
—Werner Vogels, Amazon CTO and Vice President

3. Categories of NoSQL:

3.1 Wide Column Database:

Column-oriented database systems (column-stores) have attracted a lot of attention in the past few years. Column-stores, in a nutshell, store each database table column separately, with attribute values belonging to the same column stored contiguously, compressed, and densely packed, as opposed to traditional database systems that store entire records (rows) one after the other. Reading a subset of a table's columns becomes faster, at the potential expense of excessive disk-head seeking from column to column for scattered reads or updates. One of the most-often cited advantages of column-stores is data compression. The intuitive argument for why column-stores compress data well is that compression algorithms perform better on data with low information entropy (high data value locality).

In wide column databases data is stored in columns instead of rows. The major advantages of row based database i.e. RDBMS, SQL etc is that they ensure consistency Isolation, Atomicity, Dependability i.e. ACID properties. One of the major advantages of using row based databases is that it is extremely easy to modify a record we modify a row in a table. One of the major disadvantages is that it offers poor scalability. The advantages of wide coloumn databases are that it is distributed and highly scalable. Wide column database also has disadvantages like it is slow as compared to row based databases when it comes to multi column writes. According to one of the research paper named Column stores vs. Row Stores by Abadi, Maden and Hacchem row schema averaged approximately six times slower than column based designs when it comes to time taken for querying the data. Also Row store with column schema averaged approximately 2.5 times slower. One of the most-often cited advantages of column-stores is data compression. The intuitive argument for why column-stores compress data well is that compression algorithms perform better on data with low information entropy (high data value locality).



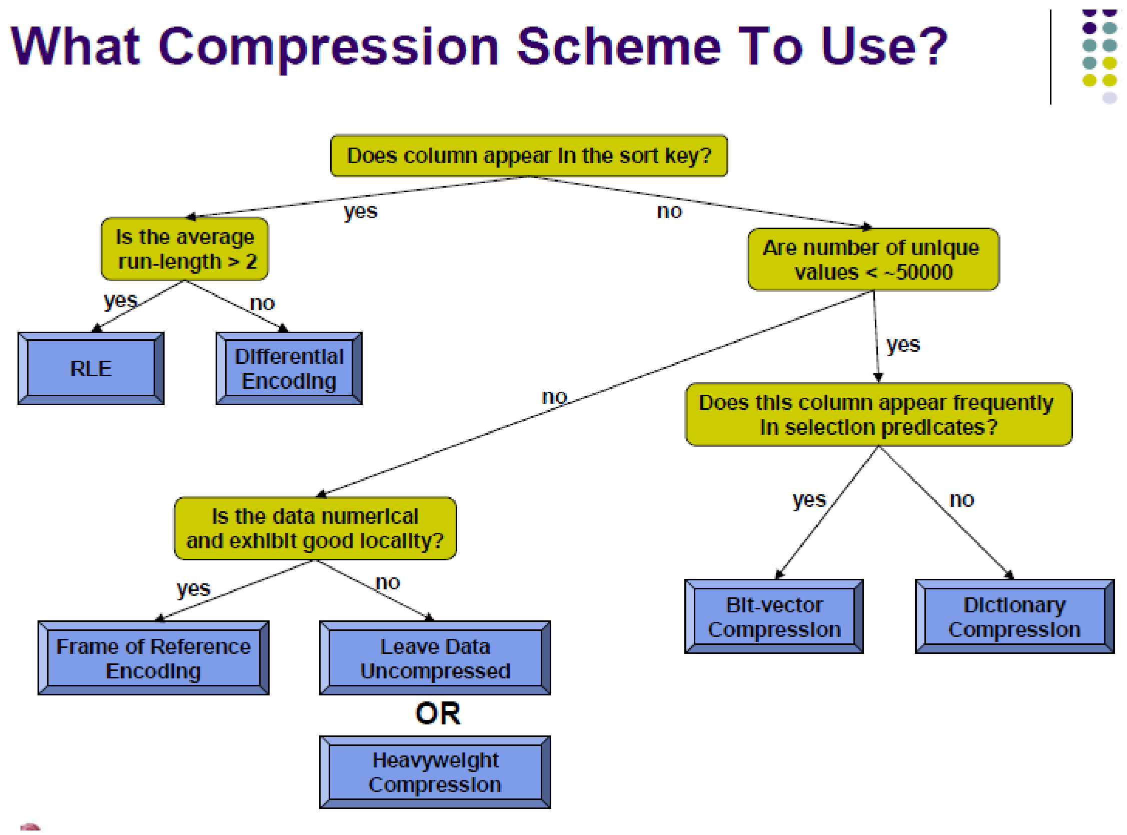
Types:

