A COMPREHENSIVE PERFORMANCE EVALUATION OF DATA STORES IN CONTEXT WITH APPLICATION MANAGEMENT TOOLS

SACHIN SHARMA

GyanVihar University

Received: 20-06-2012, **Revised:** 16-08-2012, **Accepted:** 28-09-2012, **Published online:** 22-11-2012

ABSTRACT

As of 2012, limits on the size of data sets that are feasible to process in a reasonable amount of time were on the order of exabytes of data.business informatics. The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s, as of 2012, every day 2.5 quintillion (2.5×10^{18}) bytes of data were created and it is required to analyze a vast amount of data. A number of companies have developed sophisticated
monitoring tools for data analysis. For specialized APIs, can monitor single invocations. To maximize the benefit of data monitoring, the data has to be stored for an extended period of time for ulterior analysis. This new wave of big data analytics imposes new challenges especially for the application performance monitoring systems. The monitoring data has to be stored in a system that can sustain the high data rates and at the same time enable an up-to-date view of the underlying infrastructure. With the advent of modern key-value stores, a variety of data storage systems have emerged that are built with a focus on scalability and high data rates as predominant in this monitoring use case. In this work, we present our experience and a comprehensive performance evaluation of six modern (open-source) data stores in the context of application performance monitoring.

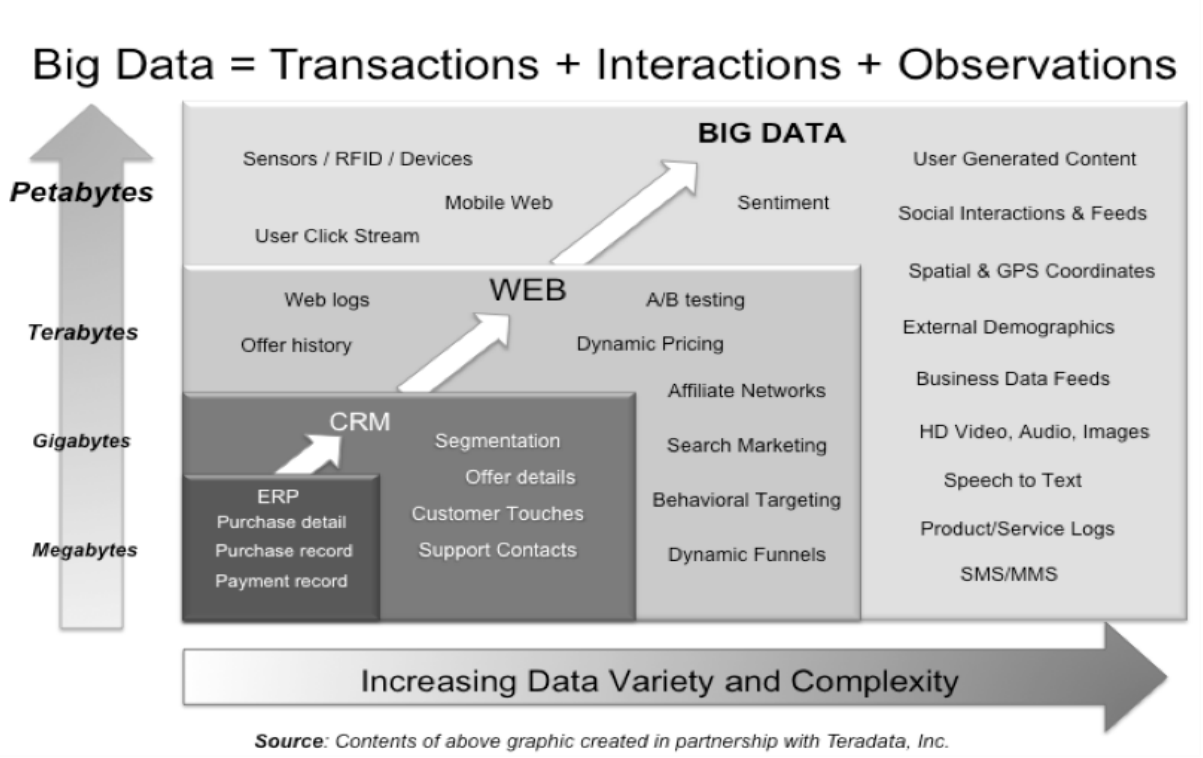
We evaluated these systems with data and workloads that can be found in application performance monitoring, as well as, on-line advertisement, power monitoring, and many other use cases. We present our insights not only as performance results but also as lessons learned and our experience relating to the setup and configuration complexity of these data stores in an industry setting.

