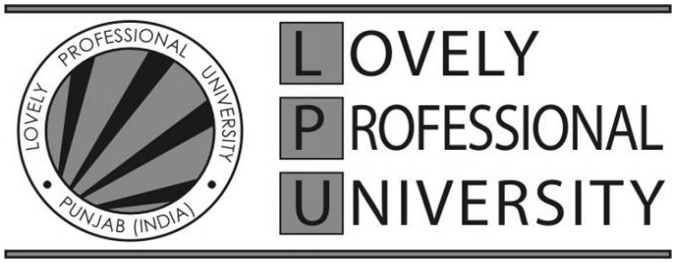

Performance Improvements in NoSQL Databases

INDICANO: An In-Memory Distributed Cache for NoSQL

A Dissertation

Submitted By

Akshay Sood

To

Department of Computer Science

In partial fulfilment of the Requirement for the

Award of the Degree of

Master of Technology in Computer Science

Under the guidance of

Mr. Ravinder Singh

(Assistant Professor)

(May 2015)

PAC Form

INDIA'S LARGEST UNIVERSITY

LOVELY PROFESSIONAL UNIVERSITY

Transforming Education, Transforming India

School of: Computer Sciene & Engineering.

DISSERTATION TOPIC APPROVAL PERFORMA

Name of the Student: AKSHAY SOOD Registration No: 11305847

Batch: K2305 2013-15 Roll No. B43

Session: 2014-15 Parent Section: K2305

Details of Supervisor: Designation: Asst. Professor

Name: Ravinder Singh Qualification: Mtech

U.ID: 17750 Research Experience: 1yr

SPECIALIZATION AREA: DATABASES (pick from list of provided specialization areas by DAA)

PROPOSED TOPICS

1. BIG DATA Efficient working of HDFS architecture and increasing System throughput
2. HANDLING UNSTRUCTURED DATA
3. DATABASE SECURITY

Signature of Supervisor 17750

PAC Remarks:

first topic approved

11/7

APPROVAL OF PAC CHAIRPERSON: 11/04 Signature: 30/9/14 Date:

*Supervisor should finally encircle one topic out of three proposed topics and put up for approval before Project Approval Committee (PAC)

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to Supervisor.

Abstract

The performance and efficiency of NoSQL is very critical in today's technology leading, cut throat competitive world. Every company today make extensive use of Big Data analysis to target their customers, better their offerings and sell their products.

There are basically two types of major problems being faced by current implementations of NoSQL for data management:

1. The data block access latency due to low bandwidth and io speed of physical disk.
2. The data block access latency due to the network travel of data while processing data.

In the dissertation, INDICANO library has been developed which provides three major functions:

1. Cache tier to NoSQL to reduce physical disk access and store data in cache that is repeatedly being used.
2. A coordinating node that keeps track of the data being cached in the cache tier and its location.
3. Distribution transparency to the cache tier.
4. A standard, universal interface to all the cache users.

The INDICANO library provides solution to some of the major issues being faced in field of databases and cache as well.

CERTIFICATE

This is to certify that Akshay Sood has completed M.Tech dissertation titled “Performance Improvements in NoSQL Databases, INDICANO: An In-Memory Distributed Cache for NoSQL” under my guidance and supervision. To the best of my knowledge, the present work is the result of his original investigation and study. No part of the dissertation proposal has ever been submitted for any other degree or diploma.

The dissertation is fit for the submission and the partial fulfilment of the conditions for the award of M.Tech Computer Science & Engineering.

Date:

Signature of Advisor

Name:

UID:

ACKNOWLEDGMENT

This report would not have been possible without the support of many people. Many thanks to my guide, Mr. Ravinder Singh, who guided me thoroughly, read my numerous revisions and helped make some sense out of the confusion. Also thanks to the Lovely Professional University, Department of CSE for allowing me to pursue my dissertation in the topic of my choice.

And finally, thanks to my classmates, parents, and numerous friends who endured this long process with me, always offering support and love.

DECLARATION

I hereby declare that the dissertation entitled “Performance Improvements in NoSQL Databases, INDICANO: An In-Memory Distributed Cache for NoSQL”, submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma. The work was done under the guidance of Assistant Prof. Ravinder Singh, at Lovely Professional University, Phagwara.

Akshay Sood

Date: (Reg. No. 11305847)

Table of Contents

List of Tables

List of Figures

CHAPTER 1
INTRODUCTION

In today's technology driven world, the push towards electronic storage of all type of data is continuously increasing. All of us are storing our commercial, financial, social, personal, and organizational and media data electronically in the cloud. Today, petabytes of data is flowing through the internet on daily basis. All this structured and unstructured data is known referred to as Big Data. Simply, any data source is referred to as Big Data if it has at least these three characteristics:

- Large Volumes of data
- High Velocity of data
- Wide Variety of data

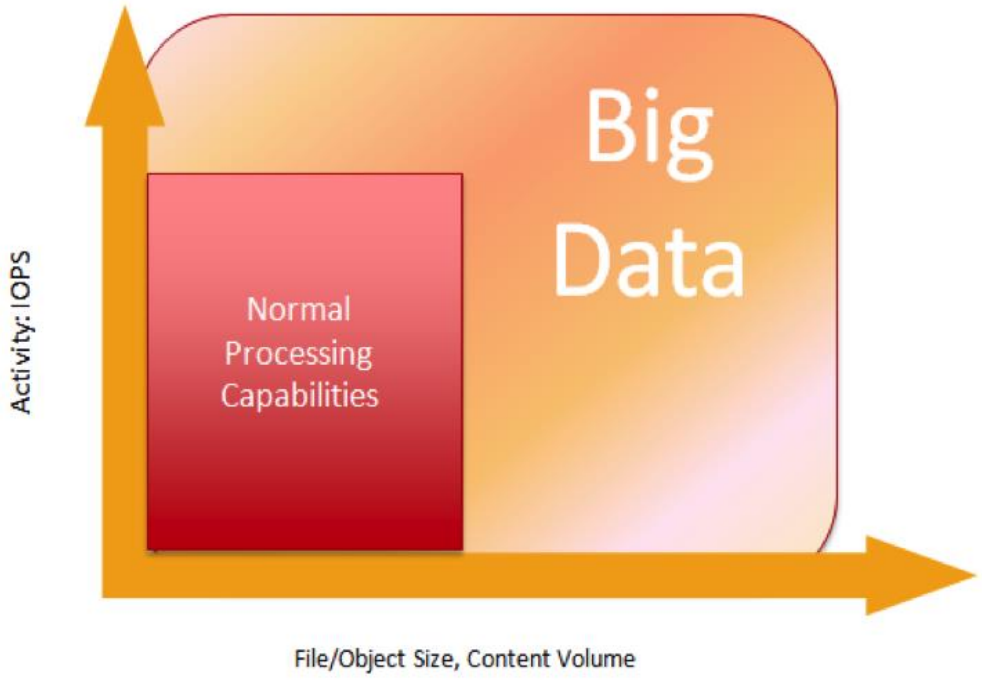

Figure 1: Big Data: A concept (EMC Corporation, 2011)

Big data is being generated everywhere, every moment, by every device or service. Big data includes but is not limited to:

i. Location data generated by mobiles, GPS devices etc.
ii. Web clicks
iii. E-Commerce
iv. Departmental and grocery stores purchases
v. Social media
vi. Financial transactions using net banking, credit and debit cards.
vii. RFID devices and tags
viii. Aeronautics: Airplanes, spaceships and missiles.

All this data being generated, collected and processed is put to use. Some of the industries which use big data are:

i. Financial Sector
ii. Healthcare industry
iii. Retail industry
iv. Internet based companies
v. Manufacturing industry
vi. Governments

Big data is composed of structured, semi structured, quasi structured and unstructured data.

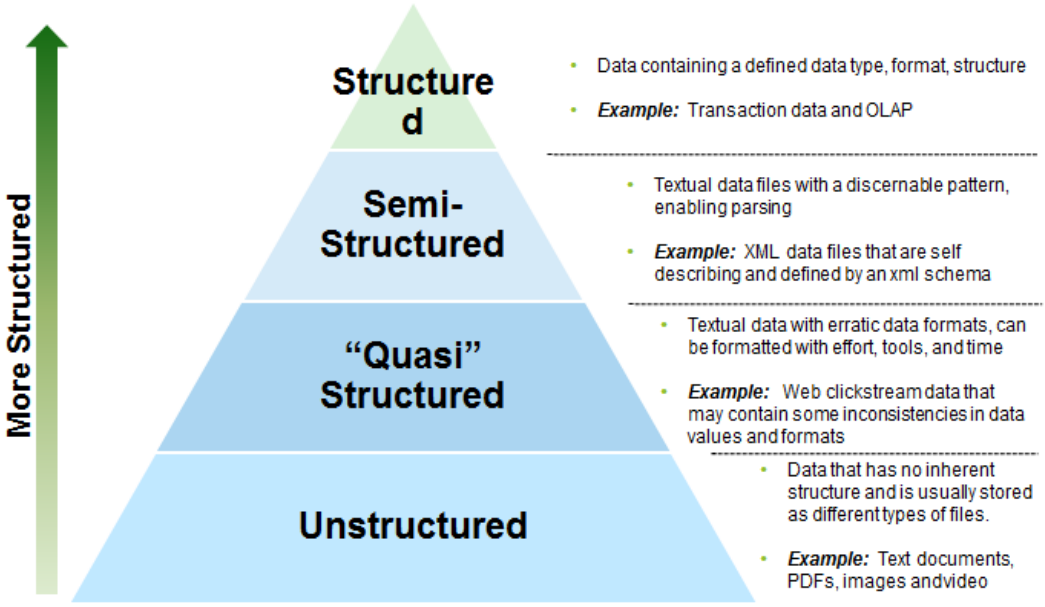

Figure 2: Big Data: Composition (EMC Corporation, 2011)

Big data is important because it enables organizations to gather, store, manage, and manipulate vast amounts data at the right speed, at the right time, to gain the right insights. But processing this large amount of data efficiently is still a challenge. There are many frameworks that offer Big Data processing and analytics.