Some of the most important and famous wide column databases are as follows

- Haddop/ Hbase
- Cassandra
- Hypertable
- Clouddata
- Cloudera
- Amazon Simple DB

Compression:

In wide column databases there is an increased opportunity to store in column store. There is a tradeoff between I/O operations for CPU cycles. Here techniques like run length encoding are far more useful. The locality of data in column stores is much higher. It can even use the extra space made available through compression to store multiple copies of data in different orders


What Compression Scheme To Use?
Does column appear in the sort key?
yes
no
Is the average run-length > 2
yes
no
RLE
Differential Encoding
Are number of unique values < ~50000
no
yes
Does this column appear frequently in selection predicates?
yes
no
Bit-vector Compression
Dictionary Compression
Is the data numerical and exhibit good locality?
yes
no
Frame of Reference Encoding
Leave Data Uncompressed
OR
Heavyweight Compression

Applications:

There are many applications of column databases. Some traditional ones areas where they are useful are data warehousing which includes high end clustering and personal analytics, business intelligence and data mining e.g. proximity and information retrieval. Another promising application of column databases is scientific data management, where we need to showcase some results. There is also some recent work going on in addressing alternative data models such as object data models, array data models, XML and RDF using column stores. The practical applications where these are used are facebook, twitter, Netflix etc.

3.2 Document Based Database:

One of the first questions that come to people's minds specially those who have worked or have experience with relational databases is what document based storage is? It is basically an organization that provides flexible data creation where fields can be dynamically added and the ones which are not required can be constrained. Here the schema or design migrations are application and not data driven.
From the perspective of relational databases (Table based) here tables and columns define grouping, attributes and constraints. Schema migration required as data evolves and hence it is data driven and not application driven.

Document storage NoSQl implementations are a cross between a KVP(Key Value Pair) and record structure paradigms. There are many enhancements to the structure that can be done like dynamic restructuring of the schema and representation of non structured data. It is even possible to enhance the KVP by grouping of KVP to form records that gives a structure to KVP lack of organization. Another enhancement that can be done is to not enforce any schema at the database level. It should be followed at the document level which is one of the strengths of document storage model. Here the important thing to understand is that the application must provide the validation. Another major advantage of document based storage is that it provides on the fly structural changes and has tolerance for variations in organization and attributes.

Normalization plays a big role in RDBMS and plays a big role in providing many of the benefits like consistency, Isolation etc. Document based database in particular MongoDB supports application enforced normalization through design. But even denormalization has its own advantages.

Types:

There are many types of databases which comes under this category. They are as follows

- Couch Db
- Mongo Db
- Terrastore
- Through db
- OrientDb
- Raven Db

We have taken an example MongoDB to explain things in much detail and which will make things lot more clear and better your understanding of document based storage.

MongoDB

In MongoDB databases are independent storage areas in a mongo server, providing separate security, configuration and a database which consists of multiple applications. Here collections are sets of documents where the maximum amount of records that can be stored is 24,000. These records are indexable and are shared and replicated. Here if you want you can keep the collection can be of fixed size. They are performance oriented for caching and maintain insertion order. Documents are the records or the data and are have a client server communication which uses BSON and the maximum size of the document is 16Mb and many types are supported.

BSON is binary encoded JSON and supports JSON like data with certain enhancements for date, byte array and binary data. There are certain tradeoffs when using BSON like encode and decode speed advantages and embedded string length and bound string queries.

Data Representation:

Here data is a document and we call it a record. It is basically a set of key value data.

For e.g. “number”: [“123”, “45”, “345”]

Here a fields value can even consists of an array of data and elements can be single values or data objects. There is also surrogate keys available which are managed by the application layer ad are generated by Mongo.