1. INTRODUCTION

Enterprise systems today are very large and comprise complete data centers with thousands of servers. These systems are heterogeneous and have many interdependencies which make their administration a very complex task. To give administrators an on-line view of the system strength, monitoring frameworks have been developed. Common examples are Ganglia [4] and Nagios [5]. These are widely used in open-source projects and academia (e.g.Wikipedia). However, in industry settings, in presence of rigid response time and availability requirements, a more thorough view of the monitored system is needed. Application Performance Management (APM) tools, such as Dynatrace2, Quest PerformaSure3, AppDynamics4, and CA APM5 provide a more sophisticated view on the monitored system. These tools instrument the applications to retrieve information about the response times of specific services or combinations of services, as well as about failure rates, resource utilization, etc. Different monitoring targets such as the response time of a specific servlet or the CPU utilization of a host are usually referred to as metrics. In modern enterprise systems it is common to have thousands of different metrics that are reported from a single host machine. In order to allow for detailed on-line as well as off-line analysis of this data, it is persisted at a centralized store.

1.1 INTRODUCTION TO BIG DATA

In the starting, the data was in the form of files and soon emerged as database. But as the Tedd Codd's stone tablet inscribed in 1970 and began to gain commercial attention in 1980, the need for Big Data arise. The phrase Big Data refers to huge, frequently increasing and often formless datasets from varied sources, stored in remote server farms, and mined for significance. Many internet-based companies hold petabytes of data, Yahoo alone has over 25 petabytes [1]. In 2009, more than one petabyte (1015 bytes) of personal location-based data was generated across the globe [2]. In 2010, every continent on the planet produced in excess of 50 petabytes of data.



***Source**: Contents of above graphic created in partnership with Teradata, Inc.*

By some accounts, 90% of the recorded data in the world today has been created in the last two years - and the rate is accelerating. Big data is not only huge, but also diverse. Big data sources include environmental sensor networks, traffic monitoring systems, mobile phones, satellite imagery, video surveillance, posts to social media sites, and transaction records of online purchases, electronic health records, and satellite imagery, just to list a few important ones. If your organization is just like others, you are capturing and sharing more data from more sources than ever before. As a result, you are facing the challenge of managing high volume and high-velocity data streams quickly and analytically. Big Data is all about finding a needle of value in a deposit of unstructured information. Companies are now investing in solutions that infer consumer behavior, detect fraud, and even predict the future!