One such framework is NoSQL databases, often called as Not Only SQL.

1.1 NoSQL Databases

NoSQL databases are Non-relational, distributed and horizontally scalable. Also, NoSQL databases are Schema-less which means no schema is needed to be defined in order to store data. This essentially means that with every bit of data being stored, the schema can be entirely different. These features make NoSQL perfectly suitable for handling semi-structured, quasi- structured and unstructured data.

There are currently 150 different NoSQL databases (NoSql Databases, n.d.). These are broadly classified into following classes:

i. **Wide Column Store / Column Families:** These store data tables as sections of columns of data rather than as rows of data. Physical tables are a collection of columns, each of them is a table with a single field.

ii. **Document Store:** These store and manage document oriented information.

iii. **Key Value/ Tuple Store:** These store key/value pairs in a persistent data store. Reads are done using the keys.

iv. **Graph Databases:** These use graphs with nodes, edges and their properties to represent and store data.

v. **Multimodel Databases:** These support multiple data models and use cases.

vi. **Object Databases:** In these, data are managed as objects, their attributes, methods and classes.

vii. **Multidimensional Databases:** Optimized specifically for OLAP.

viii. **Multivalue Databases:** Derived from multi-dimensional database. It has the support for attributes which can store a list of values.

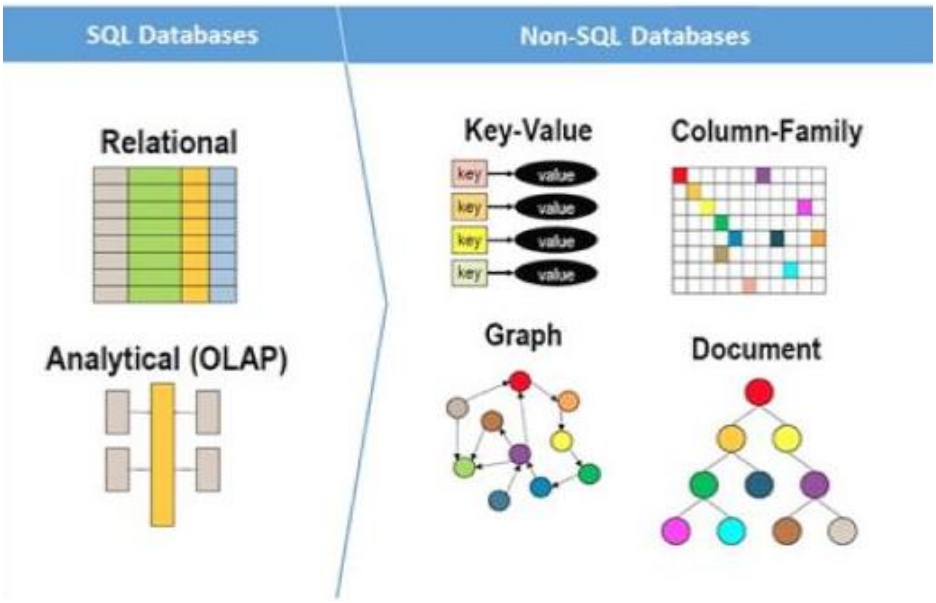

Figure 3: SQL vs NoSQL Structures (Flash Memory Summit 2014, 2014)

In practical applications, relational data does exist along with the unstructured data. In most of the cases, there is certain amount of structured that has to be handled. For example, transactions.

To handle this kind of relational data, the NoSQL falls short on it abilities. This is due to the fact that most of the NoSQL systems do not have the ability to perform joins in the queries. So the following techniques are often used to handle relational data in NoSQL database. (NoSQL - Wikipedia, the free encyclopedia, n.d.)

i. Fire Multiple Queries: Instead of the single query, one has to issue multiple queries to retrieve the required data.
ii. Redundant Data: Along with the foreign keys, store the most used data in the main table/document as well.
iii. Nest Data: Store all the related data in single collection so that single document contains all the required data.

The NoSQL databases offer many advantages over the traditional DBMS. Few of the advantages are:

i. Handles very large volumes of data including structured and unstructured data.
ii. Provides inexpensive scale-out architecture.
iii. Provides dynamic schema.
iv. Works well with the agile development process as the requirements keep on changing.
v. Provides native auto sharding capability to distribute data across servers.
vi. Supports replication to provide high availability and disaster recovery and prevention.

Due to the benefits of the NoSQL, today many big IT companies and industries use NoSQL. Few examples along with the NoSQL technology used are:

Table 1: Companies and their NoSQL implementations

Company	**NoSQL Technology**
Google	BIGTABLE
Facebook	CASSANDRA
Mozilla	HBASE
Adobe	HBASE
Twitter	Hadoop, Cassandra, PIG etc.
LinkedIn	Voldemort
Digg.com	Redis
Amazon	DynamoDB

1.2 Memcached

Memcached (memcached-a distributed memory object caching system, n.d.) is an in-memory caching mechanism built specially for increasing the performance of traditional databases. Memcached is essentially an in-memory key-value store. It stores the data in the memory as a value corresponding to a key. A key can be anything, a plain text or a hashed key or any random string.

Memcached provides API command set like set and get to cache a key value pair in memory and retrieve the data by using the same key respectively. The cache management and expiration is handled by memcached itself.

Although an expiration can be set along with the data being cached, memcached does not reclaim the memory on an active basis. What this means is, even when the cached data expires, it remains in the LRU list until it gets to the end of LRU queue. If a try is made to fetch the expired data, memcached will find the data and check that it has expired and will proceed to free its memory.

The general implementation of memcached is illustrated in the figure below:

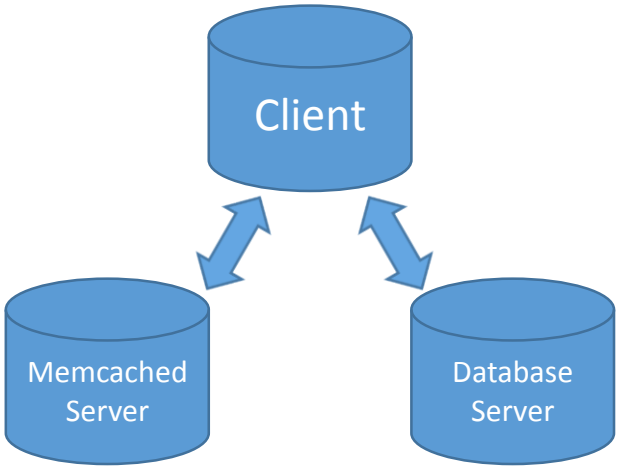

Figure 4: Memcached General implementation

Even though memcached is a robust caching solution, it is not without its fair share of limitations. Few of these limitations are:

i. Distribution Transparency: The memcached servers are standalone nodes. They are completely independent and unaware of the presence of other memcached nodes in the cluster. The distributed architecture has to be implemented by the client only.

ii. Serialization and Snapshots: Memcached has no serialization or snapshot mechanism. Which essentially means that cached data is not persistent. After a power outage or system reboot, the cache contents are lost.

1.3 PostgreSQL

PostgreSQL (PostgreSQL: The world's most advanced open source database, n.d.), first released in 1996, is basically an object-relational database. It is completely open source. It was raised from the Ingres Project by University of California, Berkely. (PostgreSQL - Wikipedia, the free encyclopedia, n.d.)

The PostgreSQL boasts of strict compliance with SQL standards. That is, it fully supports all the standard relational constructs. Apart from having standard SQL features, postgres also has the following features:

i. Cross Platform Compatibility: Postgres can be installed on all mainstream operating systems including but not limited to: Windows, Linux, Mac OS, BSD, Solaris, AIX.

ii. Support for Multiple Programming Languages: It has native support for over 12 programming languages including but not limited to: C/C++, Python. Java, Perl, Tcl, ODBC, .Net, PHP, LISP.

iii. Generalized Search Tree (GiST): It is an advanced indexing system which includes variety of algorithms for sorting and searching. These algorithms include but not limited to: B tree, B+ tree, ranked B+ trees, partial sum trees, R tree.

iv. Ability to work as a Structure less NoSQL Data Store: Postgres has support for JSON and XML fields. This makes it suitable to be used as a structure –less NoSQL database.

Apart from the above mentioned features, Postgres also has some enterprise grade features which makes it suitable to be used in large deployments to handle terabytes of data. Some of its enterprise features are:

i. Fault tolerance using write ahead logging.
ii. Query planner and optimizer
iii. Nested transactions
iv. Tablespaces
v. Point in time recovery
vi. Asynchronous replication
vii. Multi version Concurrency Control

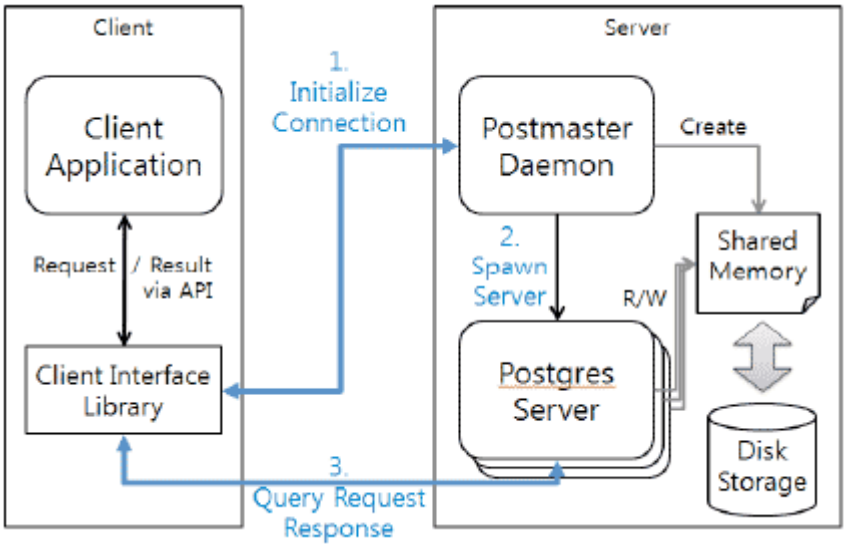

Figure 5: PostgreSQL: Working (PostgreSQL at a glance | CUBRID Blog, n.d.)