Identifying documents in Mongo is by checking the _ifd field as all documents are stored in Mongo has an _id field and it is assigned by the application and the default assignment by mongo is a 12byte id which progressively increases as documents are added.

Querying in MongoDB:

Queries are expressed as JSON (BSON) and are organized very similarly to SQL statements.

Notation: Basic Key/Field : Value/Range Queries

example: db.[collection].find({"number" : "123"})

3.3 Key-value store

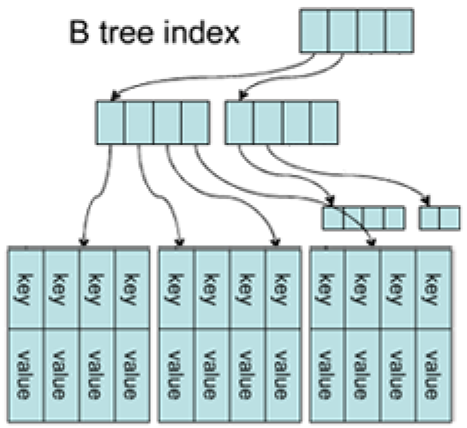
Key-value store is one of the methods that allow the application to store its data without schema. Generally it stores string which consists of key and the actual data so it doesn't need any formatted data model. Arbitrary data could be stored with any order with indexing single key which allows retrieval. So the advantage for that is fast storing but the retrieving speed should be considered to be improved. Another advantage is when everything is emerged according to the key values; the result code looks very clean and simple.

Types:

- Azure table Storage
- MEMBASE
- Riak
- Redis
- Chordless
- GenieDB
- Scalaris
- Tokyo Cabinet
- G.T.M
- Berkeley DB

Tokyo Cabinet

The Tokyo Cabinet is one of the databases using the key value store developed by Mixi Inc in Japan. It is programmed in C using hash table, B+tree, or fixed-length array and provided as application program interface of C, Pearl, Ruby, Java, and Lua. They showed the biggest advantage, speed of store, that when they stores 1 million records, it takes only 0.7 seconds for the regular hash table and 1.6 seconds in the B-Tree engine. Also in case of that it is not enough, it supports for Lempel-Ziv or BWT compression algorithms which reduces the size of database up to 25%. Tokyo cabinet is successive version of GDBM and QDBM improving some points. It improves space efficiency, time efficiency, parallelism, usability, robustness and it supports 64-bit architecture.



Hash database engine has one key and one value without any duplicates and it is crazy fast. B-tree database works same with hash database functionally, except using the underlying structure which makes the keys could be ordered by user so the user could do prefix , range matching on a key and traverse entries in order. The structure of the B-tree index is shown in the figure 1. It consists of bunch of leaves which is the location to store the data. The top point which starts the tree is root and there are bunch of nodes called children under the root and number of node under each

Figure 1: B-tree index

node is order and each level is called depth. Mostly, I/O is very expensive to do in a computer system and B-tree index helps to store data in efficient way. Whenever the I/O process, it only takes time to move to a lower depth in the tree using the indexes of the tree. So depends on the number of depth it could help to reduce the time to retrieve the data.

Another database engine is fixed-length engine which is just simple array. It doesn't have hashing so it only access through the natural number keys so it can't be any faster. At last, very strong engine, Table engine is also one of the methods that Tokyo cabinet used. Its structure is very similar to the relational database, but doesn't need to have predefined schema. User can declare arbitrary indexes on their columns and also they can queries the data.

Also Tokyo Cabinet has a package of network interface, called Tokyo Tyrant written in the C language. Tokyo cabinet has high performance but it couldn't cover the case when multiple processes share the same database, but Tokyo Tyrant does. Tokyo Tyrant supplemented Tokyo Cabinet for the concurrent and remote connections.

They also provide Ruby friendly version, Rufus-Tokyo, which interacting with Tokyo Cabinet and Tokyo Tyrant. Comparing two version, Tokyo cabinet and Rufus-Tokyo, Tokyo-cabinet is much faster. Tokyo cabinet hash takes 7.19s when Rufus hash takes 18.38s and Tokyo cabinet B-tree takes 11.92s when Rufus B-tree takes 21.93s.