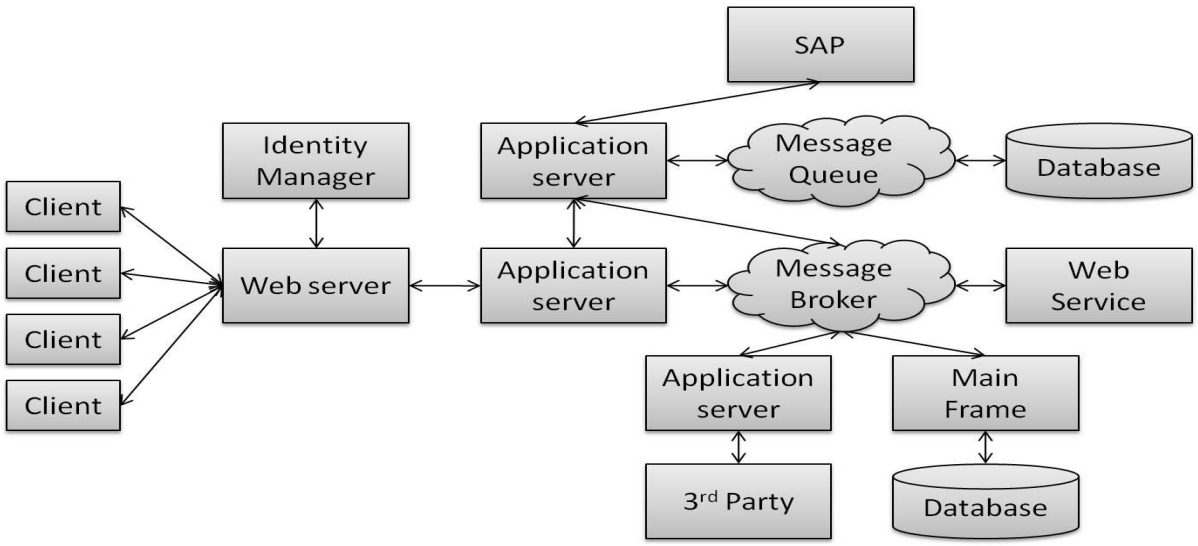
McKinsey released a report in May 2011 stating that leading companies are using big data analytics to gain competitive advantage. They predict a 60% margin increase for retail companies who are able to harvest the power of big data. To support these new analytics, IT strategies are mushrooming; the newest techniques include brute force assaults on massive information sources, and filtering data through specialized parallel processing and indexing mechanisms. The results are correlated across time and meaning, and often merged with traditional corporate data sources. New data discovery techniques include spectacular visualization tools and interactive semantic query experiences. Knowledge workers and data scientists sift through filtered data asking one unrelated explorative question after another. As these supporting technologies emerge from graduate research programs into the world of corporate IT, IT strategists, planners, and architects need to both understand them and ensure that they are

enterprise grade. Planning a Big Data architecture is not about understanding just what is different. It's also about how to integrate what's new to what you already have – from database-and-BI infrastructure to IT tools, and end user applications. Oracle's own product announcements in hardware, software, and new partnerships have been designed to change the economics around Big Data investments and the accessibility of solutions. The real industry challenge is not to think of Big Data as a specialized science project, but rather integrate it into mainstream IT.

1.2 APPLICATION PERFORMANCE MANAGEMENT

Application performance management refers to the monitoring and managing of enterprise software systems. An example of an enterprise system is shown in Figure above.

There are different ways to manage this kind of architecture. A common approach is the ARM standard [3]. In this approach every section has to implement the ARM API that is available for C and Java. Prominent ARM instrumented applications are the Apache HTTP server and IBM DB2. Although several common enterprise software systems are already ARM enabled it is often not feasible to implement the ARM API in existing systems. Another approach that is applicable for Java based systems is byte code instrumentation. This is enabled by the Java Virtual Machine Tool Interface that was specified in JSR-163 and introduced in J2SE 5.0. Its intention is to present an interface for profiling and debugging. Byte code instrumentation allows augmenting Java based software components with agents that have access to the state and the method invocations. The approach that is used in the CA APM products enables monitoring components, tracing transactions, and root cause analysis without changing the code base of the monitored system. The monitoring can be done on a very high level of detail. Current systems can generate millions of measurements per second where each data point may contain important information and therefore has to be stored safely. To fully understand the state of the monitored systems thousands of queries have to be processed on the measurements per second. Due to these enormous insert and query rates the underlying data store has to be highly scalable. To support the necessary insert rates current systems often store the measurement information in at least on disk and process them asynchronously and in batch. The introduction of an integrated data store would allow more sophisticated data analysis like trend analysis that current systems cannot offer.



