Isai Mercado Oliveros

misaie

October 8, 2015

NoSQL Databases

NoSQL Databases started as a movement to improve relational databases' weaknesses. Non relational databases have existed since 1960, but they were not named "No SQL" until recently by the use of giant software companies such as Facebook, Google, etc. Moreover, Several NoSQL models have been developed to improve different aspects of storing information. The main types of NoSQL databases are key-value, document, graph, and column based. This paper will discuss a brief history, data model, storage system, failures/consistency, and scalability of BigTable, Neo4j, and CouchDB, which are column based, graph based, and document based databases respectively.

BigTable

In 2004 Google was facing problems to store and read giant amounts of information from the web, to solve the problem, a new type of database was developed. This new model needed to be able to store and retrieve data from thousands of little machines, and it needed to process millions of entries in a fraction of a second. The resulting database was called BigTable. BigTable's technology was only known to Google, until 2006 when the Google team led by Fay Chang, published "Bigtable: A Distributed Storage System for Structured Data". Based on the research, several open source projects emerged such as Hadoop, Cassandra, HBase, and Bluefish (Psaroudakis, 2012).

BigTable's data model is a map whose values are retrieved from a row key, a column key, and a time-stamp key. Row keys, and column keys are strings. Time-stamp keys are integers. Row keys are ordered alphabetically. Column keys are grouped by families. Column families store columns of the same type. Range of continuous rows are compressed into SSTables. SSTables store a sequence of 64KB blocks. SSTables also store at the end, an index-table to keep track of blocks' contents, and a bloom filter to check whether a row is contained inside the SSTable or not (Grigorik, 2008). SStables are grouped by tablets. Thus, tablets are also a range of continuous rows. Tablets make up a bigTable table, and they are served by tablet servers. Tablet servers are coordinated by a master server.

When a client makes a write request, it contacts the master server which contains a root-metadata-tablet with information to find data-tablets. The root-metadata-tablet maps the request to a second-level-metadata-tablet in another server. The second-level-metadata-tablet maps the request to the node that contains the data-tablet. Once the write request has reached the tablet server, the write request row is saved in a in-memory table before being saved into disk. When the in-memory table is full, its contents are saved as SSTables into disk. This process saves bigTable from writing to its disk too often (Gribble, 2013).

When a client makes a read request to the cluster, it contacts the master server which contains metadata to redirect the request to the corresponding tablet server. The tablet server looks for data in its in-memory table, and then it runs the bloom filter contained inside all its SSTables to assure if data for the requested keys exists inside the SSTable data or not. If the bloom filter of an SSTable returns positive, then the SSTable's index table is scanned to find the block containing the data. The data gathered from all the blocks inside the SSTables and the in-memory table is sent to the client (Malviya, 2010).

BigTable is very good at dealing with failures. The master server is backed up by four other replicas that may be chosen as a master server, if the current master server fails. In addition, the Google file system saves all data three times in different locations to prevent data loss. Thus, the master node constantly pings tablet severs to find out if they are still running. If any tablet server fails, the master server reassigns the failed tablet server’s data to another tablet server, and the cluster keeps running. Moreover, bigTable uploads as much information as it can into memory for fast access. If the server fails all that information would be lost, but bigTable deals with that by keeping logs of what has been loaded into its in-memory tables (Kohler, 2011).

In addition, bigTable is super scalable. It is programmed so that servers can be added while the cluster is running, and bigTable makes use of the new added capabilities as soon as it detects a new server. Google bigTable runs on Google’s computer cluster that contains thousands of small machines, and bigTable mobilizes data when individual servers die, or when new servers are added (Mehra, 2015).

Neo4j

Modern databases started development back in the 70’s. From that time until the 2000’s relational databases were predominant. They performed fine for several years when data was small or did not have so many relationships. However, as data started to grow and relationships increase, table to table operations took longer and longer (Armbruster, 2014). In 2000 Neo founders encountered performance problems with relational databases so they started developing a graph database prototype. By 2002 the first prototype of Neo4j was released. In 2009 Neo project raised funds from Sunstone, and Connor companies to continue development. In 2010 Neo4j version 1.0 was released. In 2011 more money was raised and headquarters moved to Silicon Valley.(“Neo4j – The World’s Leading Graph Database”, 2015)

Neo4j data model is a graph. Neo4j mainly stores data as nodes or edges; however, in order to make queries and node organization more consistent, it allows nodes and edges to have labels and attributes. For example, Node one contains a person object whose attributes are name : Pedro, age: 25, nationality: Mexican. Node two is another person, and Node three contains a Tweet object whose attributes are text: “feeling good!”, date: Oct 23, 2015. Node one has an edge with node two of type Friend with attributes firstMeeting: Jan 2, 1997, place: School. And finally Node two has a relationship to node three of type Tweet with no attributes. Extracting information from this kind of database uses normal graph algorithms with some filters.