Due to all these features that postgres offers, it has gained many big names as its customers. Its clientele includes but not limited to:

i. Yahoo!: for analyzing behavior of its users.

ii. Sony: for its online multiplayer games
iii. Reddit
iv. Skype: for VoIP application databases.
v. ISS (International Space Station): to collect telemetry data in space.
vi. Instagram
vii. Disqus
viii. University of California
ix. University of New South Wales
x. University of Sydney
xi. Moscow State University
xii. National Physical Laboratory of India
xiii. Cisco
xiv. Red Hat
xv. SourceForge
xvi. Apple
xvii. IMDB.com
xviii. LHC: Large Hadron Collider Project by CERN

1.4 Today's Challenges

The NoSQL meets the requirements of massive data storage with variety buts falls short at data block access. The data block access includes latencies which arises due to lack of advancement in disk speed as compared to disk size and network speed. Due to which, the access performance of system under heavy and concurrent workload is negatively impacted.

Memcached is generally used in traditional RDBMS to improve the performance by reducing the read hits on the server disk, it has not been implemented widely in context of NoSQL databases.

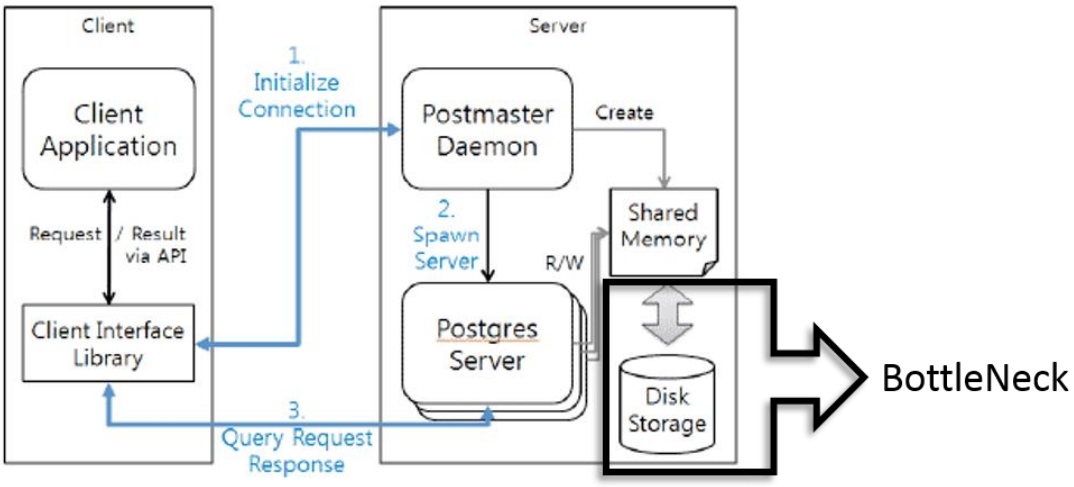

Figure 6: Disk Access Bottleneck

CHAPTER 2
LITERATURE SURVEY

Paper Citation: Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters" 19th Int'l Heterogeneity in Computing Workshop, Atlanta, Georgia (April) (Xie, et al., 2010)

In this paper, the authors take MapReduce into consideration in order to improve the performance of Hadoop. The authors point out that current Hadoop implementations makes assumption that all the nodes in the Hadoop cluster are homogeneous and most maps are data-local. Both of these assumptions does not hold true in today's highly virtualized environments. The authors show that neglecting the data locality issue in such heterogeneous can noticeably reduce the MapReduce performance. And address the problem of placing data so as each node has balanced data processing load.

In a heterogeneous cluster, after a high performance node completes processing of data at its local disk, it has to handle unprocessed data in remote slow node. The transfer overhead can be high if the amount of data is huge. Authors stress that one way of improving performance is by reducing amount of data travelling over the network, so a need for better data placement policies which can partition a large data set into data fragments and distributed across multiple nodes.

The other way is to store the replicas of the input data locally to the nodes so that the network travel time of data can be reduced. But this can potentially unlock many other problems like unnecessary need for large storing capacity for storing replicas, need for replica management system so as to co-ordinate and update replicas with each new

addition, modification or removal of data. Also, sending the replicas to number of nodes would create network congestion.

So, the authors have concentrated on the data placement techniques instead of the replica distribution and have proposed a data placement mechanism in HDFS to initially distribute a large data set to multiple nodes in as per the computing capacity of nodes. Two algorithms are used.

1. One is to initially distribute the fragments of the file to heterogeneous nodes.
2. Second is for re-organizing the file fragments to solve the data skew problem.

Using these data distribution policies, the MapReduce performance is increased and the job completion time is reduced noticeably.

Paper Citation: Chuncong Xu (2010) "Using Memcached to Promote Read Throughput in Massive Small-File Storage System" GCC, 2010 9th International Conference on Grid and Cloud Computing (Nov): Pages 24-29. (Xu, Huang, Wu, Xu, & Yang, 2010)

In this paper, the authors stress on the limitation imposed on distributed file systems due to the bottleneck offered by physical disk in IO operations. The authors emphasize that due to the LRU algorithm used in the memcached, the hot memory objects might be flushed when many short life objects are introduced in the system. So they propose to divide memcached into temporary and permanent cache. The temporary cache is further divided into parts with different priorities. This improves the system read performance by 2.65 to 8.05 times.

Paper Citation: Bogdan George Tudorica (2011) "A comparison between several NoSQL databases with comments and notes" Roedunet International Conference (RoEduNet), 2011 10th (Jun): Pages 1-5. (Tudorica & Bucur, 2011)

The authors try to compare various NoSQL databases and make comments on them. The authors note that even the SQL and NoSQL database have some common features, the do not behave the same way in similar situations. This essentially means that they can not replace each other for any particular problem.

Paper Citation: Jing Han (2011) "Survey on NoSQL database" 6th International Conference on Pervasive Computing and Applications (ICPCA) (Oct): Pages 363-366. (Han, E, Le, & Du, 2011)

In this paper, the authors emphasize on the need to move to the NoSQL databases due to the fact that traditional RDBMS falls short while handling big data as they are unable to fulfill the demand for high performance reads and writes. Also, the authors tells about the characteristics, properties and classifications of NoSQL databases. Also, they state that while choosing NoSQL, considerations should be made about the requirements of transactions, ACID properties and business model.

Paper Citation: Prof. Dr. Bernhard Plattner, Dr. Xenofontas Dimitropoulos, Dr. Patrick Stuedi and Dr. Patrick Droz (2012) "An In-Memory RDMA-Based Architecture for the Hadoop Distributed Filesystem" IBM Research Zurich (August). (Plattner, Dimitropoulos, Stuedi, & Droz, 2012)

In this paper, the authors show interest in real-time and low latency systems for cloud. The authors stress that for these, the data storage system should be designed as such to make

efficient use of datacenter's powerful hardware. So, the authors propose HDFS redesign for in-memory storage and remote direct memory access (RDMA). When the cluster memory can accommodate the data, the HDFS is made to operate in memory natively and operate on RDMA communication due to which, the system offers low latency and better cpu utilization. The system keeps only hot data in physical memory and operate on virtual memory.

This approach solves a number of problems. First, it improves the random read efficiency of Hadoop. Also, it might be viable to add file update feature in future which is easily supported by this implementation. Second, this implementation maintains the data locality as data is stored in the memory attached to the node, so it increases the performance of Hadoop in virtual clusters. Third, it also eliminates the bottlenecks traditionally experienced by Hadoop while interacting with file system.

Modifying the HDFS to perform in-memory requires modifications in every part of the architecture. The name node is modified to maintain a list of stags for every data node. Stags are used by client to write a new block. Also, for every block that is stored in cluster, name node keeps the information of data node on which the block is physically stored and also its exact memory location. Client use this information during reads. The data nodes are also modified. A component to manage memory and RDMA operations is required. Secondly, the data should be found in memory during a read or write of block. The slow storage devices are kept out of the critical path of flow of information. Also, the node should be able to allocate memory for future write operations and also provide a mapping of memory locations and stored blocks.

The data nodes inform the name node about their available stags and the memory locations of the stored blocks. The clients to name node communication include the stag information during all the read and write operations.

Paper Citation: Arcot Rajasekar, Hye-Chung Kum, Thomas Carsey, Howard Lander, and Sharlini Sankaran (2012) "DataBridge - A Sociometric System for Long-Tail Science Data Collections" NSF (September). (Arcot, Kum, Carsey, Lander, & Sankaran, 2012)

This paper concentrates on the management of long tail scientific and sociometric data by implementing algorithms and tools to enable data discoverability and reuse. Databridge is a collaboration between collaboration between University of North Carolina at Chapel Hill, Harvard University, and North Carolina A&T State University. This collaboration is funded by NSF and aims to develop an e-Science environment to provide measurements between different datasets. Here, the authors concentrate on the fundamental Big-Data problem that is, how to enable easier discoverability, and reuse of large number of small datasets that exists in isolation from each other? How to determine the relevancy of a data set to another thus making it easier to discover these relevant data?

Databridge is basically an indexing mechanism for scientific datasets. The authors point out that unlike a normal web search engine, the search space for scientific data sets is quite different and needs extra resources like tags, naming conventions, metadata and contexts to identify relevancy. But the typical long tail datasets in isolation provide very sparse information content for search and discovery.

Initially, the proposed system would draw upon Dataverse Network (DVN) and the Integrated Rule-Oriented Data System (iRODS) due to their rich set of real world structured data and metadata which will help validating the algorithms and analysis. But eventually it will gather information from multiple data resources maintained by individuals, projects, regional or disciplinary repositories, and national collaborations.