1.3 APPLICATION REPONSE MANAGEMENT

ARM is an API jointly developed by an industry partnership that is used to monitor the availability and performance of applications. This monitoring is done from the perspective of the application itself, so it reflects those units of work that are important from the perspective of the business. The ARM standard is vendor-neutral and is targeted toward managing the performance of distributed applications.

2. LITERATURE REVIEW

In literature [1], the key points of Big Data are mentioned. The Big Data definition and its characteristics (variety, volume, and velocity) are explained in the work. The Hadoop File System is also explained. Hadoop is a scalable fault-tolerant file distributed system for data storage and processing. Core Hadoop is of two types: Hadoop File Distributed System (HFDS) and Map-Reduce: Fault Tolerant Distributed System.

In work [2], the need of switching to Big Data has given. The data is huge not in terms of size but it also contains a number of other variations as velocity, variety, etc.

It described [3], the various key data stores that we are requiring for the enterprise set up. The various workloads are also mentioned that we require for deducing the latency and throughput of the monitoring data. The APM tools are also mentioned here.

In the research work [4][5], the monitoring frameworks have been defined. The design, structure and framework of the Ganglia monitoring system and Nagois monitoring system has been discussed.

In [6][7][8], the benchmark key data stores are discussed. The key data stores are the database system with different properties and can be employed in enterprise system as per the industry requirements.

In [9], the scalability features of the key data stores are discussed. The workloads are also mentioned with different specification (read, write and scan).

In work [10], the performance evaluation techniques are mentioned. The ranges are defined for various key stores. Based on these ranges, the key stores are evaluated for latency and throughput.

In research [11][12], the apache file system (HBase, Voltemort, VoltDB, Redis, Cassandra, and MySQL) are discussed for various characterstics.

In [13], google file system is discussed. In 2004, Google published a paper on a process called MapReduce that used such an architecture. With MapReduce, queries are split and distributed across parallel nodes and processed in parallel (the Map step). The results are then gathered and delivered (the Reduce step). Google was incredibly successful, so others wanted to copy the process. MapReduce was transformed from a framework that only Google owned to an Apache open source project named Hadoop.

3. PROBLEM STATEMENT

We are awash in a flood of data today. In a broad range of application areas, data is being collected at unprecedented scale. Decisions that previously were based on guesswork, or on thoroughly constructed models of reality, can now be made based on the data itself by analyzing the data. Big Data analysis is required in every aspect of our modern society, including mobile services, retail industry, manufacturing, financial services, life sciences, and physical sciences. Usually enterprise systems are highly distributed and heterogeneous. They comprise a multitude of applications that are often interrelated. Clients connect to a frontend, which can be a Web server or a client application. A single client interaction may start a transaction that can span over more than a thousand components, which can be hosted on an equal number of physical machines [12]. Nevertheless, response time is critical in most situations. For example, form Web page loads the consumer expectation is constantly decreasing and is already as low as 50 ms to 2 s [3]. In a highly distributed system, it is difficult to determine the root cause of performance deterioration especially since it is often not tied to a single component, but to a specific interaction of components. System components themselves are highly heterogeneous due to the constant changes in application software and hardware. There is no unified code base and often access to the entire source code is not possible. Thus, an in depth analysis of the components or the integration of a profiling infrastructure is not possible. APM has similar requirements to current Web-based information systems such as weaker consistency requirements, geographical distribution, and asynchronous processing. Furthermore, the amount of data generated by monitoring applications can be enormous. Consider a common customer scenario: The customer's data center has 10K nodes, in which each node can report up to 50K metrics with an average of 10K metrics. As mentioned above, the high