In regards to micro architecture, Neo4j is optimized to help the graph to be as native to the hardware as possible. From the socket structure to the RAM, all hardware components are design to increase graph algorithms’ speed. On the other hand, macro architecture is composed by load balancers that redirect requests to the nodes in the cluster. Servers in the cluster are interconnected to every other server. One server is the master server, and all other servers are slave servers. If the master server fails, any other server can be called as the master and take over the responsibilities (Hoff, 2009).

In terms of design from the CAP theorem, Neo4j chose consistency and availability. Neoj4 is not very good at partitioning because when a client sends a write request, the load balancer only redirects write requests to the master server. The master server is the only one that can write to the cluster. This may be a potential bottle neck with big amounts of write requests, but it assures consistency on the data (Weber, 2015).

In addition, ideally Neo4j would like to have the entire graph into one huge memory for rapid access, but that is not possible for big sets of data, so the cluster gives portion of the graph to individual servers to store. When the client sends read requests, the load balancer redirects the request to the server that contains that part of the graph. Then the server processes the request, sends the data and loads the small portion of the graph into memory. By doing so, the server starts to load the most important chucks of the graph into its memory. Thus, the server will reply to the most common queries really fast (Jim, 2011). This is the closest it can get to load the whole graph into memory. This technique is called Cache Sharding, and it makes Neo4j highly available.

Neo4j is very good at dealing with failures. As mentioned before, the cluster spreads portions of the graph across all servers, and data is also redundantly stored. If a server stops working, the other servers with the same data start responding to the incoming read requests (Montag, 2013).

Neo4j chose consistency and availability from the CAP theorem. It is highly available since all slave servers respond to read requests, and it is highly consistent since only the master server writes to the cluster. This implies some problems when scaling though. Read horizontal scaling is easy. If one sever serves 100 requests per second, and the cluster has three servers, then the cluster can respond 300 requests per second. If a forth server is added, then the cluster will be able to serve 400 requests per second, and so on (Sashin, 2013). However, write scaling faces a problem. Since only one master server can write, scaling can only go vertically, which means that the master server needs to be upgraded to be faster (Hoff, 2009).

CouchBase

CouchBase is an open source NoSQL document database managed by CouchBase Inc. The company headquarters are located in Mountain View, California with offices in San Francisco, Bangalore and the United Kingdom. In 2011 CouchBase Inc. was created by the merger of Membase Inc., and CouchOne Inc. Membase Inc. was created on 2009. It developed and maintained the NoSQL key-value database MemBase that was really fast. CouchOne Inc. was also created in 2009. It developed and maintained CouchDB, another open source NoSQL document database that is very popular for its features. The union of both companies meant the combination of MemBase in-memory which has great speed technology and CouchDB which has great features to create CouchBase an easily scalable, high performance document oriented database.

CouchBase data model is a key-value pair. The value is a JSON version of the object that is being stored. The key is a string defined by the user. The structure that contains the key-value pairs, or in other words the “database” is called bucket. An application may start any number of buckets. A bucket is a software structure that reserves hardware, such as disk space, RAM, etc for the database. In addition, It contains all the information necessary for the server and the client to map a database to an application. The bucket’s section that stores the database is broken into several pieces. Each piece is called vBucket (“VBuckets: The Core Enabling Mechanism for Couchbase Server Data Distribution”, 2012). A vBucket is just a container that holds key-value pairs. A bucket is broken into vBuckets so that the cluster may store and replicate the “database” in several nodes for protection against node failure (“Couchbase Server Under the Hood”, 2013).

To communicate to the cluster, an application has a map, which is part of the couchbase API, that takes as input a documentID, and it outputs the node that contains the vBucket, so that the client may contact that node directly. Couchbase does not have master nodes. The client comunicates directly to any node in the cluster depending on the documentID. Therefore, when the application makes a read request, the documentID is given to the vBucket map. The map returns the node that contains the vBucket. The application sends the documentID (documentID and key are the same thing) and the read request to the node. The node receives the read request and hashes the key to find the value. It then returns the value to the application (“Document basics”, 2014).

When the application writes, the procedure is very similar. The application makes a new key, and grabs the value to be stored. It enters the key or in other words the documentID into the vBucket map to get the node that will be storing this document. The application sends the key, value, and write request to the specified node. The node hashes the key. It receives some place in the memory disk, and it proceeds to store the document (“VBuckets”, 2014).

In terms of storage, couch base has two types of buckets which are memcached and couchbase. Memcached buckets store everything in memory. They are extremely fast, and They scale horizontally with no problem. However, they have some limitations. The item size limit is 1MB. They do not feature persistence, replication, backup, and encryption. On the other hand, a couchbase bucket allows files as big as 20MB. It persists data to disk in form of vBuckets. vBuckets are replicated, re-balanced, backed up across nodes in the cluster (Norbye, 2013). A couchbase bucket also keeps track of statistics. It allows to store encrypted data in form of binaries, and the bucket it self may be given a password for protection (“Bucket disk storage”,   2015).

In addition, couchbase clusters have no problem scaling horizontally. As soon as the cluster detects new nodes, it transfers some vBuckets from other nodes to the new node, and all vBuckets maps in the client applications update, so that they may map to the new nodes. Since couchbase does not have master servers, the servers in the cluster are in constant communication and when servers are added or fail, documents are re-balanced and transferred from node to node (Biyikoglu, 2014).

