RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's

Rabi Prasad Padhy
Oracle India Pvt. ltd.
Bangalore, India
rabi.padhy@gmail.com

Manas Ranjan Patra
Dept. of Computer Science
Berhampur University
Berhampur, India
mrpatra12@gmail.com

Suresh Chandra Satapathy
Dept. of CSE,
ANITS
Sangivalasa, India
sureshsatapathy@ieee.org

Abstracts - **In the world of cloud computing, one essential ingredient is a database that can accommodate a very large number of users on an on-demand basis. Distributed storage mechanisms are becoming the de-facto method of data storage for the new generation of web applications used by companies like Google, Amazon, Facebook, Twitter, Salesforce.com, Linkedin.com and Yahoo! etc., which are processing large amount of data at a petabyte scale. The main limitations with RDBMS are it is hard to scale with Data warehousing, Grid, Web 2.0 and Cloud applications, non-linear query execution time, unstable query plans and static schema. Even though RDBMS have provided database users with the best mix of simplicity, robustness, flexibility, performance, scalability and compatibility still next generation NoSQL (not only SQL) databases mostly being non-relational, distributed, open-source and horizontally scalable. The main characteristics of these databases are schema-free, no join, non-relational, easy replication support, simple API and eventually consistent.**

The aim of this paper is to provide a better understand of the non-relational database design, architecture and comparison between them and also identify important research directions in this increasingly important area.

Keywords: NoSQL, Database Architecture, Distributed Databases, Grid Computing, Cloud Computing, Cloud Databases.

I. Introduction

The new generation of applications like Business Intelligence, Web 2.0, Social networking requires processing of terabytes and even petabytes of data. This is achieved by distributed processing. This is one of major reasons for the power of web companies such as Google, Amazon, Salesforce.com, facebook, twitter and Yahoo!. Relational databases are found to be inadequate in distributed processing involving very large number of servers and handling Big Data applications.

There are several reasons for distributed processing. On one hand, programs should be scalable and should take advantage of multiple systems as well as multi-core CPU architectures. On the other end, website servers have to be globally distributed for low latency and failover. We believe that RDBMS systems still have their place and use

cases (such as business intelligence applications). However, the web has changed the requirements of storage database systems for the next generation of applications.

A relational database is a data structure that allows you to link information from different 'tables', or different types of data buckets. A data bucket must contain what is called a key or index (that allows to uniquely identify any atomic chunk of data within the bucket). Other data buckets may refer to that key so as to create a link between their data atoms and the atom pointed to by the key.

A non-relational database just stores data without explicit and structured mechanisms to link data from different buckets to one another. It also called as NoSQL's database's which has many forms (e.g. document-based, graph-based, object-based, key-value store, etc.).

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.

A. Types of NoSQL Databases:

On a basic level, there are three core categories of NoSQL databases

- Key-value Stores: Data is stored as key-value pairs such that values are indexed for retrieval by keys. These systems can hold structured and unstructured data. An example is Amazon's SimpleDB.

- Column-oriented Databases: These types of databases contain one extendable column of closely related data rather than sets of information in a strictly structured table of columns and rows as is found in relational databases. The ColumnFamily databases stem from Google's internally-used BigTable, Cassandra and HBase.

- Document-based Stores: Data is stored and organized as a collection of documents. Users are allowed to add any number of fields of any length to a document. They tend to store JSON-based documents in their databases. Examples of document databases include MongoDB, Apache CouchDB.

B. Characteristics of NoSQL Databases:

- CAP Theorem - Essentially NoSQL databases pick two of the three principals (Consistency,

Availability, Partition-tolerance) of the CAP theorem. Many of the NoSQL databases have loosened up the requirements on Consistency in order to achieve better Availability and Partitioning. This resulted in systems known as BASE (Basically available, Soft-state, Eventually consistent).

- Vertically and Horizontally Scalable - Traditionally relational databases reside on one server which can be scaled by adding more processors, memory and storage to provide scalability. Relational databases residing on multiple servers usually use replication to keep the databases synchronized. NoSQL databases can reside on a single server but more often are designed to work across a cloud of servers.

- Key/Value Store - Relational databases are usually comprised of a collection of columns in a table and/or view (fixed schema, join operations). NoSQL databases often store a combination of key and value pairs or Tuples (schema free).

- In-Memory Dataset, On-Disk Storage - Relational databases almost always reside on a disk drive or a storage area network. Sets of database rows are brought into memory as part of SQL select or stored procedure operations. Some of the NoSQL databases are designed to exist in memory for speed and can be persisted to disk.

- Document-Oriented - Document-oriented databases contain records of documents, fields and XML. Relational databases are characterized by data organized in tables, rows and columns. SQL select operations normally return cursors to a single row or a set of rows containing the columns specified. NoSQL database election operations are often done in code or an interface.