number of metrics result from the need for a high-degree of detail in monitoring, an individual metric for response time, failure rate, resource utilization, etc. of each system component can be reported. In the example above, with a modest monitoring interval of 10 seconds, 10 million individual measurements are reported per second. Even though a single measurement is small in size, below 100 bytes, the mass of measurements poses similar big data challenges as those found in Web information system applications such as on-line advertisement [6] or on-line analytics for socialWeb data [7]. These applications use modern storage systems with focus on scalability as opposed to relational database systems with a strong focus on consistency. Because of the similarity of APM storage requirements to the requirements of Web information system applications, obvious candidates for new APM storage systems are key-value stores and their derivatives. Therefore, we present a performance evaluation of different key-value stores and related systems for APM storage. Specifically, we present our benchmarking effort on open source key-value stores and their close competitors. We compare the throughput of Apache Cassandra, Apache HBase, Project Voldemort, Redis, VoltDB, and a MySQL Cluster. Although, there would have been other candidates for the performance comparison, these systems cover a broad area of modern storage architectures. In contrast to previous work [8, 9, 10], we present details on the maximum sustainable throughput of each system. We test the systems in two different hardware setups:

- a memory
- a disk-bound setup.

Our contributions are threefold:

- We present the use case and big data challenge of application performance management and specify its data and workload requirements.
- We present an up to date performance comparison of six different data store architectures on two differently structured compute clusters.
- We report on details of our experiences with these systems from an industry perspective

4. OBJECTIVE

The study aims at the performance evaluation of the key stores in context with application performance management (APM). We evaluate these systems with varying workload, deduce and then check for their performances. The key stores are basically open source. We can also evaluate the configuration complexity while setting up these data stores in industry standard. APM refers to the monitoring and managing of enterprise software systems. There are two common approaches to monitor enterprise systems:

- An API-based approach, which provides a programming interface and a library that has to be utilized by all monitored components.
- a black-box approach, which instruments the underlying system components or virtual machines to obtain information about the monitored system.

The first approach gives a high degree of freedom to the programmer on how to utilize the monitoring toolbox.

5. BENCHMARK KEY STORES AND SET UP

We choose six key value stores to to get an overview of the performance impact of different storage architectures and design decisions. Our goal was not only to get a pure performance comparison but also a broad overview of available solutions. The key stores are as:

5.1 HBase

HBase [11] is an open source, distributed, column-oriented database system based on Google's BigTable [5]. HBase is written in Java, runs on top of Apache Hadoop and Apache ZooKeeper and uses the Hadoop Distributed Filesystem (HDFS) [1] (also an open source implementation of Google's file system GFS [5]) in order to provide fault-tolerance and replication.

Specifically, it provides linear and modular scalability, strictly consistent data access, automatic and configurable sharding of data. Tables in HBase can be accessed through an API as well as serve as the input and output for MapReduce jobs run in Hadoop. In short, applications store data into tables which consist of rows and column families containing columns.

5.2 Voldemart

Project Voldemort [14] is a distributed key-value store (developed by LinkedIn) that provides highly scalable storage system. With a simpler design compared to a relational database, Voldemort neither tries to support general relational model nor to guarantee full ACID properties, instead it simply offers a distributed fault-tolerant, persistent hash table. In Voldemort, data is automatically replicated and partitioned across nodes such that each node is responsible for only a subset of data independent from all other nodes. This data model eliminates the central point of failure or the need for central coordination and allows cluster expansion without rebalancing all data, which ultimately allow horizontal scaling of Voldemort. Through simple API, data placement and replication can easily be tuned to accommodate a wide range of application domains. For instance, to add persistence, Voldemort can use different storage systems such as embedded databases (e.g., BerkeleyDB) or standalone relational data stores (e.g., MySQL). Other notable features of Voldemort are in-memory caching coupled with storage system so a separate caching tier is no longer required and multi-version data model for improved data availability in case of system failure.