Paper Citation: Jing Zhang, Gongqing Wu, Xuegang Hu and Xindong Wu (2012) "A Distributed Cache for Hadoop Distributed File System in Real-time Cloud Services" ACM/IEEE, 13th International Conference on Grid Computing (September): Pages 12-21. (Zhang, Wu, Hu, & Wu, 2012)

In this paper, the authors aim to improve the data block access performance by implementing a 3 layered distributed cache system termed as HDCache. This cache service sits between client and HDFS name node and between name node and data nodes. It contains a client library and multiple cache services. The three layers of cache service are

1. an in-memory cache,
2. a snapshot of local disk and
3. the actual disk view as provided by HDFS.

This service completely bypass the direct HDFS access by the client and route all the block access requests through its own service. The cache services are managed using distributed has table in P2P fashion. The HDCache uses LRU cache replacement policies for Local cache replacement as well as purging disk contents.

The authors point out the limitations of using memcached as a cache service for Hadoop which acted as motivations of designing a completely new cache system. These disadvantages are:

1. The memcached system was traditionally designed for database data. It lacks a typical cloud storage system. It can efficiently handle SQL query results but is inappropriate for online personal data.
2. The Memcached does not have any local serialization or snapshot mechanism which leads to the problem of cached contents being lost after a server shut down or crash. It is very expensive to reconstruct the lost contents.

3. The Memcached servers are independent from each other. The distribution transparency is the function of client which leads to complex management issues.
4. The consistency checking mechanism of Memcached is very basic. It works on setting an expiration time on cache which could lead to network overload when large amount of cache is expired at once.

The disadvantages of using this system is that is adds an extra layer of abstraction in the system. The clients no longer communicate directly with HDFS. The clients have to connect to HDCache service using HDCache client library. If the HDCache service goes down, the whole HDFS goes down.

Paper Citation: Gurmeet Singh, Puneet Chandra and Rashid Tahir (2012) "A Dynamic Caching Mechanism for Hadoop using Memcached" University of Illinois at Urbana Champaign, Department of Computer Science. (Singh, Chandra, & Tahir, 2012)

In this paper, the authors target the disk access time and bandwidth as culprits for high access time to data blocks in HDFS. To eliminate the bottleneck introduced by the high disk access time, the authors propose the integration of a fetching and caching mechanism based on Memcached with Hadoop. Memcached is a distributed in-memory key-value caching system which is traditionally used for database data by caching memory objects in RAM and hence reducing number of times the database must be read.

In this case, the authors propose to register the block cached at data node into the hash table of the Memcached Server. Memcached will store each entry as a key-value pair where key will be the block id and value would be the data node-id where the block is cached. The authors propose to store the data node identifiers when multiple nodes cache same block to

provide locality optimization. 30% Ram at each node has been reserved to serve as cache. It is proposed to eventually let this value be determined empirically.

Some of the data nodes are used as dedicated Memcached servers which keeps logs of most recently cached blocks in the entire system. These servers also store the corresponding node where the block is cached. When a particular data block is required, two requests are generated, one to the Memcached servers which return the address of the node having the cached version of block in need and the other request is sent to the name node that returns the information about the replicas of the block.

Two different but simple greedy caching policies are combined to have one "Two-Level greedy caching" policy so as to felicitate the cached block to be reused by future block requests of the same block. LRU is used as the local cache replacement policy. However, a global cache eviction policy is also proposed to efficiently manage caching across all the three replicas of the data block.

Paper Citation: Wei Zhang, Sundaresan Rajasekaran and Timothy Wood (2013) "Big Data in the Background: Maximizing Productivity while Minimizing Virtual Machine Interference" ASBD (June). (Zhang, Rajasekaran, & Wood, 2013)

In this paper, the authors point out the problem of idle resources in Big Data processing in Virtualized environments. The authors stress that data centers are often lightly loaded. Which leaves us with spare capacity of CPU time, Ram, and disk IO, which is often left idle. The authors believe that this excess capacity can be used to perform meaningful work. In this paper, the authors analyze the level of work load currently experienced by data centers and also try to find out the methods to safely increase the data center utilization taking into consideration the workload fluctuations and dangers of interfering virtual

machines. The authors firmly believe that as many big data applications have a resource hungry and distributed nature, they are perfect fit for consuming this idle data center capacity by being deployed in virtual machines. Due to being readily designed for distributed workload, big data frameworks such as MapReduce will be able to spread the jobs throughout the datacenter.

But there are several challenges that prevent this sort of utilization by big data frameworks. One challenge is that due to these application's IO and CPU intensive nature, they can interfere with other applications. Other challenge is variability in the resource availability to the big data applications cause a decline in overall performance due to speculative job scheduling.

The authors have tried to run the test on XEN virtualization environment using TPC-W benchmark. These tests show that Xen is able to provide good performance for big data applications if virtual machines are provided with dedicated CPU's. But in case when the CPU's are shared, Xen's priority mechanism falls short.

Paper Citation: Jiri Schindler (2013) "Profiling and Analyzing the I/O Performance of NoSQL DBs" ACM SIGMETRICS'13 (June): Pages 389-390. (Schindler, 2013)

In this article, the author suggest that the modern NoSQL databases use I/O and other resources inefficiently. The author stresses that majority of clustered NoSQL systems rely on the operating system and file system services to manage the storage. They do not directly manage the storage. This implies that the local disk file system determines the performance of the database system.

The author stress that in general, the I/O patterns of the NoSQL databases does not differ from the traditional RDBMS. The difference is in their different programming style and data management techniques.

Paper Citation: Meenakshi Shrivastava, Dr. Hans-Peter Bischof (2013) "Hadoop-Collaborative Caching in Real Time HDFS" International Conference on Parallel and Distributed Processing Techniques and Applications (July). (Shrivastava & Bischof, 2013)

In this paper, the authors focus on reducing the MapReduce job's execution time by using a caching mechanism called collaborative caching for efficiently using resources and system. By using the proposed Hadoop-Collaborative caching mechanism, the authors aim to improve the performance, reduce access latency and increase the throughput. This new caching mechanism includes collaborative caching, reference caching and modified ARC algorithm. Cache is managed at every data node using a dedicated cache manager. The purpose of this cache manager is to manage caching, replacement, collaborative caching and eviction.

In this technique, the authors cached the local data as well as the information about data cached on remote nodes and serve that data as input to the MapReduce jobs. This lead to a new hierarchy layer Name Node cache, Data Node's Cache, Remote Data Node's Cache and the disk. A global cache is formed by taking together the caches of all the data nodes. Name Node co-ordinates the global cache, but allows the decisions of remote caching to be taken by Cache manager of respective nodes.

Reference caching technique is used to make the caching of data faster and easier. In this, references to meta file (referring to the checksum value of data) and block file (referring to

the actual data) are cached. This helps in faster checksum checks and helps caching data faster in memory as the data has not to be searched form the petabytes of data.

Cache replacement policy used here is Modified-ARC. According to the authors, it maximize cache hit ratio and improve efficiency. Here, the caches are divided into two sections cached objects and history objects. Cached section is further divided in Recent Cache, its Recent History and Frequent Cache and its frequent History. Cached objects are the actual cached data items and History objects are the history of evicted data items.

Paper Citation: Yishan Li (2013) "A performance comparison of SQL and NoSQL databases" IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) (Aug): Pages 15-19. (Li & Manoharan, 2013)

In this paper, the authors put the claim of NoSQL databases to perform better than traditional databases to test. The authors test the performance of operations like read, write, looping through records and deletes. The authors find that every NoSQL is not better performer than SQL databases. Moreover, the performance deflects with database to database and operation to operation.

Paper Citation: Raja Appuswamy, Christos Gkantsidis, Dushyanth Narayanan, Orion Hodson, and Antony Rowstron (2013) "Scale-up vs Scale-out for Hadoop: Time to rethink?" ACM, SOCC (13) (October): Article No.20. (Appuswamy, Gkantsidis, Narayanan, Hodson, & Rowstron, 2013)

This paper compares the application performance in case of single scale-up servers and scale-out distributed processing cluster for traditional analytical job processing for big data such as Hadoop. The authors suggest that while these systems were originally developed

for petabytes of data processing, in real world scenarios, the majority of jobs process less than 100GB. So, the authors claim that a single scale-up server would outperform a cluster in terms of ratios of performance, cost, power and server density. In short, they are trying to implicate that a single scale-up server could be better suited for real world Hadoop jobs in some scenarios and competitive in all cases.

The authors take a set of 11 representative Hadoop jobs and performs the evaluation on these. To achieve the desired performance, several modifications were done to the Hadoop runtime engine targeting scale-up configuration performance improvement. The authors claim these improvements to be transparent, not requiring any application code changes and not compromising scale-out performance. These simple optimizations remove bottlenecks and improve performance of both scale-up and scale-out configurations for sub tera-scale jobs.

The authors find that with the proposed improvements, a 32 cores scale-up server outperforms an 8 node, 32 cores scale-out configuration on 9 out of the 11 test jobs. Also. For the remaining 2 jobs, the performance ratio is not below 11% of the scale-out configuration. A larger cluster would obviously improve performance but at the cost of increase financial burden, power requirements and space usage. So, even while comparing to a larger cluster such as 16 node scale-out cluster, a scale-up server still provide better cost-to-performance ration for all the 11 jobs.

Book Citation: Ahmed Soliman (2013) “Getting Started with Memcached” Packt Publishing Ltd. (Soliman, 2013)

In this book, the author tell about installing and using memcached with major languages and frameworks. Also, the author tells about the memcached APIs, functions and objects and using it with relational databases.

Paper Citation: Hely Shah, Mohammed Husain Bohara (2014) "A Brief Introduction to Memcached with its Limitation" International Journal of Engineering Research & Technology (February). (Shah & Bohara, 2014)

In this paper, the authors introduce memcached, its architecture and implementation in its very basic is also discussed. The authors stress that the scalable architecture of memcached can be further enhanced by addressing some of its limitations related to its data structures and some due to the characteristics of memcached.