I. Amazon SimpleDB

Amazon SimpleDB is a highly available, scalable, and flexible non-relational data store that enables to store and query data items using web services requests. It requires no schema, automatically indexing and providing a simple API for storage and access. Data is accessed by http through REST and SOAP protocols.

SimpleDB consists of multiple domains and each domain stores a set of records. Each record has a unique key and a set of attributes, which may or may not be present in every record. Within each domain, the records are automatically indexed by every attribute. The main operations are to read/write a record by key or to search for a set of records by one or more attributes.

A. SimpleDB Data Model:

Data is stored in domains which are only defined by their name and domain size cannot exceed 10 Giga. Domains contain

Items. It is normally not possible to have more than 100 domains per account. An Item is made of an Item Name (unique ID in the domain) and several Attributes. An Attribute is made of an attribute name and one or several values. Attribute values can only have String type. Items in one domain can have different attribute name.

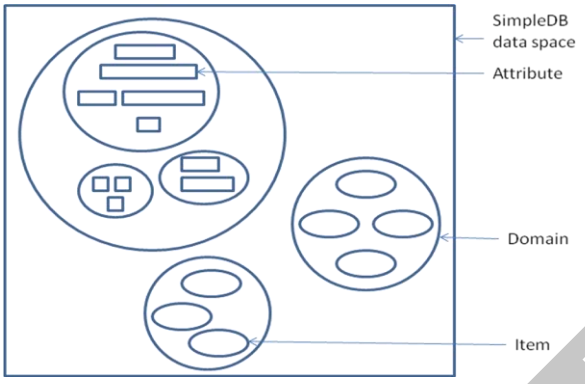

Figure 1: Amazon Simple data model

SimpleDB lets the client organize the data into domains, which can be compared with tables in relation databases, with the difference that a domain can contain a different set of attributes for each item. All attributes are byte arrays with a maximum size of 1024 bytes. Each item can have multiple values for each attribute. Due to restrictions in the Query Model it is impossible to model relations of objects with SimpleDB without creating redundant information. So the developer can either denormalize its data or handle relations in the application layer.

Queries can only be processed against one domain, so if a client needs to aggregate data from different domains, this must also be done in the application layer. But it would be unwise to put everything into one domain, because domains are used in SimpleDB to partition the data, which results in restrictions in the size of one domain. Furthermore, the query performance is dependent on the size of such a domain.

B. SimpleDB Architecture:

SimpleDB is based on Amazon's S3 – Simple Storage Service in which users are granted unlimited data storage capacity at very inexpensive rates. Data in the S3 system is stored across a number of servers or storage devices in the Amazon scalable distributed network.SimpleDB and S3 are extensions of the Cloud Computing Architecture in which computing resources, software applications, and data are shared across the web on a demand basis.

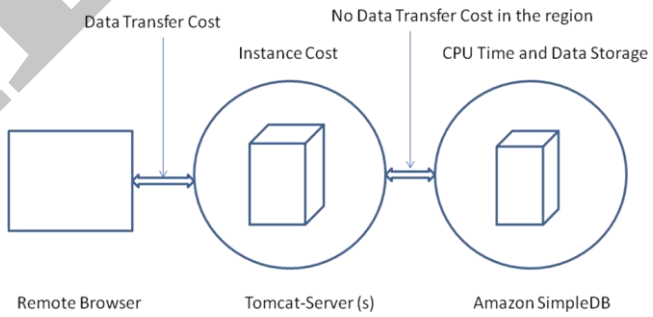

Figure 2: Amazon SimpleDB Architecture and its Components

Software resources such as Applications and Data are stored on distributed servers so when a user demands use of a Word Processing Application, that instance is provided to the user via the web browser. So, the user need not keep or store applications or data on their computing devices but will depend on the reliability and availability of the internet for access.

	Google BigTable	Amazon SimpleDB	Apache CouchDB	MangoDB	Cassandra	Hbase
Data Model	Column database	Document Oriented	Document Oriented (JSON)	Document Oriented (BSON)	Column database	Column database
Interface	TCP/IP	TCP/IP	HTTP/REST	TCP/IP	TCP/IP	HTTP/REST
Storage Type	Columns	Document	Document	Document	Columns	Columns
Data Storage	GFS (Google File System)	S3 (Simple Storage Solution)	Disk	Disk	Disk	Hadoop
Query Method	Map/Reduce	string-based query language	Map/Reduce	Map/Reduce	Map/Reduce	Map/Reduce
Replication	Asynchronous / Synchronous	Asynchronous	Asynchronous	Asynchronous	Asynchronous	Asynchronous
Concurrency Control	Locks	None	MVCC (Multi Version concurrency Control)	Locks	MVCC (Multi Version concurrency Control)	Locks
Transactions	Local	No	No	No	Local	Local
Written In	C, C++	Erlang	Erlang	C++	Java	Java
Operating System	Linux Mac OS X Windows	Linux Mac OS X Windows	Linux Mac OS X Windows	Linux Mac OS X Windows	Linux Mac OS X Windows	Linux Mac OS X Windows
Characteristics	Consistency High Availability Partition Tolerance Persistence	Highly Available And Scalable	High Availability Partition Tolerance Persistence	Consistency Partition Tolerance Persistence	High Availability Partition Tolerance Persistence	Consistency Partition Tolerance Persistence

Table 1. Comparison between various non-relational databases.

