Improving the Availability of NoSQL Databases for Cloud Storage

Phyo Thandar Thant, Thinn Thu Naing
University of Computer Studies, Yangon
pphyothandarthant@gmail.com, thinnthu@gmail.com

Abstract

Data storage management plays a vital role in cloud computing environment. The researchers are still trying to make the cloud data storage system to be more mature. And there are two major requirements of datastores in cloud computing environment: (1) high until almost ultimate scalability and (2) low administration overhead. As a result, scalable, distributed databases to store peta bytes of data becomes a necessity. In this case, NoSQL databases are the most suitable solutions for cloud computing. Three properties of distributed databases that are commonly desired are: CAP (consistency, availability, partition tolerance). It would be ideal if distributed databases can deliver all three guarantees. However, providing all three attributes in NoSQL is still a challenge.This paper intends to examine how NoSQL works and to improve the availability ,by using replication mechanism and latency reducing mechanism, in NoSQL databases that are suitable for cloud computing environment.

1. Introduction

A relational database is a structured collection of data related to some real life phenomena. In traditional, the most widely used database is a relational database. A relational database is one where the database structure is in the form of tables. Formally, a relation R defined over n sets D_1, D_2, D_3, … , D_n is a set of n tuples $<d_1,d_2,\ldots,d_n>$ such that $d_1 \epsilon D_1$, $d_2 \epsilon D_2,\ldots$, $d_n \epsilon D_n$.

The data stored in the database can be retrieved. Applications that use databases are generally designed for complex environments of large information management. So, it can be said that the goal of using database storage is for large scale applications. Although there are several ways to organize data in a database, a relational database is one of the most effective ways as it used a set of mathematical theories to organize data.

However, nowadays, cloud computing technology is very popular and this technology employ NoSQL as it can provide significantly higher data throughput than traditional RDBMS. A number of NoSQL databases exist. Most of them inherit from Google Bigtable and Amazon Dynamo.

This paper intends to present some improvements in the availability for NoSQL databases on the perspectives of number of replications and reducing the latency in querying the data from these databases. The rest of paper is organized as follows: section 2 lists related work, section 3 describes the theory Background, section 4 gives information about the improvements of availability, generally accepted consistency and partition tolerance and section 5 provides conclusion.

2. Related Work

M. A. Olson, K Bostic, M Seltzer from sleepycat software Inc [16] introduced BerkeleyDB. It is an open source embedded database system that can be used in applications requiring high performance concurrent storage and retrieval of key value pairs. G DeCandia, D Hastorun, M Jampani, G Kakulapati, A Lakshman, A Pilchin, S Sivasubramanian, P Vosshall and W Vogels [4] presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance. In 2000 E.Brewer [7] presented the CAP conjecture and the feasibility of consistent, available, partition tolerant web services. In this paper, they prove this conjecture in the synchronous network model and discuss solutions to this dilemma in the partially synchronous model. In [12], Toby J. Teorey and Wee Teck Ng estimate the availability, reliability and mean transaction time (response time) for repairable database configurations, centralized and distributed in which each service component is continuously available for repair.

3. Theory Background

3.1. The Necessity of NoSQL

A distributed database can be defined as a database logically integrated but physically distributed on several machines that can communicate through a network infrastructure. The main advantage of a relational database is data accessibility and ease in retrieving the necessary information. This is highly related with the chosen SQL language and also it's the proper

use. When the situation requires a large set of data, relational databases lose their power. Therefore, distributed relational databases are more and more substituted by non SQL database versions. Figure 1 shows the difference between NoSQL and RDBMS.