The authors point out the following limitations of memcached:

i. Non-persistent cache: After a server reboot, the cached data is lost. Rebuilding the cache is quite an intensive process.

ii. Limitations arised due to Scale-Out Nature: When new servers are added to memcached cluster, the keys are required to be remapped to different nodes and clients are also required to be updated with this remapping. Otherwise, client may query obsolete data from a node, resulting in unwanted hit on database server.

iii. Lack of Unified Monitoring: As the memcached nodes behave independent of each other, and are completely unaware of other memcached nodes, this makes it difficult to manage and monitor the memcached cluster.

CHAPTER 3
PRESENT WORK

3.1 Problem Formulation

Big data includes structured, semi structured, quasi structured and unstructured data. To store this data, NoSQL is used. In an ideal world, NoSQL DBMS deployments would only have to deal with semi, quasi or un-structured data that is, text files, documents, multimedia files, sensor data etc. However, this is not always the case in real world. Most of the deployments of databases, whether Relational, NoSQL or ORDBMS, has to deal with structured and relational data. Structured data, although less, but still make up an important part of big data. For example, financial transaction data (including e-commerce, net banking, credit or debit card details), Log records data etc. Now, structured or unstructured, any type of data has to be stored on the physical disks. The physical splatter and spindle disk has some physical limitations like rotations per minute and seek time, where physics and mechanics come into play. Although advances in disk IPOS have been made but the physical limitations will always be there. Today, the network and the processing speeds are far above the physical disk speed. So, at the time of fetching the data from the physical disk, there is a bottleneck. The bottleneck belonging to the physical characteristics of the disk. By implementing an inexpensive cache tier for the NoSQL database, the disk accesses can be reduced which in turn will provide lower read times.

Also, each node in the memcached, as it is, is a standalone server. The distribution transparency is implemented by the client. Different clients can have their own strategy to achieve distribution. Which might or might not be compatible with other clients accessing the same memcached servers. By implementing a library that creates and maintains a coordinating node that keeps the location of cached data in memory, a truly cross-

compatible distributed architecture can be obtained. Each client can then use the same library to interact with the memcached cluster.

Also, as, the servers in the datacenters usually are lightly loaded, they have some excess capacity which can be put to use. So, instead of wasting those resources, if some part of their memory is used for cache, it would provide efficient utilization of resources and save on the cost of implementing completely separate cluster for cache tier.

3.2 Objectives of the problem

The major objectives of the problem are:

i. Implementing a cache tier for NoSQL databases to reduce the number of disk access hence improving the overall read efficiency.

ii. Using the excess resources in servers at datacenter to create the cache cluster so as to save on the cost of completely separate cluster and provide efficient utilization of resources.

iii. Implementing a coordinator node for memcached so as to achieve true distribution transparency at cache tier level instead of at application level.

iv. Implementing a library for the cache tier that will provide the access to cache tier through the coordinator node and provide compatibility to different clients so that each client does not have to handle distribution on its own thereby preventing different distribution architectures that are incompatible with each other.

3.3 Research Methodology

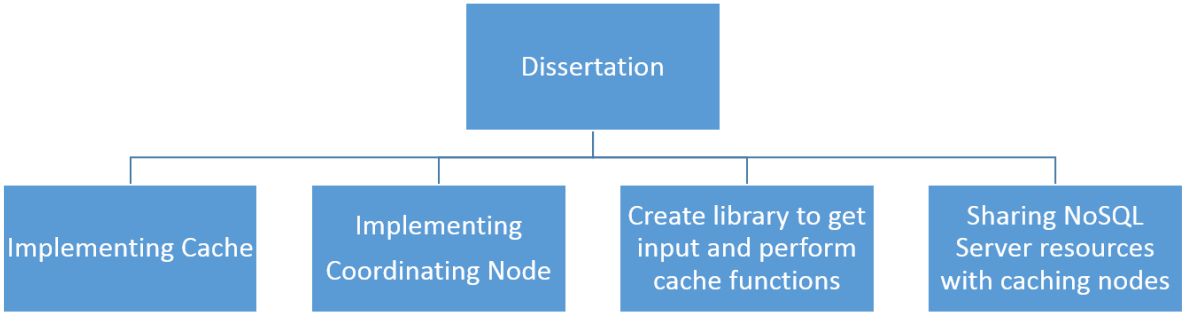

Figure 7: Parts of dissertation

The research work was divided into the following parts:

3.3.1 Implementing Cache

The first step was to implement a general caching solution. A general caching solution is a very simple deployment.

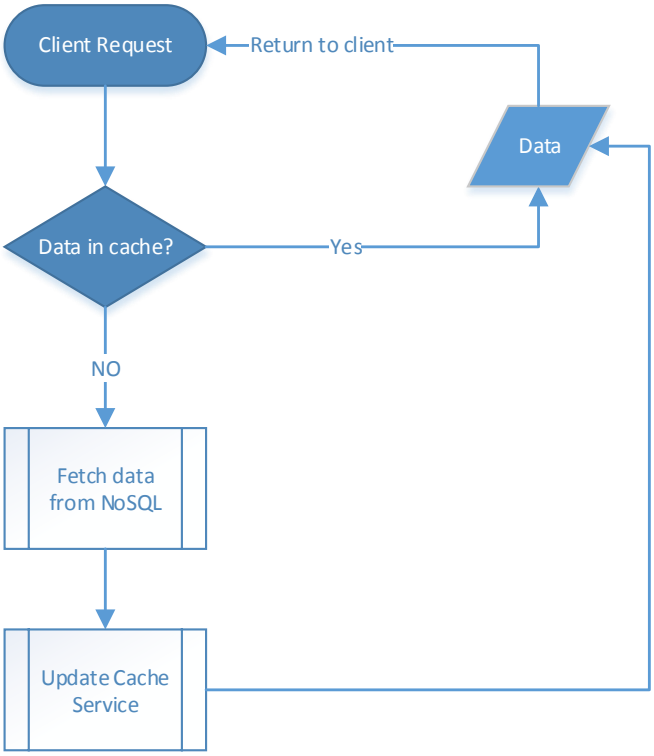

Figure 8: General Cache Implementation Flowchart

3.3.2 Implementing Coordinating Node

Next step was to include more number of nodes for cache. As, the number of nodes went plural, there was a need to implement a coordinating node which can coordinate all the caching nodes.

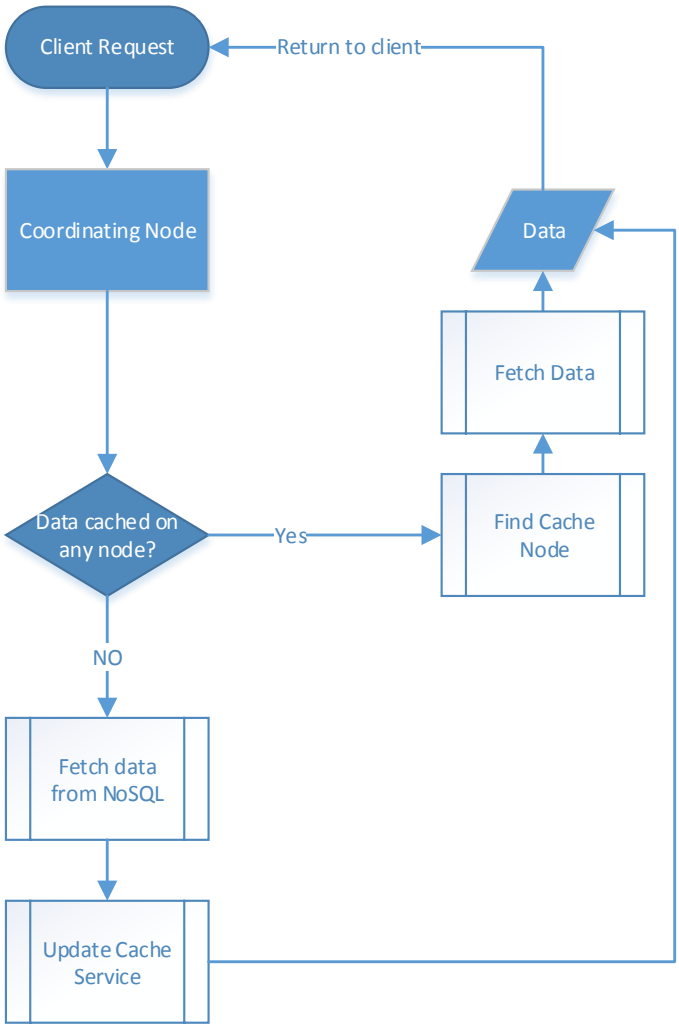

Figure 9: Coordinating Node in Cache Implementation Flowchart

3.3.3 Creating Library

After implementing the cache and the coordinating node, the next step was to implement a library which handles the following functions:

i. Act as an interface to the data retrieval requests

ii. Communicate with the coordinating node to find out the location of the cached data.

iii. Retrieve data from the caching node.

iv. Contact the NoSQL node if data not found in the cache.

v. Return the found data to the requesting client.