II. Apache CouchDB

CouchDB is a document-oriented database server, accessible through REST APIs. Couch is an acronym for "Cluster Of Unreliable Commodity Hardware", emphasizing the distributed nature of the database. CouchDB is designed for document-oriented applications, such as forums, bug tracking, wiki, email, etc. The CouchDB project is part of the Apache Foundation and is completely written in Erlang. Erlang was chosen as programming language, because it is very well suited for concurrent applications through its light-weight processes and functional programming paradigm. CouchDB is ad-hoc and schema-free with a flat address space.

CouchDB is not only a NoSQL database, but also a web server for applications written in JavaScript. The advantage of using CouchDB as a web server is that applications in CouchDB can be deployed by just putting them into the database and that the applications can directly access the database without the overhead of a query protocol.

A. CouchDB Data Model:

Data in CouchDB is organized into documents. Each document can have any number of attributes and each attribute itself can contain lists or even objects. The Documents are stored and accessed as JSON objects, this is why CouchDB supports the data types String, Number, Boolean and Array.

Each CouchDB document has a unique identifier and because CouchDB uses optimistic replication on the server side and on the client side, each document has also a revision identifier. The revision id is updated by CouchDB every time a document is rewritten.

Update operations in CouchDB are performed on whole documents. If a client wants to modify a value in a document, it has first to load the document, make the modifications on it and then the client has to send the whole document back to the database. CouchDB uses the revision id included in the document for concurrency control and therefore can detect if another client has made any updates in the meantime.

The query model of CouchDB consists of two concepts one is Views which are build using MapReduce functions and another is HTTP query API, which allows clients to access and query the views. A View in CouchDB is basically a collection of key-value pairs, which are ordered by their key. Views are build by user specified MapReduce functions, which are incrementally called whenever a document in the database is updated or created. Views should be specified before runtime, as introducing a new View requires that its MapReduce functions are invoked for each document in the databases. This is why CouchDB does not support dynamic queries.

B. CouchDB Architecture:

There are three main components of CouchDB i.e. Storage Engine, View Engine and Replicator.

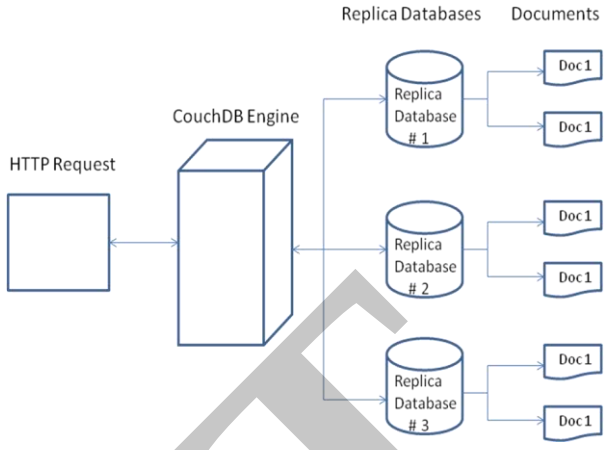

Figure 3: Simple Architecture of Apache CouchDB database

Storage Engine: It is B-tree based and the core of the system which manages storing internal data, documents and views. Data in CouchDB is accessed by keys or key ranges which map directly to the underlying B-tree operations. This direct mapping improves speed significantly.

View Engine: It is based on Mozilla SpiderMonkey and written in JavaScript. It allows creating adhoc views that are made of MapReduce jobs. Definitions of the views are stored in design documents. When a user reads data in a view, CouchDB makes sure the result is up to date. Views can be used to create indices and extract data from documents

Replicator: It is responsible for replicating data to a local or remote database and synchronizing design documents

III. Google's Big Table

BigTable is a distributed storage system for managing structured data. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. It has several goals like wide applicability, scalability, high performance, and high

availability. It is built on Google File System "GFS" and Google offers access to it as part of Google App Engine which is built on top of DataStore and DataStore is built on top of Bigtable. Bigtable is used by more than sixty Google products and projects including Google Analytics, Google Finance, Orkut, Personalized Search and Google Earth.