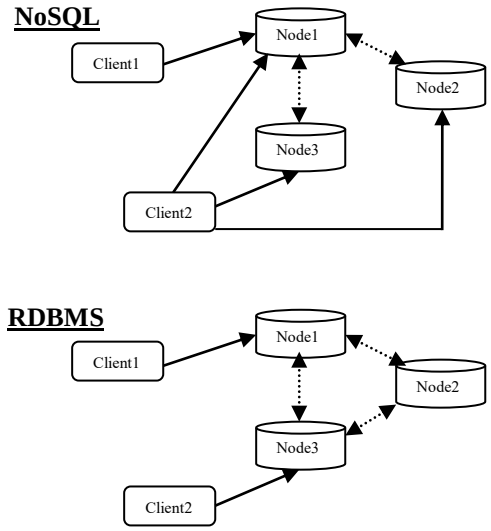

Figure 1. Comparison of NoSQL and RDBMS

3.2. NoSQL

NoSQL solutions tend to shine over their relational counterparts because of their high availability and the ability to concurrently service millions of users by scaling out horizontally among several machines, spread across multiple data centers. The widely usage of NoSQL databases is in the cloud computing environment. Many cloud service providers exist and multiple NoSQL products are available. The most popular service providers are Google and Amazon. And it is difficult to distribute a RDMS while retaining all the features and guarantees. With the rise of the large Internet systems, with their huge amount of data and requests, new solutions for the distribution of databases were needed. This is why within the last years alternative data management systems, so-called NoSQL DMS (or NoSQL data stores), have been created and are becoming more important. There are several types of NoSQL such as:

Key-value stores

- is a system that stores values indexed for retrieval by keys.
- systems can hold structured or unstructured data.

Column-oriented databases

- column-oriented databases contain one extendable column of closely related data.
- Facebook created the high-performance Cassandra to help power its website.

Document-based stores

- store and organize data as collections of documents, rather than as structured tables with uniform sized fields for each record.
- users can add any number of fields of any length to a document.

3.3. CAP Theorem

CAP is first conceived in 2000 by Eric Brewer and formalized into a theorem in 2002 by Nancy Lynch, has become a useful model for describing the fundamental behavior of NoSQL systems[5]. And providing all three properties is still a challenge. The following figure shows the CAP representation.

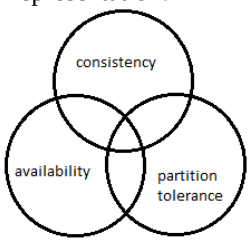

Figure 2. Representation of CAP

3.3.1. Consistency

Consistency is about guaranteeing that a database always appears truthful to its clients. Every operation on the database must carry its state from one consistent state to the next. Consistency can be classified as:

1. **Strict Consistency**
   - Atomic or highest form of consistency.
2. **Sequential Consistency**
   - Every client sees all changes in the same order.
3. **Causal Consistency**
   All changes that are causally related are observed in the same order by all clients.
4. **Eventual Consistency**
   - Eventually all updates will propagate through the system and all replicas will be consistent.
5. **Weak Consistency**
   - No guarantee is made that all updates will propagate and changes may appear out of order to various clients.

The most well known example of strong consistency in databases is **ACID** (Atomicity Consistency Isolation Durability), used in most relational databases. However, most of the NoSQL database system architectures favor partition tolerance and availability over strong consistency. Thus, they provide **BASE** (Basically Available Soft-state Eventual consistency).The following table shows the differences between ACID and BASE.

Table 1. ACID versus BASE

ACID	BASE
Strong consistency Isolation Nested transactions	Weak consistency Availability first Appropriate answers faster

Most often, weak consistency comes in the form of eventual consistency which means the database eventually reaches a consistent state. Eventual consistency is a compromise between strong consistency and weak (no guarantees) consistency. Not all implementations of eventually consistent are equal. In particular, an eventually consistent database may also elect to provide the following:

- **Causal consistency**
  - the receiving session will always see the updated value.
- **Read your own writes**
  - a session that performs a change to the database will immediately see that change, even if other sessions experience a delay.
- **Monotonic consistency**
  - a session will never see data revert to an earlier point in time. Once we read a value, we will never see an earlier value.

3.3.2. Availability

The availability of a system is measured according to its ability to fulfill requests. But computers can experience all manner of failures, from hardware component failure to network disruption to corruption. Any computer is susceptible to these kinds of failure. So, for a system to be highly available, it must typically include multiple networked computers, and the software they're running must then be capable of operating in a cluster and have some facility for recognizing node failures and failing over requests to another part of the system.

High-Availability means a system that is tolerant of node failures and can also remain available during software and hardware upgrades

Replication is an important technique for increasing computer system availability. All the NoSQL databases use replication to provide their high-availability solution. Moreover, reducing the latency in data locality within the network is also important for availability.