In conclusion. SQL databases were great when application started being developed because data was small and very structured. When data started growing and it started losing structure, SQL databases started having performance problems. The solution was to have databases that scale and hold large amounts of data with more flexible unstructured schema. The new movement was called NoSQL. NoSQL databases may be divided in four main categories, Column based, graph based, key-value based, and document based.

BigTable being a column based NoSQL database is great to store huge amounts of data. It allows to keep copy of several versions of the same data due to the three dimensional table structure. It has no problem scaling horizontally. When new servers are added the system starts using them immediately.

Neo4j is a graph based NoSQL database. It stores everything as a node, or an edge, or an attribute. It is great for information with huge amounts of relationships, such as social media, and family trees. It queries relationships super fast and adding complexity to the graph does not slow down finding relationships since old relationship stay the same. Since Neo4j decided to be consistent and available rather than partitioned, it has some problem scaling horizontally for write requests. However, read request scale horizontally since parts of the graph are loaded to the memory of all the servers in the cluster.

Finally, Couchbase is document based NoSQL database. It is great for huge amounts of data that lack structure. It is very fast since most of the vBuckets are stored in memory for fast access. It has no problem scaling horizontally. Since it does not have master nodes, all nodes detect when new nodes are added to the cluster, and they coordinate to re-distribute vBuckets to the new nodes.

References

Armbruster, S. (2014). Introduction to Graph databases and Neo4j. Retrieved 2015, from http://www.slideshare.net/barcelonajug/introduction-to-graph-databases-and-neo4j-by-stefan-armbruster

Biyikoglu, C. (2014). Multi-tenancy with Couchbase Server. Retrieved 2015, from http://blog.couchbase.com/multi-tenancy-couchbase-server

Couchbase (2012). VBuckets: The Core Enabling Mechanism for Couchbase Server Data Distribution. Retrieved October 15, 2015, from http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/Technical-Whitepaper-Couchbase-Server-vBuckets.pdf

Couchbase (2013). Couchbase Server Under the Hood. Retrieved 2015, from http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/Couchbase_Server_Architecture_Review.pdf

Couchbase (2014). Document basics. Retrieved 2015, from http://docs.couchbase.com/developer/java-2.0/documents-basics.html

Couchbase (2014). VBuckets. Retrieved 2015, from http://docs.couchbase.com/admin/admin/Concepts/concept-vBucket.html

Couchbase (2015). Bucket disk storage. Retrieved 2015, from http://developer.couchbase.com/documentation/server/4.0/architecture/core-data-access-bucket-disk-storage.html

Gribble, S. (2013). Big Table Course. Retrieved 2015, from http://courses.cs.washington.edu/courses/csep552/13sp/lectures/6/bigtable.pdf

Grigorik, I. (2008). Scalable Datasets: Bloom Filters in Ruby. Retrieved 2015, from https://www.igvita.com/2008/12/27/scalable-datasets-bloom-filters-in-ruby/

Hoff, T. (2009). Neo4j - A Graph Database. Retrieved 2015, from http://highscalability.com/neo4j-graph-database-kicks-buttox

Jim, J. (2011). Scaling Neo4j with Cache Sharding and Neo4j HA. Retrieved 2015, from http://jimwebber.org/2011/02/scaling-neo4j-with-cache-sharding-and-neo4j-ha/

Kohler, E. (2011). Notes on Bigtable: A Distributed Storage System for Structured Data. Retrieved 2015, from http://read.seas.harvard.edu/cs261/2011/

Malviya, R. (2010). Big Table – A Distributed Storage System For Data. Retrieved 2015, from http://www.cse.buffalo.edu/~mpetropo/CSE736-SP10/slides/seminar100409b1.pdf

Mehra, A. (2015). Introduction to Apache Cassandra's Architecture. Retrieved October 15, 2015, from https://dzone.com/articles/introduction-apache-cassandras

Montag, D. (2013). Understanding Neo4j Scalability. Retrieved 2015, from http://info.neo4j.com/rs/neotechnology/images/UnderstandingNeo4j Scalability(2).pdf

Neo4j (2015). Neo4j – The World's Leading Graph Database. Retrieved 2015, from http://neo4j.com/company/

Norbye, T. (2013). What is vbuckets, and should I care? Retrieved 2015, from http://trondn.blogspot.com/2013/09/what-is-vbuckets-and-should-i-care.html

Psaroudakis, I. (2012). Bigtable and Dynamo. Retrieved 2015, from http://www.slideshare.net/kingherc/bigtable-and-dynamo

Sashin, S. (2013). How To Use Cache Sharding: Scale Out Neo4j. Retrieved October 15, 2015, from http://www.fromdev.com/2013/10/neo4j-cache-sharding-scale-out.html

Weber, J. (2015). Impossible Is Nothing: The History (& Future) of Graph Data. Retrieved 2015, from http://neo4j.com/blog/history-and-future-of-graph-data/