5.3 Cassandra

Apache Cassandra is a second generation distributed key value store developed at Facebook. It was designed to handle very large amounts of data spread out across many commodity servers while providing a highly available service without single point of failure allowing replication even across multiple data centers as well as for choosing between synchronous or asynchronous replication for each update. Also, its elasticity allows read and write throughput, both increasing linearly as new machines are added, with no downtime or interruption to applications. In short, its architecture is a mixture of Google's BigTable [5] and Amazon's Dynamo [8]. As in Amazon's Dynamo, every node in the cluster has the same role, so there is no single point of failure as there is in the case of HBase. The data model provides a structured key-value store where columns are added only to specified keys, so different keys can have different number of columns in any given family as in HBase. The main differences between Cassandra and HBase are columns that can be grouped into column families in a nested way and consistency requirements that can be specified at query time. Moreover, whereas Cassandra is a write-oriented system, HBase was designed to get high performance for intensive read workloads.

5.4 VoltDB

VoltDB [15] is an ACID compliant relational in-memory database system derived from the research prototype H-Store]. It has a shared nothing architecture and is designed to run on a multi-node 6Jedis, a Java client for Redis cluster by dividing the database into disjoint partitions by making each node the unique owner and responsible for a subset of the partitions. The unit of transaction is a stored procedure which is Java interspersed with SQL. Forcing stored procedures as the unit of transaction and executing them at the partition containing the necessary data makes it possible to eliminate round trip messaging between SQL statements. The statements are executed serially and in a single threaded manner without any locking or latching. The data is in-memory, hence, if it is local to a node a stored procedure can execute without any I/O or network access, providing very high throughput for transactional workloads. Furthermore, VoltDB supports multi-partition transactions, which require data from more than one partition and are therefore more expensive to execute. Multi-partition transactions can completely be avoided if the database is cleanly partitionable.

5.5 My SQL

MySQL is the world's most used relational database system with full SQL support and ACID properties. MySQL supports two main storage engines: MyISAM (for managing non-transactional tables) and InnoDB (for providing standard transactional support). In addition, MySQL delivers an in-memory storage abstraction for temporary or non-persistent data. Furthermore, the MySQL cluster edition is a distributed, multi-master database with no single point of failure. In MySQL cluster, tables are automatically sharded across a pool of low-cost commodity nodes, enabling the database to scale horizontally to serve read and write-intensive workloads. For our benchmarking we used MySQL v5.5.17 and InnoDB as the storage engine. Although MySQL cluster already provides shared-nothing distribution capabilities, instead we spread independent single-node servers on each node. Thus, we were able to use the already implemented RDBMS YCSB client which connects to the databases using JDBC and shards the data using a consistent hashing algorithm. For the storage of the data, a single table with a column for each value was used.

5.6 Redis

Redis is an in-memory, key-value data store with the data durability option. Redis data model supports strings, hashes, lists, sets, and sorted sets. Although Redis is designed for in-memory data, depending on the use case, data can be (semi-) persisted either by taking snapshot of the data and dumping it on disk periodically or by maintaining an append-only log of all operations. Furthermore, Redis can be replicated using a master-slave architecture. Specifically, Redis supports relaxed form of master-slave replication, in which data from any master can be replicated to any number of slaves while a slave may acts as a master to other slaves allowing Redis to model a single-rooted replication tree. Moreover, Redis replication is non-blocking on both the master and slave, which means that the master can continue serving queries when one or more slaves are synchronizing and slaves can answer queries using the old version of the data during the synchronization. This replication model allows for having multiple slaves to answer read-only queries resulting in highly scalable architecture. For our benchmark, we used version 2.4.2.

6. METHODOLOGY

We are showing the workload specification in this part of our work. Although the experimental results are not only the workload specification are mentioned in the synopsis. We defined five different workloads. They are shown in Table below. As mentioned above, APM data is append only, which is why we only included insert, read, and scan operations. Since not all tested stores support scans, we defined workloads with (RS,RSW) and without scans (R,RW,W). As explained above, APM systems exhibit a write to read ratio of 100:1 or more as defined in workloads. However, to give a more complete view on the systems under test, we defined workloads that vary the write to read ratio. Workload R and RS are read-intensive where 50% of the read accesses in RS are scans. Workload R W and RSW have an equal ratio of reads and writes. These workloads are commonly considered write-heavy in other environments. We perform these experiments for deducing the throughput and latency of the key data store.