3.3.3. Partition Tolerance

Partition tolerance refers to the ability for a system to continue to operate in the presence of a network partitions. If a system is partition tolerant, it will continue to function (respond with correct data) regardless of failure to communicate within the system. Another way to describe partition tolerance is "No set of failures less than total network failure is allowed to cause the system to respond incorrectly." [5]. Partition tolerance is a must in virtually all web scale production systems, realistically partition tolerance is always required. As a result, one have to consider whether the system should provide CP or AP system according to their applications' needs. The following table shows some popular CP, and AP systems.

Table 2. Sample CP, AP Databases

NoSQL databases	Type	Provided CAP
HBase	Column oriented	CP
Cassandra	Column oriented	AP
Hypertable	Column oriented	CP
BigTable	Column oriented	CP
Dynamo	Key value	AP
Voldemort	Key value	AP

3.4 Popular NoSQL

3.4.1. Google BigTable

Google developed a new database called Bigtable in early 2005. It is a distributed storage system for managing structured data that is designed to scale to a very large size (petabytes of data) across thousands of commodity servers. And it is built from the ground up on highly distributed, shared nothing architecture. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications demands both in terms of data size and latency requirements. It provides a flexible, high performance solution for all of these Google products.

The goal of Bigtable is to increase higher scalability and control of performance compared with traditional one. It is designed easily to add more machines to the system and automatically start without any reconfiguration in order to realize high scalability.

Bigtable store each column contiguously on disk with the assumption that in most cases not all columns are needed for data access, column oriented layout allows more records sitting in a disk block and hence can reduce the disk I/O. Column oriented layout is also very effective to store very sparse data as well as multi-value cell.

However, it does not support a full relational data model. Data is indexed using row and column names that can be arbitrary strings. So, it operate differently than traditional relational databases such as supporting joins or multirow transactions. Bigtable can be said that it is a CP system.

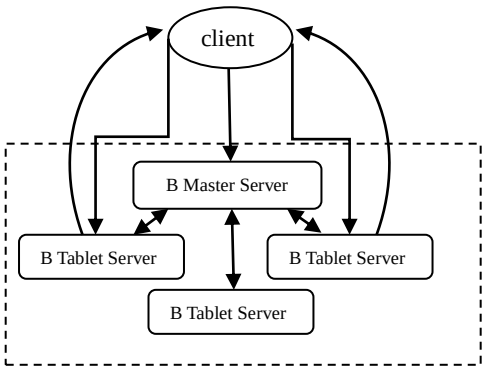

Figure 3. Basic Implementation of Bigtable

Bigtable CAP

Bigtable relies on a highly available and persistent distributed lock service called Chubby. A Chubby service consists of five active replicas, one of which is elected to be the master and actively serve requests. It uses the Paxos algorithm to keep its replicas consistent in the face of failure. Chubby provides a namespace that consists of directories and small files. Each directory or file can be used as a lock, and reads and writes to a file are atomic. The Chubby client library provides consistent caching of Chubby files. If Chubby becomes unavailable for an extended period of time, Bigtable becomes unavailable.

Table 3. Summary of Techniques used in Bigtable

Problem	Techniques
Partitioning	Chubby
High Availability for writes	Chubby
Consistency	Paxo algorithm

3.4.2. Amazon Dynamo

Dynamo data model is that of simple key-value pairs and support random read and write. This model is well suited for many web based commerce applications.

Dynamo does not rely on any underlying distributed file system and instead directly manages data storage across distributed nodes. Objects are key value pairs with arbitrary arrays of bytes. An MD5 hash of the key is used to generate a 128-bit hash value. The range of this hash function is mapped to a set of virtual nodes arranged in a ring, so each key gets mapped to one virtual node. The object is replicated at this primary virtual node as well as $N-1$ additional virtual. Each physical node (server) handles a number of virtual nodes at distributed positions on the ring so as to continuously distribute load evenly as nodes leave and join the cluster because of transient failures or network partitions. Notice that the Dynamo architecture is completely symmetric with each node being equal. Dynamo can be said that it is a AP system.

Dynamo CAP

Dynamo uses distributed object versioning and quorum consistency to enable writes to succeed without waiting for all replicas to be successfully updated. Managing conflicts if they arise is relegated to reads which are provided enough information to enable application dependent resolution.

Dynamo is able to handle transient failures by passing writes intended for a failed node to another node temporarily. Such replicas are kept separately and scanned periodically with replicas being sent back to their intended node as soon as it is found to have revived.