- BigTable can handle millions of reads/writes per second.
- BigTable is a distributed hash mechanism built on top of GFS.

A. BigTable Data Model:

Instances of Bigtable are run on clusters and each instance can have multiple tables. A bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key and a timestamp, each value in the map is an uninterpreted array of bytes.

The data in the tables is organized into three dimensions: Rows, Columns and Timestamps

(row:string, column:string, time:int64) → string

Columns: Column keys are grouped into column families. Data stored in a column family is usually of the same type. A column family must be created before data can be stored under any column key in that family. A column key is named using the below syntax: family: qualifier

Timestamps: Each cell can hold multiple versions of the same data, these versions are indexed by timestamp (64-bit integers). Timestamps can be set by Bigtable or client applications. Garbage collection mechanism

- It doesn't support joins or SQL type queries.
- Machines can be added and deleted while the system is running and the whole system just works.

Google App Engine and DataStore: Google App Engine is platform as a service technology. It virtualizes applications across multiple servers. Google App Engine is free up to a certain level of used resources. Fees are charged for additional storage, bandwidth, or CPU cycles required by the application.

A cell is the storage referenced by a particular row key, column key and timestamp

Rows: Bigtable maintains data in alphabetical order by row key. The row keys in a table are arbitrary strings. Rows are the unit of transactional consistency. Several rows are grouped in tablets which are distributed and stored close to each other. Reads of short row ranges are efficient, typically require communication with only one or a few machines

discards irrelevant versions and there are two options for this i.e. either keep last n versions or keep if recent enough

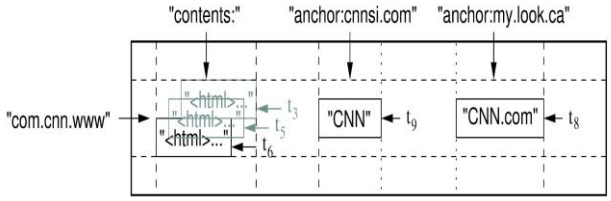

Figure 4: An example of table that stores Web pages

Example of data model: Zoo

row key	col key	timestamp
(zebras,	length,	2006 → 7 ft
(zebras,	weight,	2007 → 600 lbs
(Zebras,	weight,	2006 → 620 lbs

In the above example each col key is sorted in lexicographic order and timestamp ordering is defined as most recent appears first

B. BigTable Architecture:

Bigtable is built on top of Google File System. The underlying file format is SSTable. SSTables are designed so that a data access requires, at most, a single disk access. An SSTable, once created, is never changed. If new data is added, a new SSTable is created. Once an old SSTable is no longer needed, it is set out for garbage collection. SSTable immutability is at the core of Bigtable's data checkpointing and recovery routines.

Bigtable maintains data in lexicographic order by row key. It has a focus on quick reads from columns, not rows. The row keys in a table are arbitrary strings (currently up to 64KB in size). Every read or write of data under a single row key is atomic. The row range for a table is dynamically partitioned. Each row range is called a tablet, which is the unit of distribution and load balancing. The tablets are compressed using the secret algorithms BMDiff and Zippy.

Each data item is stored in a cell which can be accessed using a row key, column key, or timestamp. Each row is stored in one or more tablets and a tablet is a sequence of 64KB blocks in a data format called SSTable.

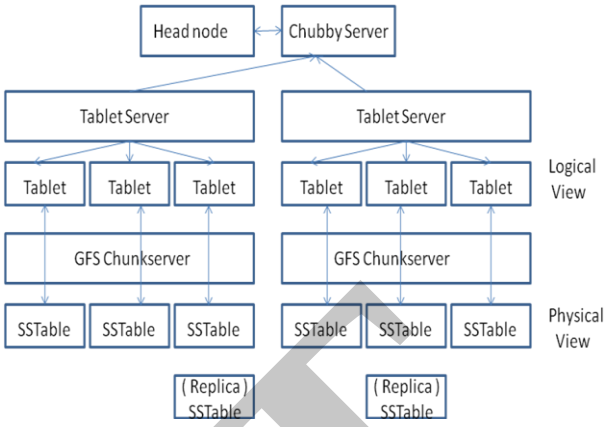

Figure 5: Google BigTable Architecture

BigTable has three different types of servers namely Master servers, Tablet servers and Lock servers

Master servers - The Master servers assign tablets to tablet servers, balances the tablet server load, detects the loss or addition of tablet servers, performs garbage collection and some other chores. Importantly, client data doesn't move through the master. In fact, the system is architected in such a way that most clients never communicate with the master, which helps keeps the master lightly loaded in practice. The master keep track of the set of live tablet servers and current assignment of tablets to tablet servers, including which tablets are unassigned. Whenever a tablet is to be loaded, the master identifies a free tablet server and sends the load request to it. The master server communicates with the tablet servers to ensure that they continue to hold on to their corresponding locks. If the tablet server has lost its lock or the master is unable to reach the tablet server, then the master server tries to get exclusive lock on the corresponding file in chubby service and deletes the lock. When a new master comes up, it first acquires the master lock in Chubby service. Then it communicates with all live tablet servers to get to know the list of tablets that they are currently service.