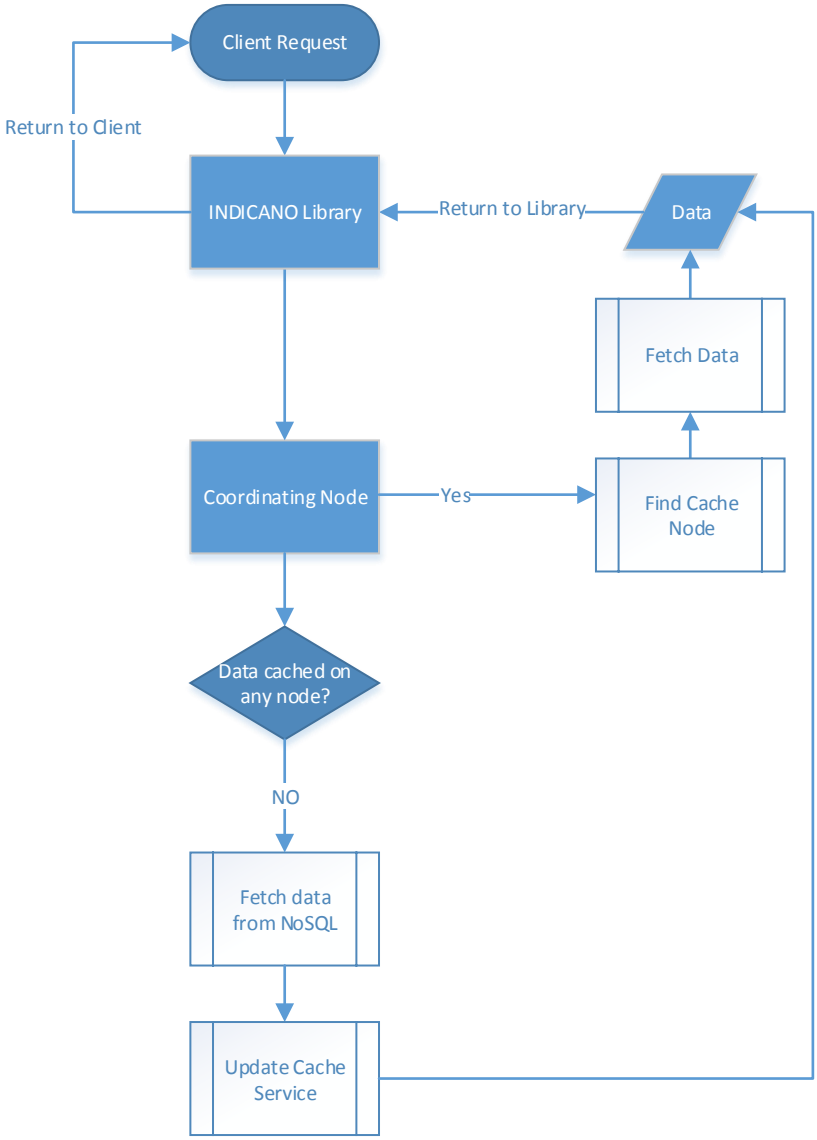

Figure 10: INDICANO Library handling the Cache Functions Flowchart

3.3.4 Resource Sharing

After properly implementing the library to handle the cache functions, next step was to implement a technique to make efficient utilization of resources. This was achieved by sharing the NoSql Nodes' memory for caching purposes as well.

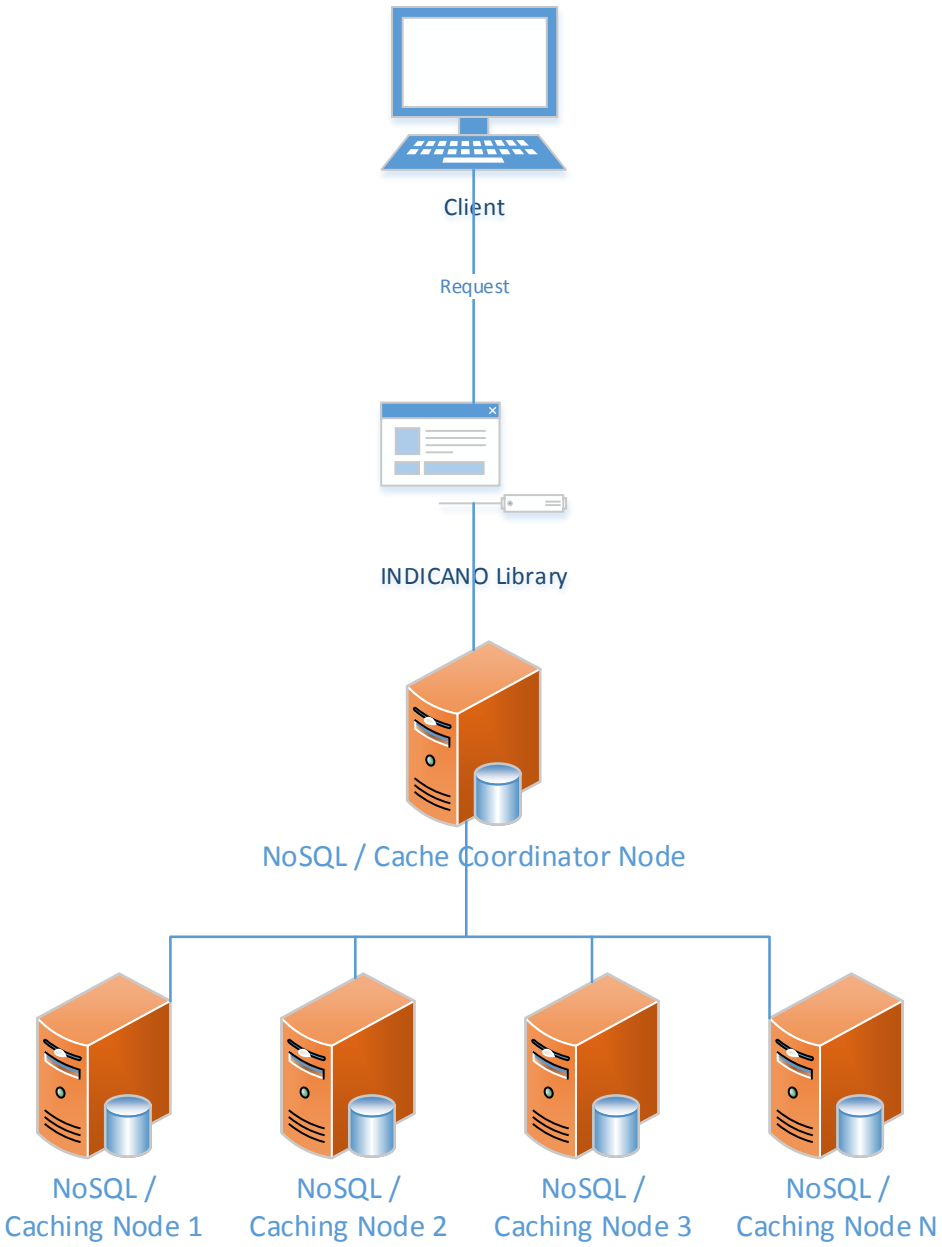

Figure 11: Resource Sharing Between NoSQL and Memcached

CHAPTER 4
RESULT AND DISCUSSION

4.1 Introduction to tool and technology used

4.1.1 Eclipse

Eclipse (Help - Eclipse Platform, n.d.) is a free and open source (Foss) integrated development environment. It is used extensively for development. Eclipse has an extensible architecture. Only the kernel of eclipse is not a plug in. Everything else in eclipse works as a plug in. This greatly increases the customizability of eclipse. (Eclipse - Wikipedia, the free encyclopedia, n.d.)

Eclipse provides support for many different technologies by using plug-ins. Some of the languages supported by eclipse are:

i. JAVA
ii. C / C++
iii. JavaScript
iv. Perl
v. Ruby
vi. Python
vii. Prolog
viii. PHP

4.1.1.1 **Installation of Eclipse** (Eclipsepedia, n.d.)

1. Download Eclipse IDE from https://www.eclipse.org/downloads/
2. Download java development kit (jdk)
3. Install JDK
4. Unzip Eclipse IDE Zip file