Dynamo targets applications that require only key/value access with primary focus on high availability where updates are not rejected even in the wake of network partitions or server failures.

Dynamo uses a synthesis of well known techniques to achieve scalability and availability: Data is partitioned and replicated using consistent hashing [10]. Dynamo employs a gossip based distributed failure detection and membership protocol. Dynamo is a completely decentralized system with minimal need for manual administration. Storage nodes can be added and removed from Dynamo without requiring any manual partitioning or redistribution. Dynamo was able to scale to extreme peak loads efficiently without any downtime during the busy holiday shopping season.

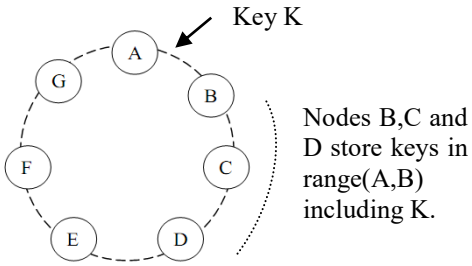

Figure 4. Partitioning and replication of keys in Dynamo Ring

Table 4. Summary of Techniques used in Dynamo

Problem	Techniques
Partitioning	Consistent Hashing
High Availability for writes	Vector clocks
Consistency	Eventual consistency

4. Availability, Consistency and Partition Tolerance

4.1. Improvements in Availability

For high availability, NoSQL uses replication mechanism for storing the data. Generally, there are 3 replicas that are used for availability. Moreover, availability varies with the number of replications.

Availability on Number of Replications

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. Changes applied at one site are captured and stored locally before being forwarded and applied at each of the remote locations. Replication provides user with fast, local access to shared data, and protects availability of applications because alternate data access options exist. Even if one site becomes unavailable, users can continue to query or even update the remaining locations that improves the availability.

If availability is calculated with 1 out of 3 replication system, assuming that the availability of each replica is 0.9, the entire system availability can be calculated for the following cases:

Table 5. Cases versus No of Replicas

	Replica1	Replica2	Replica3	Entire system
Case1	Normal	Normal	Normal	Normal
Case2	Normal	Normal	Failure	Normal
Case3	Normal	Failure	Normal	Normal
Case4	Normal	Failure	Failure	Normal
Case5	Failure	Normal	Normal	Normal
Case6	Failure	Normal	Failure	Normal
Case7	Failure	Failure	Normal	Normal
Case8	Failure	Failure	Failure	Failure

Availability in case1=0.9*0.9*0.9 = 0.729
Availability in case 2= 0.9*0.9*0.1 = 0.081
Availability in case 3=0.9*0.1*0.9 = 0.081
Availability in case 4=0.9*0.1*0.1 = 0.009
Availability in case 5=0.1*0.9*0.9 = 0.081
Availability in case 6=0.1*0.9*0.1 = 0.009
Availability in case 7=0.1*0.1*0.9 = 0.009

In this case, the entire system availability is 0.999 where the failure case (case 8) is 0.001 for RN=3. With RN=4, the availability increases and when RN=5, availability is nearly 1 (0.99999). The following figure shows the increasing availability with respect to the number of replicas (RN).

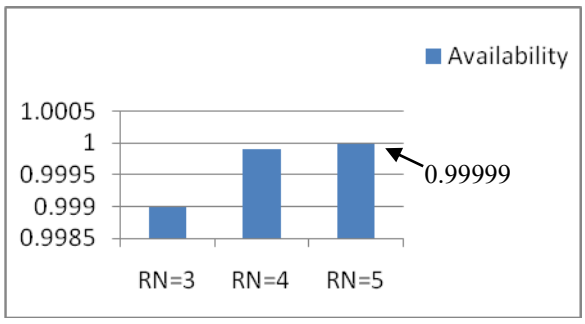

Figure 5. Availability versus No: of replication

Availability on Reducing Latency

Reducing the latency in retrieving the data is also important for the availability of NoSQL datastore. And this paper emphasizes on the NoSQL with peer to peer architecture in which resource routing protocol based on distributed hash table(DHT) named Chord is used.

Chord provides fast distributed computation of a hash function mapping keys to nodes related to them. Both data objects and nodes are assigned an m bits identifier by using a consistent hashing such as SHA-1. Along the Chord ring, all the identifiers including node ids and keys are ordered. In Chord, each node maintains a routing table, called the finger table which have at most m entries[10].