Also, the master reads the METADATA (SSTable) to know all the tablets.

Tablet servers - Each tablet server typically manages between 10 and 1,000 tablets. Each tablet size averages about 100-200 MB and Each BigTable consists of multiple tablets. Each tablet contains all the data of a group of rows. A newly created table consists of one tablet. As it grows it is dynamically broken into multiple tablets. This allows the system to automatically scale in a horizontal fashion. The Tablet servers process read/write requests for tablets. They split tablets when they exceed size limits (usually 100MB - 200MB). When a tablet server fails, then a 100 tablet servers each pickup 1 new tablet and the system recovers. Tablets are cached in RAM as much as possible. Tablet location metadata is stored as a 3 level index. The root level is stored in Chubby itself.

Lock servers - The Lock servers called as Chubby form a distributed lock service that allows a multi-thousand node Bigtable cluster to stay coordinated. Chubby itself is a cluster app. Chubby is architected to keep lock management traffic very light. Chubby also rules over tablet server life and death, stores access control lists, data schemas and the bootstrap location of Bigtable data. Operations like opening a tablet for writing, Master aribtration, and access control checking require mutual exclusion. Chubby service is used to track live tablet servers. Each tablet server when it comes up creates and holds an exclusive lock to a unique file in the Chubby service.

V. Cassandra

Cassandra is a fully distributed share nothing database. It is fault resistant, persistent and it can be used with a Hadoop cluster to run MapReduce jobs. Cassandra tries to take the best from both Bigtable and Dynamo to build a new kind of distributed database.

A. Cassandra Data Model:

The Cassandra data model is based on the below concepts:

• The Keyspace: a namespace for the ColumnFamilies
• The ColumnFamilies: are sets of Columns
• The SuperColumns: are Columns that can themselves contains Columns
• The Columns: are defined by a name, a value and a timestamp

Usually there is one Keyspace for each application that uses the Cassandra cluster. The Keyspace configuration is very important as it is the one that defines the replication factor as well as the replica placement strategy. The ColumnFamilies can be seen as table in traditional RDBMSs, they are defined in static configuration files and cannot be changed during execution of the cluster.

New data is added in form of a row, like in a relational database, except that the column of the row can be different for each row. Each row is identified by a unique Key and it contains one or more Column Family that them selves contains Columns.

The query model provided by Cassandra allows the user to access directly (for both read and write operations) to a particular column in a particular row as well as it is possible to get the whole line. It is also possible to define Slice Predicate, similar to mathematical predicate, that is described as a property that the elements of a set have in common. The SlicePredicate is useful when

a subset of Columns is needed, it can be defined in two way:

• With a list of columns names
• With a SliceRange describing how to range, order and/or limit the slice.

Finally, if Cassandra has access to a fully functional Hadoop cluster, Hadoop MapReduce jobs can be used to make complex computations on the data stored into Cassandra.

B. Cassandra Architecture:

The Cassandra cluster is collection of nodes. One leader node is elected by the ZooKeeper while other nodes are responsible for storage. The leader node assigns ranges and replicas to the storage nodes. Data is partitioned using consistent hashing. Each node is assigned a number that represents its position in the ring. Each data item is assigned to a node in the ring by hashing its key and moving clockwise in the ring to find the first node with a larger position. Each node is the coordinator for any data item that falls in the ring region between the node itself and the previous node. The coordinator stores a copy of the data item and replicates it on different nodes in the ring. Data is replicated on a number of nodes based on a replication factor N and a replica placement strategy. The replication factor is user defined and is set on the instance level. Cassandra supports three placement strategies namely Simple Strategy ,Old Network Topology Strategy and Network Topology Strategy . The leader is responsible for balancing the load over nodes so that no node is responsible for storing more than N-1 replicas. Meta data such as data Item-to-node mappings are stored in and cached in nodes. A read/write request for a key routes to any node in the cluster. For write operations, the system routes the update to all replicas of data and waits for a minimal number of replicas to confirm the success of the write.

VI. mangoDB

MongoDB is a schema less document oriented database developed by 10gen and an open source community. The name mongoDB comes from "humongous". The database is intended to be scalable and fast and is written in C++. In addition to its document oriented databases features, mongoDB can be used to store and distribute large binary files like images and videos. It is fault tolerant, persistent and provides a complex query language as well as an implementation of MapReduce.