Workload	% Read	% Scans	% Inserts
R	95	0	5
RW	50	0	50
W	1	0	99
RS	47	47	6
RSW 2	25	25	50

7. DETAIL OF RESEARCH WORK

We present the challenges that incur while storing monitoring data generated by application performance management tools. By comparing the performance of various key stores base on the workloads we deduce the required set up. Experiments are not shown only the methology has mentioned.

REFERENCES

[1] Python,Python programming language, http://www.Python.org/ (Online. Accessed 24 August 2012)

[2] Partick,Ohlinger, Wal-Mart's Data Warehouse, Vienna University of Technology,2006.

[3] Dnaiel.J. Abadi. Data Management in the Cloud:Limitations and Opportunities. IEEE Data Eng. Bull, 32(1):3–12, 2009.

[4] Fay, Chang.et al. Bigtable: A distributed storage system for structured data. In OSDI, 2006.

[5] Giueppe, DeCandia. et al. Dynamo: Amazon's Highly Available Key-value Store. In SOSP, page 220, 2007.

[6] Avinash, Lakshman. Cassandra-A Decentralized Structured Storage system. In LADIS, 2009.

[7] Memcached, http://code.google.com/p/memcached/wiki/NewOverview (Online. Accessed 24 August 2012)

[8] Loannis,Konstantinou , Distributed Indexing of Web Scale Datasets for the Cloud,MDAC'10, 2010

[9] An,Mingyuan.Using Index in the MapReduce Framework ,IEEE 10.1109, 2010.

[10] Vinay,Sudhakaran.Programming Abstractions for Dynamic, Distributed,Data-intensive computing , Msc dissertation 2011.

[11] Omkar,kulkal. Benchmarking an Amadaho-balanced cluster for Data Intensive Computing, Msc dissertation 2011.

[12] Jeff, Terrace. Object storage on CRAQ: High-throughput chain replication for read-mostly workloads. In Proc. USENIX Annual Technical Conference, June 2009.

[13] Robbert,van. Renesse. Chain replication for supporting high throughput and availability. In Proc. 6th USENIX OSDI, Dec. 2004.

[14] Add data replication to memcached. http://repcached.lab.klab.org/. (Online. Accessed 24 August 2012)

[15] Ion,Stoica.Chord: A scalable peer-to-peer lookup service for Internet applications. In Proc. ACM SIGCOMM, Aug. 2001.

[16] L.Avinash,M.Prashant,Cassandra - A Decentralized Structured Storage System,ACIM,2009.

[17] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C.Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, Bigtable: A distributed storage system for structured data, ACM Trans.Comput. Syst. (2008), no. 2.

[18] MongoDB, http://www.mongodb.org/.(Online. Accessed 24 August 2012).

[19] Project Voldemort, http://project-voldemort.com/.(Online. Accessed 24 August 2012).

[20] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears, Benchmarking cloud serving systems with ycsb, SoCC, 2010, pp. 143–154.

[21] NOSQL store, http://NOSQL-database.org/ (Online. Accessed 24 August 2012)

[22] C. J.Date. (2003). Introduction to Database Systems. 8th edition, Addison-Wesley. ISBN 0-321-19784-4.

[23] M.StoneBreaker.SQL databases V. NOSQL databases, Communications of the ACM, Vol. 53 No. 4, pp.10-11.

[24] A Conversation with Werner Vogels. http://queue.acm.org/detail.cfm?id=1142065, 2006.(Online access 17June 2012)

[25] Theo, Harder. Principles of transaction-oriented database recovery. Computing Surveys, 1983.