To lookup a given key *k*, a node will check its finger table and forward a lookup message to the finger that is closest to but does not overshoot *k* clockwise. After several iterations or recursions, the lookup message will ultimately arrive at the node that immediately precedes *k*. Then the node will return the address of its successor node, which is also the successor node of *k* where the needed data resides.

Increasing the number of nodes also increases the number of entries in the finger table. Sometimes, there may be duplicate entries in that table. So, specifying the finger table entry is very important for data retrieval. And the paper aims to provide some improvements to specify finger table entry for reducing the latency of routing the needed data with two steps procedure.

Normal Chord Finger Table Entry

In Normal chord finger table entry, the entry at row i identifies the first node that succeeds n by at least 2^{i-1}. So, the entry is calculated with $(n+2^{i-1})$ mod 2^m where $1 \le i \le m$. And there are some duplicated entries in figure tables of nodes that costs space. And if the requested key is less than the current node, it is needed to traverse along the ring until it reaches the destination. That costs the time to get the data.

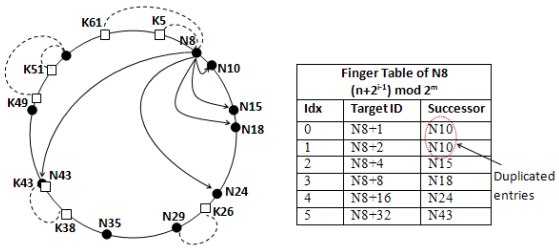

Figure 6. Normal Finger Table Entry for 6 bit chord identifier space

Proposed Finger Table Specification

The **proposed procedure** intends to solve these problems with two steps.

Algorithm : Finger Table Specification

Step 1 : **Finger Table Entry Specification**

- For i =1 to FTSize(cnode)
  if (successor[i]==successor[i+1])then
  dellist[i]=Idx[i]
- ReconstructFT(cnode,dellist)

Step 2 : **Efficient Lookup Service Definition**

(i) if key is in cnode then return cnode
(ii) else
if (key < cnode) then
node=Find-dedicate-node(key,cnode)
cnode=node
go to (i)
(iii) else
do normal routing

Function : Find-dedicate-node(key,cnode)

Step 1 : nodeList=Findsubrange(cnode)
node=key; min=key; j = 1;

Step 2 : While ((j< nodeList.size( )) AND
(key>nodeList[j]))

(i) If (min > (key – nodeList[j]))
Begin
min = key – nodeList[j]
node=nodeList[j]
End

(ii) Increase j by 1

Step 3 : return node;

Using the above two steps procedure and Find-dedicate-node( ), the improved finger table entry and efficient lookup service can be obtained by dedicating the node that is closest to the requested key. And improved finger table entry specification can reduce the finger table size by eliminating some duplicated entries. With efficient lookup, the latency in getting the data is reduced and that leads to improve the data availability.

4.2. Consistency

One of the most significant differences between the new generation of non-relational (NoSQL) databases and the traditional RDBMS is the way in which consistency of data is handled. In a traditional RDBMS, all users see a consistent view of the data. Once a user commits a transaction, all subsequent queries will report that transaction and certainly noone will see partial results of a transaction.

In general, the most widely accepted method for consistency in NoSQL is eventual consistency. Eventual consistency can be illustrated with NRW notation.

NRW Notation

NRW notation describes at a high level how a distributed database will trade of consistency, read performance and write performance.

N= no of copies of each data item that the database will maintain
R = no of copies that application will acess when reading the data item
W= no of copies of data item that must be written before the write can complete

Table 6. NRW configuration and corresponding outcome

NRW configuration	Outcome
W=N, R=1	Read optimized strong consistency
W=1, R=N	Write optimized strong consistency
W+R <= N	Weak eventual consistency
W+R>N	Strong consistency through quorum

When N=W then the database will always write every copy before returning control to the client. If it is more concerned about write performance than read performance, then it can be set W=1, R=N. Then each read must access all copies to determine which is correct, but each write only has to touch a single copy of the data. Most NoSQL databases use N>W>1: more than one write must complete, but not all nodes need to be updated immediately.

4.3. Partition Tolerance

The most widely mechanism for partition tolerance is consistent hashing. Consistent hashing performs the followings:

(1) map node IDs to a large circular space
(2) map keys to the same circular space
(3) key belongs to the nearest node

Consistent hashing can provide good load balancing.