BerkeleyDB

Berkeley DB is one of the big computer software libraries that is embedded database using key and value data. It is developed from Berkeley to manage the database in Berkeley and later, to manage more formally, SleepyCat took over it to develop and distribute in 1996. In 2006, it is merged to Oracle. This DB is used widely because it is easy to use and the performance is also good and it can be realized easily without bug. It is written in C and supports C, C#, C++, PHP, Java, Perl, Python, Ruby, Tcl, and Smalltalk APIs. It uses various methods like, extended linear hashing, B+tree, queue, fixed and variable length record access method. It doesn't have the concept of query language like SQL or table, but it supports locking, transactions, logging and Join, replication and in-memory databases.

BerkeleyDB uses main 4 access methods, B+tree, Hash, Queue and Recno. B+tree and hash have been explained above and both of them could have multi data key. Queue is storing the data in the queue that is fixed-length of record. This was developed to insert data at the end quickly. So this is very useful for the big application that its data should be updated simultaneously. The logical record is used as a key. Last method, Recno stores data in the variable-length record and the logical record is used as a key like queue. It retrieves and updates the data by record number. It can also load a text file into a database and it reads each line treating records. So like text editor, it could search very fast by line number. Recno is developed on top of the B+tree access method and stores the sequentially-ordered data values. So the programmer's point of view, they number the data from one sequentially and later, whenever the data is added or deleted, if updating data is lower number records, then it automatically renumbers. It supplemented queue's faults, but because it is using variable-length record, it also has lots of loads.

When not much data needed to be stored so it is loaded in the memory, there is no big difference of performance between B+tree and Hash. But when there is huge data that need to be loaded in the disk, the performance could have big difference between those two. Most of the case, B+tree work ok, but if the data set is huge, then we need to consider the Hash too. Also when the system uses parallel operation many times, then the queue or Recno is more proper.

Still, BerkeleyDB has some disadvantages on some functions. Using the Mutexes, it has disadvantages that in care of lock intensive, it could hold a lot of instructions blocking the parallelism. Also there is bunch of advantages using only key/data pairs, but at the same time there is obvious disadvantages for no locking, transactions, or replication which is that no concurrent access by readers or writers.

3.4 Graph database

Lastly, most complicated database of NoSQL is graph database This model consists of labeled information and the direction about the information. The object-oriented programmer could think nodes as objects to understand which represents entities. The properties are related information to the nodes. Nodes are connecting lines between nodes to nodes or nodes to properties and it stores information about the relationship between those two. The graph database is the one that uses smallest size when they stores data but also it is most complicated databases. The graph theory could represent the relationship between various domains very usefully and reliably. Many algorithms like shortest distance search, Pagerank, HITS and eigenvector centrality were used for the applications.

Types

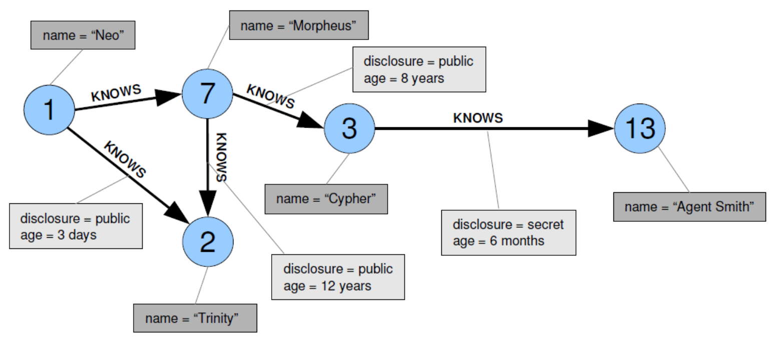
- Neo4j
- Infinite Graph
- Sones
- InfoGrid
- Trinity
- Hyper Graph DB
- Big Data
- DEX
- VertexDB

Neo4j:

Most famous database engine of graph database is Neo4j which is open-source and commercial engine implemented in Java developed by Neo Technology. It has bindings to Python, Ruby, Clojure, Scala, Java object mapping, Groovy and PHP. Also those bindings come with a REST extension. As it says above, it uses store structure which stores data in graphs rather than in tables. It is very useful for web development like tagging, metadata annotations and social networks. It is using disk-based which maximizes the performance and scalability. Neo4j's scalability is very massive as much as it could handle billion of each node, relationship and property for single machine.