4.1.1.2 Run Eclipse

1. Open the previously extracted folder from the eclipse Zip file.

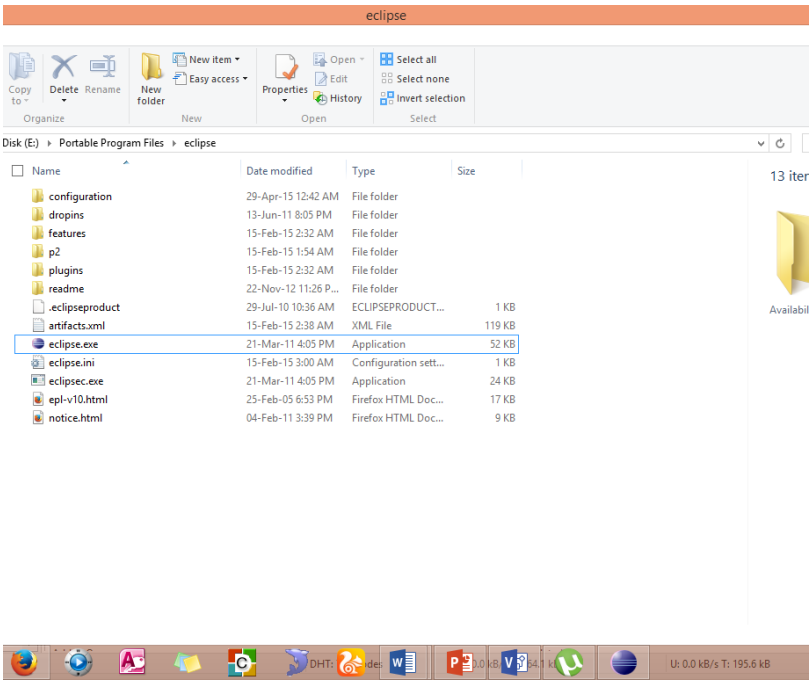

Figure 12: Eclipse: Extracted contents

2. Double click on “eclipse.exe” to run eclipse.

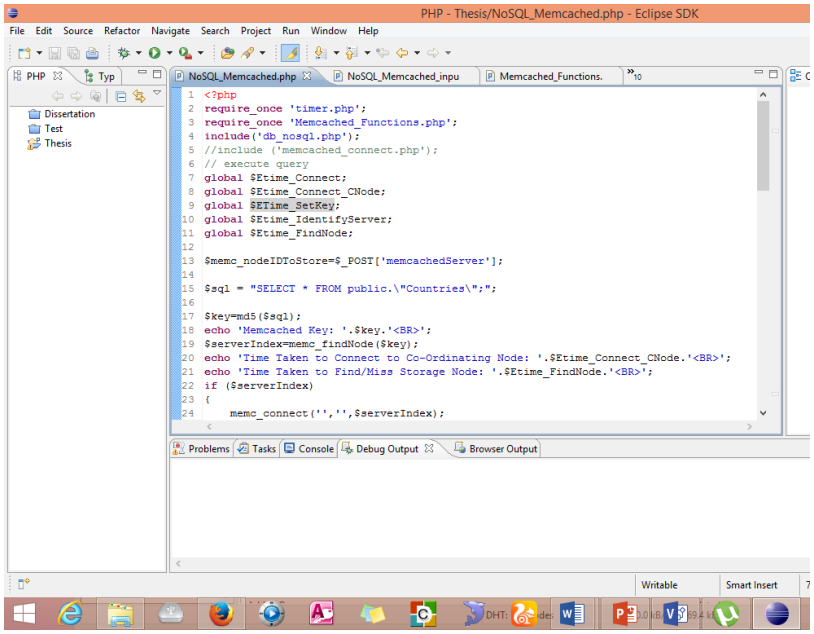

Figure 13: Eclipse Running

4.1.1.3 Add PHP plug-ins to eclipse (Help - Eclipse Platform, n.d.)

1. In eclipse, click on Help→Install New Software…

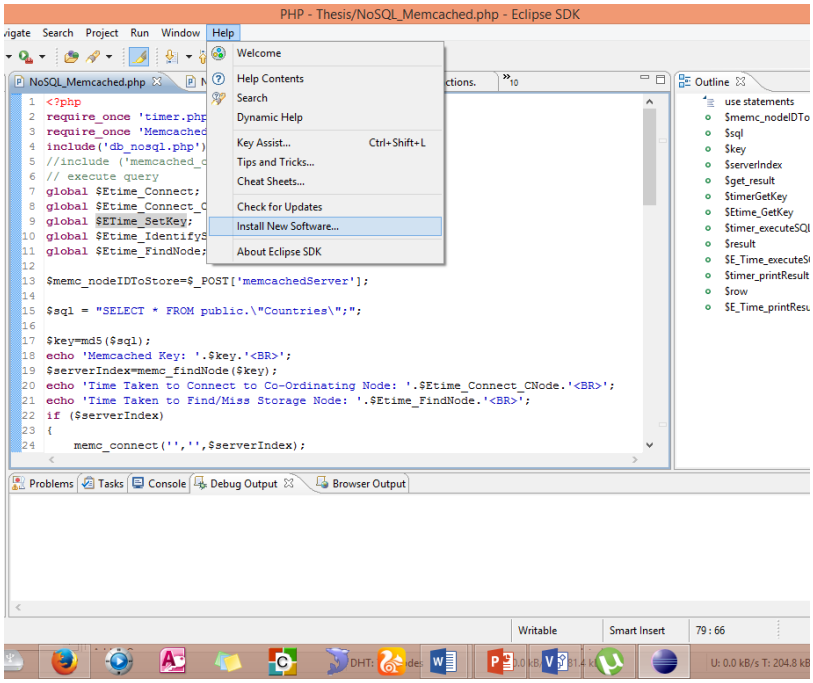

Figure 14: Eclipse: Installing PHP 1

2. In the Install window, set “Work with” to “all available sites”

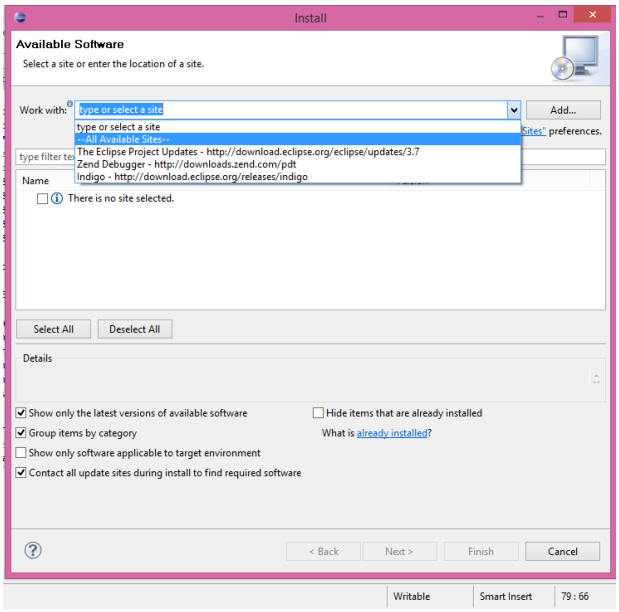

Figure 15: Eclipse: Installing PHP 2

3. The list will be loaded. Expand “Programming Languages” category and select “PHP Development Tools”

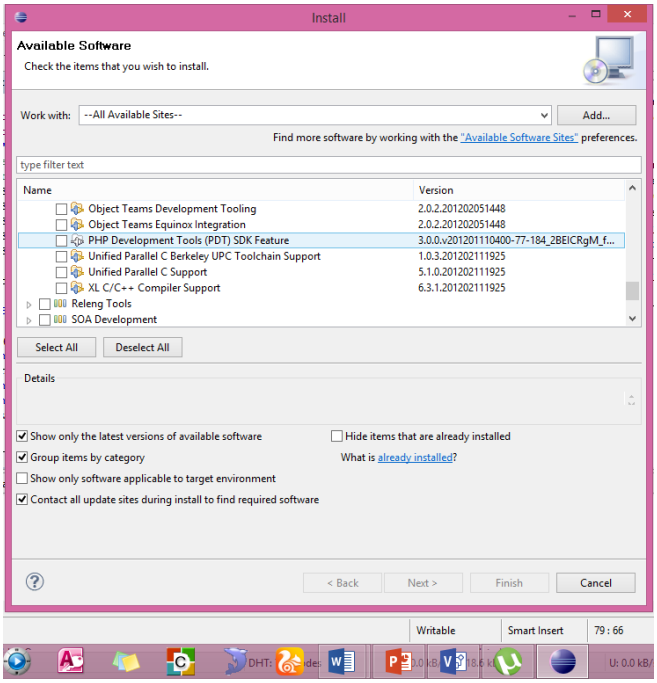

Figure 16: Eclipse: Installing PHP 2

4. Click Finish and the PHP plug-ins would be installed to eclipse.

4.1.1.4 Add PHP Project to Eclipse

1. In eclipse, click on File→New→Project

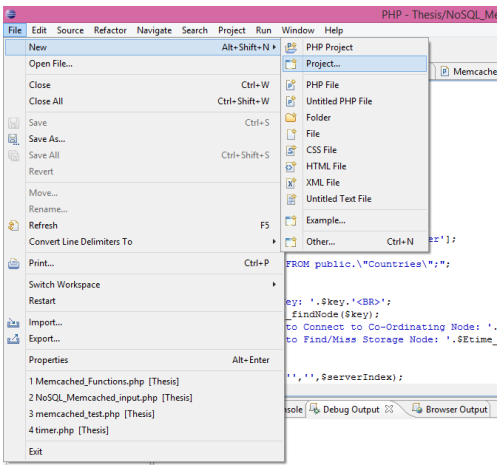

Figure 17: Eclipse: Add New PHP Project 1

2. In the New Project window, select category “PHP” then select “PHP Project”. Then click on Next.

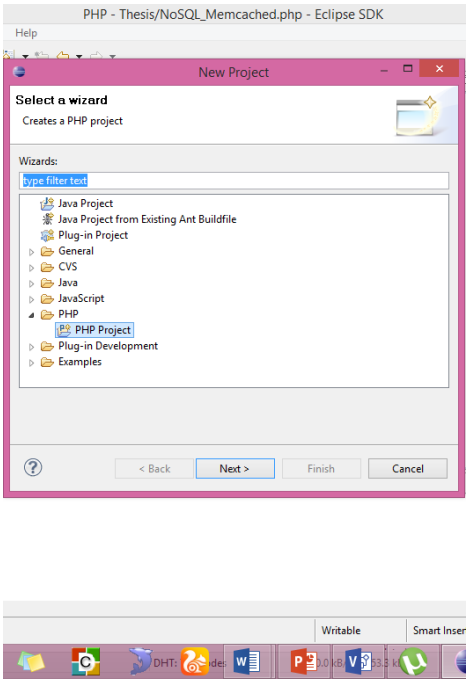

Figure 18: Eclipse: Add New PHP Project 2

3. In the "New PHP Project" window, give name to the project and select other settings as appropriate. Then click on "Finish".

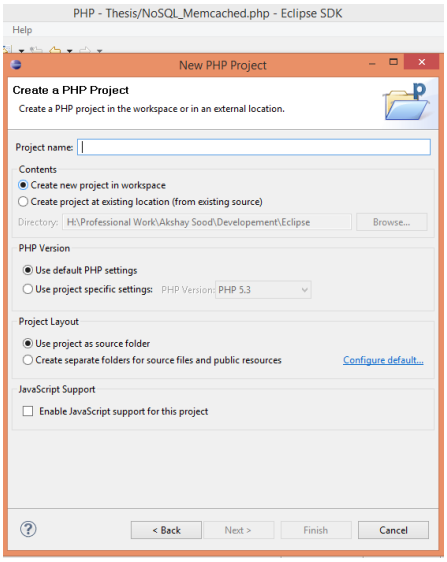

Figure 19: Eclipse: Add New PHP Project 3

4.1.2 PHP

PHP (PHP: What is PHP? - Manual, n.d.) stands for PHP: Hypertext Pre-Processor. It is a server side scripting language. PHP is a language generally used for web

development, but is also used for programming for other purposes as well. PHP is free and open source.

PHP works in conjunction with other web technologies like HTML, Java Script, CSS etc. PHP can be embedded in a HTML code. The PHP code is interpreted by PHP interpreter. The interpreter is usually deployed as a module on the web server.

4.1.2.1 **Working of PHP** (PHP - Wikipedia, the free encyclopedia, n.d.)