Partition= key mod (total virtual nodes)

In consistent hashing, each of the available machines is placed on consistent hashing ring around a circle based on the calculation of hash table such as machine id. To determine where to store the object or key, the object finds the corresponding point on the ring by hash table functionality. Then it walks clockwise to

determine the target nodes, by selecting the next highest angle which an available node points to. The hash code is ranged between $0 \sim 2^{32}$. If no machine is found, then it is saved to the first machine on the ring.

If a node is newly added, the affected nodes are only those that were located between the previous node. So, only a minimum changes are needed in the network because of the consistent hashing. It provides good partition tolerance capability.

5. Conclusion

This paper introduces the concepts of NoSQL and CAP theorem and presents the availability of NoSQL databases on two different views. Providing three properties of distributed databases (CAP) in NoSQL is a challenging area. In this paper, improvements in the availability of NoSQL databases is presented based on number of replications and reducing the delay time for retrieving the data by better specification of the finger table entry. It is clear that availability is directly proportional to the number of replicas. So, replication is the obvious way to improve availability in NoSQL. Moreover, the paper also presents another way of reducing latency in retrieving the desired data to improve availability. In this case, a 2 - steps procedure for finger table specification is proposed to reduce duplicated entry in finger table and to provide efficient lookup for the data.

References

[1] A Carter, " The CAP Theorem as it Applies to Contemporary NoSQL Storage Systems", 2011.

[2] D Carstoiu, E Lepadatu, M Gaspar, " Hbase – non SQL Database, Performance Evaluation"doi: 10.4156/ ijact.vol2.issue5.4, 2009.

[3] F Chang, J Dean, S Ghemawat, W.C . Hsieh, D A. Wallach, M Burrows, T Chandra, A Fikes, R E Gruber, "Bigtable: A Distributed Storage System for Structured Data" , 2006.

[4] G. DeCandia, D Hastorun, M Jampani, G Kakulapati, A Lakshman, A Pilchin, S Sivasubramanian, P Vosshall, "Dynamo: Amazon's Highly Available Key-value Store", ACM, 2007.

[5] D. Gala "Cassandra (DHT)", 2010.

[6] H Gokavarapu "Exploring Cassandra and HBase with BigTable Model", Indiana University Bloomington.

[7] G. Harrison, "Consistency Models in Non-Relational Databases", 2009.

[8] A. Kumar, K Rugg, "Brewer's Conjecture and a characterization of the limits, and relationships between Consistency, Availability and Partition Tolerance in a distributed service" , 2011.

[9] D Karger, E Lehman, T Leighton, M Levine, D Lewin, R. Panigrahy " Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web" 1995.

[10] D. Kalia, P. Ramachandran, Y. Li, S. Chaudhary , "Implementation of chord P2P protocol using bidirectional finger tables", 2010.

[11] N. Leavitt, "Will NoSQL Databases live up to their promise", Technology News, IEEE Computer Society, 2010.

[12] T.J Teorry, W. T Ng "Dependability and Performance Measures for the Database Practitioner, IEEE Computer Society, 1998.

[13] D. Bruhn, " Comparison of Distribution Technologies in Different NoSQL Database Systems", 2011.

[14] J. Bohman, J. Hilding "Cassandra", 2010.

[15] K. Oren, " Analysis and Classi_cation of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer". A Dissertation for the fulfillment of master of science, 2010.

[16] M.A.Olson, K. Bostic, M. Seltzer, Sleepycat Software Inc. ,1999.

[17] C. Strauch, " NoSQL Databases", Lecture, Selected Topics on Software Technology, 2011.

[18] G Shroff, " Enterprise Cloud Computing, Technology, Architecture and Applications", Cambridge University Press, ISBN 978-0-521-13735-5 , 2010.

[19] Y. Shiraishi, E M.M Swe "SaaS and Cloud Computing", Web and Cloud, Information and Communication Training Institute, Union of Myanmar, 2011.

[20] S. Tiwari, "Professional NoSQL", John Wiley & Son, Inc. ISBN: 978-0-470-94224-6, 2011.

[21] R. Steinmetz, K Wehrle (Eds), "Peer to Peer Systems and Applications", ISBN 3-540-29192-X, Springer 2005.