A. mangoDB Data Model:

MongoDB stores documents as BSON (Binary JSON) objects, which are binary encoded JSON like objects. BSON supports nested object structures with embedded objects and arrays like JSON does. MongoDB supports in-place modifications of attributes, so if a single attribute is changed by the application, then only this attribute is send back to the database.

Each document has an ID ″ field, which is used as a primary key. To enable fast queries, the developer can create an index for each query-able ″ field in a document. MongoDB also supports indexing over embedded objects and arrays.

For arrays it has a special feature, called "multikeys": This feature allows to use an array as index, which could for example contain tags for a document. With such an index, documents can be searched by their associated tags.

Documents in MongoDB can be organized in so called "collections". Each collection

can contain any kind of document, but queries and indexes can only be made against one collection. Because of MongoDB's current restriction of 40 indexes per collection and the better performance of queries against smaller collections, it is advisable to use a collection for each type of document. Relations in MongoDB can be modeled by using embedded objects and arrays. Therefore, the data model has to be a tree. The first option would imply that some documents would be replicated inside the database. This solution should only be used, if the replicated documents do not need very frequent updates. The second option is to use client side joins for all relations that cannot be put into the tree form. This requires more work in the application layer and increases the network traffic with the database.

B. mangoDB Architecture:

A MongoDB cluster consists of three components namely Shard nodes, configuration servers and routing services or mongos.

Shard nodes : Shard nodes are responsible for storing the actual data. Each shard can consist of either one node or a replication pair. In future versions of mongoDB one shard may consist of more than two nodes for better redundancy and read performance.

Configuration servers: The config servers are used to store the metadata and routing information of the MongoDB cluster and are accessed from the shard nodes and from the routing services.

Routing services or mongos : Mongos, the routing processes are responsible for the performing of the tasks requested by the clients. Depending on the type of operation the mongos send the requests to the necessary shard nodes and merge the results before they return them to the client. Mongos for themselves are stateless and therefore can be run in parallel.

MongoDB is implemented in C++ and consists of two types of services: The databases core mongod and the routing and autosharding service mongos. For storage MongoDB uses memory-mapped ″ files, which lets the operating system's virtual memory manager decide which parts of the database are stored in memory and which one only on the disk. This is why MongoDB cannot control, when the data is written to the hard disk.

The motivation for the usage of memory mapped ″ files is to instrument as much of the available memory as possible to boost the performance. In some cases this might eliminate the need for a separate cache layer on the client side. But there is currently an alternative storage engine for MongoDB in development, which will allow MongoDB more control over the timing of read and write operations. Indexes are stored in MongoDB as B-Trees like in most other databases.

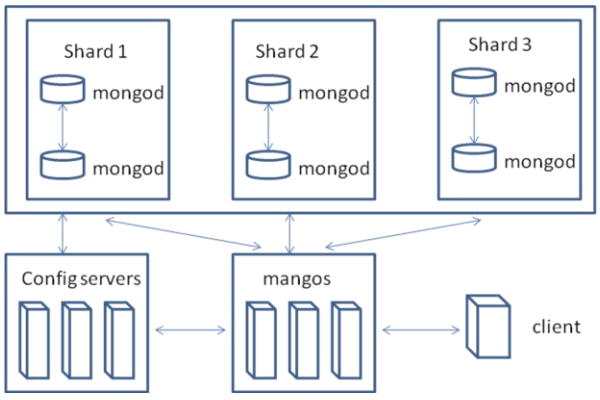

Fig 6. Typical architecture of a MangoDB Cluster

The documents in a MongoDB cluster are partitioned by their owning collection and by a user specified shard key. A shard key in MongoDB is very similar to an index and can contain multiple fields.

This shard key is used to partition the whole collection into shards. Each shard stores its assigned documents ordered by this key. The documents inside a shard are organized into chunks. Each chunk contains an ordered set of documents from one start shard key to a specific end shard key. If a chunk gets too big, it will be split. Chunks are used by MongoDB for the automatic rebalancing of the shards. If the size of one shard is too big, some of its contained chunks are migrated to other shards. Chunks are also used to redistribute the data when new nodes are added or removed. The configuration servers store a record for each chunk in the cluster, consisting of the start and end key of the chunk and its assigned shard. This information is used by the mongos to decide which shard nodes are needed for which requests. Depending on the type of the operation either only one chard or nearly all shards need to be consulted to fulfill a request. For example can a simple find query to search for a document with a specific id be routed only to the shard that stores this document, if the used id is also the shard key. But queries that can not be restricted with the usage of a shard key need to be send to each shard node of the cluster.