Addition to the designing for network-oriented data, it also works to handle semi-structured data. As a name of it, semi-structured data has few requirements to be structured but rests of them are optional attributes. So sometimes the structure could be very dynamic which varies even between every single element. Generally data with a lot of variance could be hard to be represented in relational database schema but Neo makes it able to represent. Neo is robust database. It supports for JTA, JTS, 2PC and so on.

As a graph database, it consists of nodes, relationships and properties as shown in the Figure 2. To represent the network, it is similar to the APIs of object-oriented math libraries. Most of cases, the system has more information in the relationships rather than in the nodes themselves. The



Neo

Figure 2: Example of graph database capture the value of a network in the connections.

To traverse databases, Neo doesn't use declarative queries, but uses method used for navigational database which navigate through the nodes and relationships for queries. Instead of using API that could be used generally which is laborious and complex, it has powerful traversal framework. It is Java Iterator which traverses with four basic components; the starting nodes, the relationships, stop criteria and selection criteria.

There are several benefits on Neo model rather than the relational model. If the application has domain with naturally ordered data this is very good application to be used, because it is very useful to traverse over the domain data. Also whenever the semi-structured data is used, Neo knows it very easily. The system prefers individual structure avoiding centralization. So in the web now, semi-structured data is already increasing day by day. But the problem is that relational database almost impossible to handle this kind of model, but Neo could handle this mode very easily and efficiently. In business layer, schemas are evolving very quickly and the characteristics of Neo make it follow those variances which are very flexible and forgiving attitude towards properties. This helps to develop refactoring rapidly and iterate shortly for agile methodologies. So at last, the Neo has the advantages which is shorter development times, higher performance and lower maintenance.

But even though it has many advantages, it still has problems to be solved. First of all, Neo is still working on to standardize the model to be used very alternatively but still it is lack of alternative implementation. Also they are developing some embedded tools to support for Neo but still there are much more tools that could support relational databases better. Neo is the database engine that concentrates on the navigational model as semi-structure. Of course, they also support for structured data too, but still it can't handle arbitrary queries as much as relational database does. Neo has hardness to traverse along the relationships in the node space for some kind of queries. To cover little bit of those problems, they developed search engine integrating for arbitrary property lookups.

4. Disadvantages

The main disadvantage of NoSQL is that there is absence of SQL. As SQL databases have been present for such a long time and people are so used to it that transitioning would be a big problem. There is a big learning curve which can create a big hindrance in more people taking it up. NoSQL is more useful for large unstructured data but for applications where there is not very large quantities of data and it is more or less structured there is not much performance difference and it is better to use RDBMS rather than NoSQL databases.

5. Conclusion

With new idea, a tremendous system has been developed and it leads the database system for many areas. They don’t want to be called as a database anymore but like partition storage system for structured data management. NoSQL handles enormous data pouring out of web and that could be done using very cheap PC server. Also it could solve the problem of bottleneck phenomenon with avoiding some complex steps to manage the data. It aims to get only as much as they need. But even with these advantages, it has some drawback that they need to concern about. Depends on how they approach those problems, next generation would be changed.

6. References

http://stackoverflow.com/questions/1245338/what-nosql-means-can-someone-explain-it-to-me-in-simple-words - disadvantages

http://fallabs.com/tokyocabinet/ TokyoCabinet, TokyoTyrant

http://www.igvita.com/2009/02/13/tokyo-cabinet-beyond-key-value-store/ TokyoCabinet

http://www.usenix.org/event/usenix99/full_papers/olson/olson.pdf - BerkeleyDB

http://en.wikipedia.org/wiki/Graph_database - graph database

http://dist.neo4j.org/neo-technology-introduction.pdf - neo technology

http://blog.neo4j.org/2010/02/top-10-ways-to-get-to-know-neo4j.html - neo4j

http://cs-www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf - Wide Column Database

http://nosql.mypopescu.com/post/573604395/tutorial-getting-started-with-cassandra -Wide Column Database

http://www.mongodb.org/display/DOCS/Tutorial#Tutorial-LimitingtheResultSetvia{{limit%28%29}} – Document Database