VII. HBase

HBase is an open source project, written in Java and developed by the Apache software foundation. HBase works with Apache's Hadoop Distributed File System (HDFS) as basic data storage. It is optimized for high performance real time queries of very huge amounts of distributed data, while datasets are edited seldom but very often inserted. HBase provides a RESTful web service interface supporting XML. It provides a real-time, distributed, versioned and structured database on top of the Hadoop distributed file system. Exactly like Bigtable, the system is made of two layers. First the distributed file system HDFS stores all the data in a persistent way, meaning that it is responsible for data replication, node failure and data distribution across the nodes. The second layer is the one made of HBase itself, where each region server is responsible for a list of regions meaning that it has to record the updates and writes into memtables and it also acts as a cache for the data stored in the HDFS level. It is interesting to describe the architecture of an HBase cluster because it is much more complicated than a system where all the nodes are equal.

A. HBase Data Model:

The HBase data model is based on the below concepts:

- The Tables : they are made of rows and columns
- Every Column belongs to a given column family
- Each row is identified by its key
- A Table cell is the intersection of the a row and a column and is versioned

The Tables are created at the schema definition but new one can easily be added during normal operations. The number of versions is defined at the column family creation step, HBase stores everything as uninterpreted bytes, meaning that almost everything, even a serialized data structure,

can be used as name for a column family or as key for a row.

With HBase, the user of the database can read data using either Gets or Scans. The Gets are used to retrieve all the columns or only a subset of them for a given row. The Scans are used to get all or part of the rows in a given Table. Like the Get operation, the subset of wanted columns can be
specified but unlike it, it is also possible to define a range of rows using rows keys for the start and for the end of the range. HBase Tables can be used as inputs and output for Hadoop MapReduce jobs when more complex computation than simple value lookup or update is needed. The computation will be distributed according to the number of regions available in the cluster, meaning that the number of Map jobs will always be equal to the number of regions.

B. Hbase Architecture:

An HBase cluster is in fact two distinct clusters working together, often on the same servers but not necessarily. The HDFS cluster is composed of one name node, that acts as the entry point of the cluster, meaning that this particular node knows which are the data nodes that store any information wanted by a client.

The HBase cluster is composed of one master, a Zookeeper cluster that acts as the entry point of the cluster and finally the region servers that serve the data. The masters and region servers are all registered into the Zookeeper quorum and if one of them dies it is repaired and replaced by Zookeeper. The master is doing background administrative tasks such as keeping the cluster balanced and moving regions from failing regions servers to other region servers.

Conclusion

Recent trends in both grid and cloud computing are making the non-relational model more desirable. The need to scale quickly, disassociate hardware from the data model and provide more efficient databases are all contributing factors in this transition. The most important aspect of the NoSQL database movement has been the many varieties of database that are available to developers outside the legacy systems. Now developers do not have to settle for the relational model when data needs dictate a different approach to storage. RDBMS won't go away, they're still definitely needed. However, storage requirements for the new generation of applications are huge different from legacy applications. As demands for performance and scalability increase, variation of the key-value database will continue to evolve.

References

[1] Abadi, D.J., Data management in the cloud : Limitations and Oppertunities. IEEE Bulletin of the Technical Commmittee on Data Engineering, Vol. 32, issue 1, pp 3-12 2009.

[2] Dean, D., Ghemawat, S., MapReduce : Simplified Data Processing on Large Clusters. Communications of the ACM, Vol. 51, issue 1, pp 107-113 January 2008.

[3] Franklin, M., Halevy, A., Maier, D, From Databases to Dataspaces: A New

Abstraction for Information Management. ACM SIGMOD Record, Vol.34, issue 4, pp 27-33 2005.

[4] Pokorny, J., Database Architecture: Current Trends and their Relationships to environmental data management. Environmental Modeling & Software, Elsevier Science, Vol.21, issue 11 pp1579 – 586 2006.

[5] Lang, W., Patel, J.M., Towards Eco-friendly Databases management Systems. International Proc. of 4th Biennial Conf. on Innovative Data Systems Research (CIDR), Asilomar, 2009.

[6] Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2006) Bigtable: a distributed storage system for structured data. In: Proceedings of OSDI

[7] Cooper B, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen H, Puz N, Weaver D, Yerneni R (2008) PNUTS: Yahoo!'s hosted data serving platform. In: Proceedings of VLDB

[8] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key- value store. In: Proc. of SOSP, pp 205–220

[9] A. Singhal, "Modern information retrieval: a brief overview", IEEE Data Engineering Bulletin,
Special Issue on Text and Databases, Vol.24, issue 4, Dec. 2001.

[10] R. Pike, S. Dorward, R. Griesemer, S. Quinlan, Interpreting the data: Parallel analysis with Sawzall,

in: Dynamic Grids and Worldwide Computing, Scientific Programming 14 (2006) (special issue).

[11] Codd, E. F. (1970) A Relational Model of Data for Large Shared Data Banks. Communications of

the ACM, Vol.13, issue 6, pp 377-387.

[12] Amazon Elastic Compute Cloud (AmazonEC2): http://aws.amazon.com/ec2/.

[13]Amazon SimpleDB. http://aws.amazon.com/documentation/simpledb/

[14]AzureWatch. http://www.paraleap.com/azurewatch.

[15] Google AppEngine. code.google.com/appengine.

[16]HBase Homepage. http://hbase.apache.org.

[20] The Apache Cassandra Project. http://cassandra.apache.org/.

[21] Windows Azure Platform. http://www.microsoft.com/windowsazure/.

[22] The Apache Couchdb Project. http://couchdb.apache.org/docs/index.html

[23] The Google BigTable. http://labs.google.com/papers/bigtable.html

[24] Robert L Grossman,Yunhong Gu. Data Mining Using High Performance Data Clouds: Experimental Studies Using Sector and Sphere. In Proc. of the 14th ACM SIGKDD International conference on Knowledge discovery and data mining , pp 920-927, 2008.

[25] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska. Building a Database on
S3. In Proc. of SIGMOD, pp 251–264, 2008.

[26] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In IFIP WG 11.3 on Data and Application Security, 2006.

[27] R. Jones, Anti-RDBMS: A list of distributed key-value stores, http://www.metabrew.com/article/anti-rdbms-a-list-of-distributed-key-value-stores/.

[28] R. Rawson, HBase committer, HBase, http://docs.thinkfree.com/docs/view.php?dsn=858186

[29] Lindsay, B.G., et. al., "Notes on Distributed Databases", Research Report RJ2571(33471), IBM Research, July 1979

[30] Urs Hoelzle , Luiz Andre Barroso, "The Datacenter as a Computer: An Introduction to the
Design of Warehouse-Scale Machines," Morgan and Claypool Publishers, 2009.

[31] L.M. Kaufman, "Data security in the world of cloud computing," Security & Privacy, IEEE, vol. 7, 2009, pp. 61–64.

[32] NoSQL systems. http://nosql-database.org/

[33] K.D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," Proceedings of the 16th ACM conference on Computer and
communications security, 2009, pp. 187–198.

[34] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system,"
ACM SIGOPS Operating Systems Review, vol. 44, 2010, pp. 35–40.

[35] Xie Yi,Gao hong wei and Fan chao dong. "A Survey on NoSqlDatabase[J]
"."Communication of modern technology", Vol. 08, pp46-50 , 2010

[36] "NoSQL Relational Database Management System: Home Page". Strozzi.it. 2007-10-02. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page.
Retrieved 2011-08-12.

[37]MongoDB. http://www.mongodb.org/, Retrieved 2011-08-19.

[38] Joe Lennon," Exploring CouchDB-A documentoriented database for Web applications", 31
Mar 2009 [Online]. Available: http://www.ibm.com/developerworks/opensource/librar y/os-couchdb/ [Retrieved 2011-08-02].

[39] "Homepage – CouchDB", Apache Software Foundation, 2008 [Online]. Available: http://couchdb.apache.org/ [Retrieved 2011-08-09].

[40] Alvanos Michalis, "Bigtable: A Distributed Storage System for Structured Data", May 8 2009 [Online]. Available: http://www.csd.uoc.gr/~hy558/reports/report_bigtable.p df [Retrieved 2011-08-20].

Rabi Prasad Padhy

He has achieved his MCA degree from Berhampur University. He carries 8 years of extensive IT Experience with MNC's like EDS, Dell, IBM and Oracle, Rabi Prasad Padhy is currently working as a Senior Software Engineer – Oracle India Private Ltd. Bangalore, His area of interests include IT Infrastructure Optimization, Virtualization, Enterprise Grid Computing, Cloud Computing and Cloud databases. He is a certified professional for Oracle and Microsoft SQL Server database.

Dr. Manas Ranjan Patra

He holds a Ph.D. Degree in Computer Science from the Central University of Hyderabad, India. Currently he is heading the Post Graduate Department of Computer Science, Berhampur University in India. He has about 23 years of experience in teaching and research in different areas of Computer Science. He had visiting assignment to International Institute for SoftwareTechnology, Macao as a United Nations Fellow and for sometime worked as assistant professor in the Institute for Development and Research in Banking Technology, Hyderabad. He has about 75 international and national publications to his credit. His research interests include Service Oriented Computing, Software Engineering, Applications of Data mining and E-Governance. He has presented papers, chaired technical sessions and served in the technical committees of many International conferences.

Dr. Suresh Chandra Satapathy

He is a Professor in Computer Science Engg in ANITS, Vishakapatnam. He has published many IEEE conference papers on Data clustering using PSO, GA etc. His areas of interest include Data mining, machine learning, Swarm Intelligence etc.