PHP works as following:

i. Client sends request containing the php code to the web server
ii. The web server sends the php code to the PHP interpreter module on the web server.
iii. The interpreter interprets the php code.
iv. The interpreted code is then executed by the php module.
v. The result is then sent back to the main web server process.
vi. The web server sends back the php output as response to the client.

The working of PHP is illustrated in the following figure:

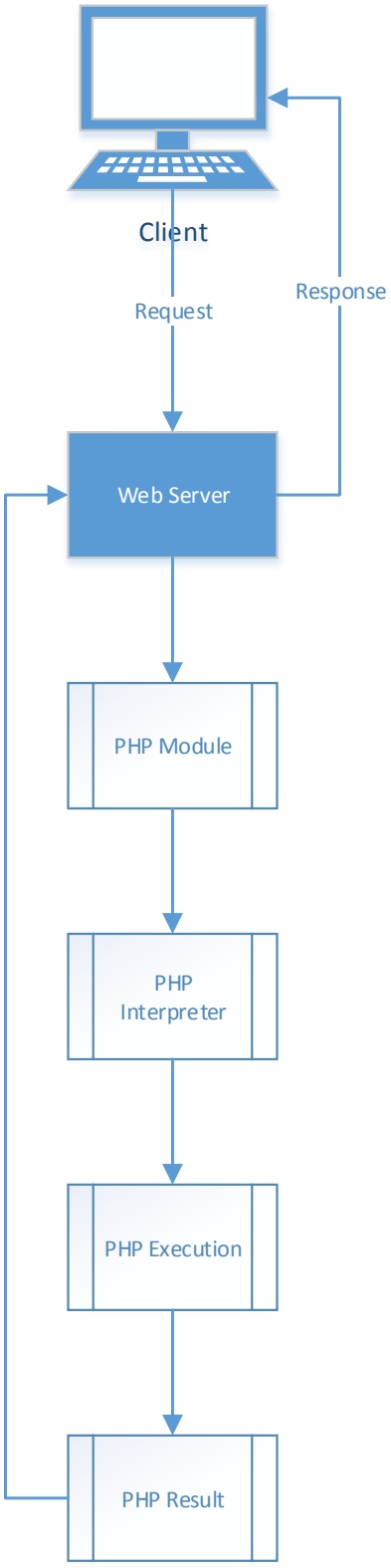

Figure 20: PHP: General working flow

4.1.2.2 PHP Architecture (PHP - Wikipedia, the free encyclopedia, n.d.)

PHP has an extensible plug in based architecture. This allows the functionality of the php to be extended. Plug-ins can be developed for different products to allow their interaction with PHP. Also, PHP is object oriented, that allows many of the features of object orientation like inheritance to be used.

It should be noted that overloading is not allowed in php.

4.1.2.3 PHP Capabilities (PHP: What can PHP do? - Manual, n.d.)

Some of the things that PHP can do includes (but are not limited to)

vii. Produce HTML code for a web page

viii. Generate multimedia

ix. Manipulate data from different data sources

x. Creating APIs for web applications

4.1.2.4 PHP Syntax

All the PHP code should be enclosed with in "<?php" and "?>". The interpreter looks for the code inside the <?php block, interprets it and execute it.

A single PHP file can contain more than one <?php ?> blocks.

The whole numbers and floating numbers in PHP are platform dependent.

4.2 Implementation

The implementation was done using PHP in eclipse. For the implementation, some PHP extensions had to be enabled and/or installed.

xi. For PostgreSQL: php_pgsql.dll

xii. For Memcached: php_memcache.dll

4.2.1 Coding

The coding work was divided into following parts:

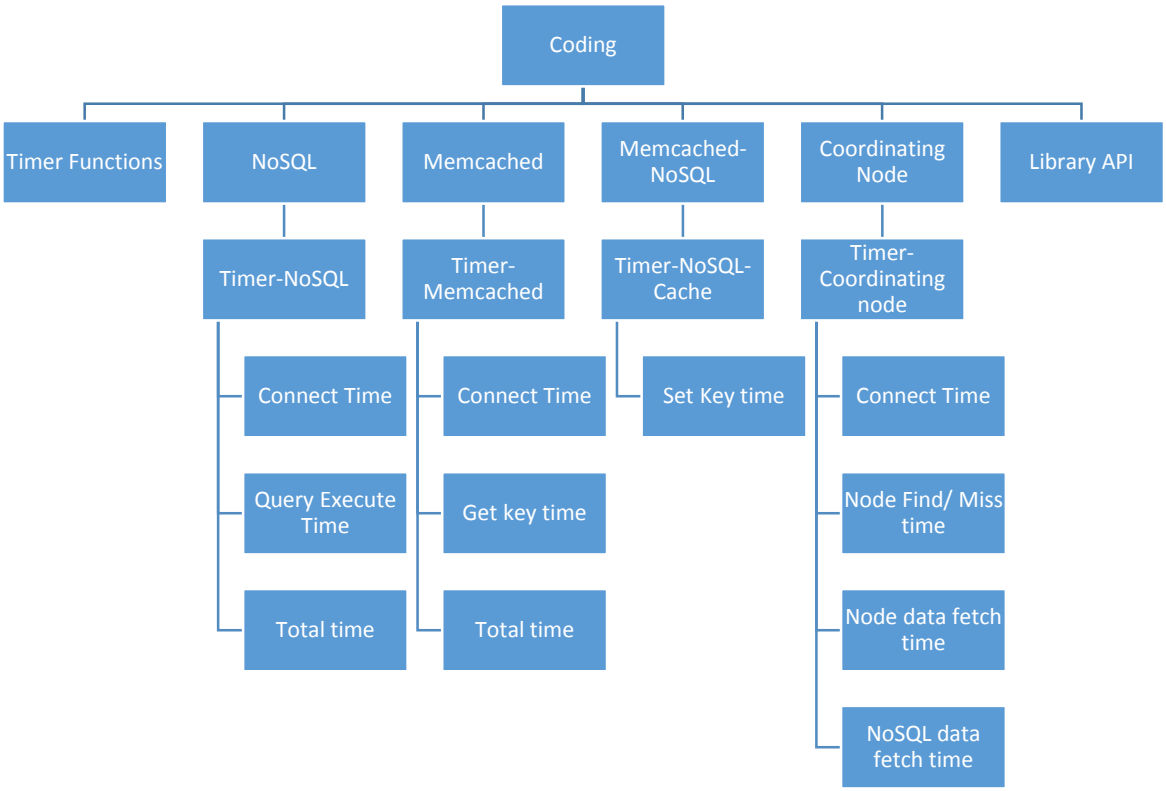

Figure 21: Coding Parts

4.2.1.1 Timer Function Class

The timer class is implemented to measure the time taken by each operation. The timer function can be used anywhere as:

```
require_once 'timer.php';  //import timer class
$timer=new Timer();
//operations
$Elapsed_time=$timer->elapsed();
```

4.2.1.2 Coordinating Node Functions

The coordinating node holds all the data about the cached content and the node where data is cached. The following functions implements the coordinating node:

i. memc_connect_Cnode()
ii. function memc_connect($server='localhost',$port=11212,$serverIndex=0)
iii. function memc_SetKey($serverIndex,$key,$value,$secondsToStore=0)
iv. function memc_getKey($key)
v. function memc_findNode($key)
vi. function memc_identifyServer($serverIndex)

4.2.2 Running System Screenshots

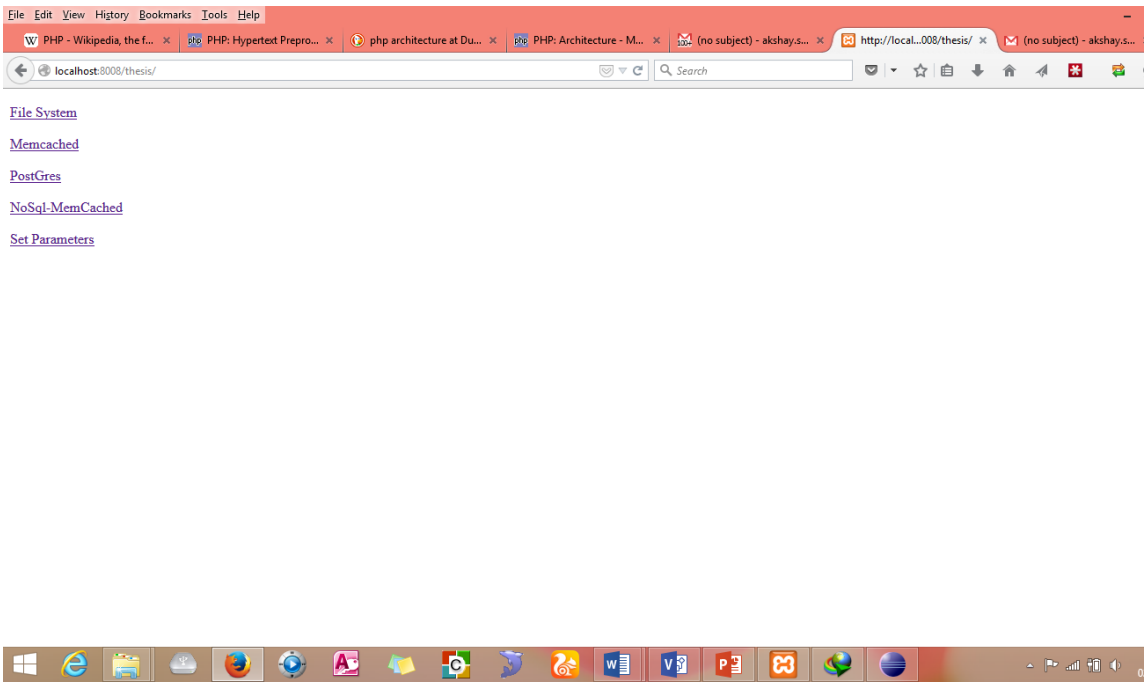

Figure 22: Implementation: Main Menu

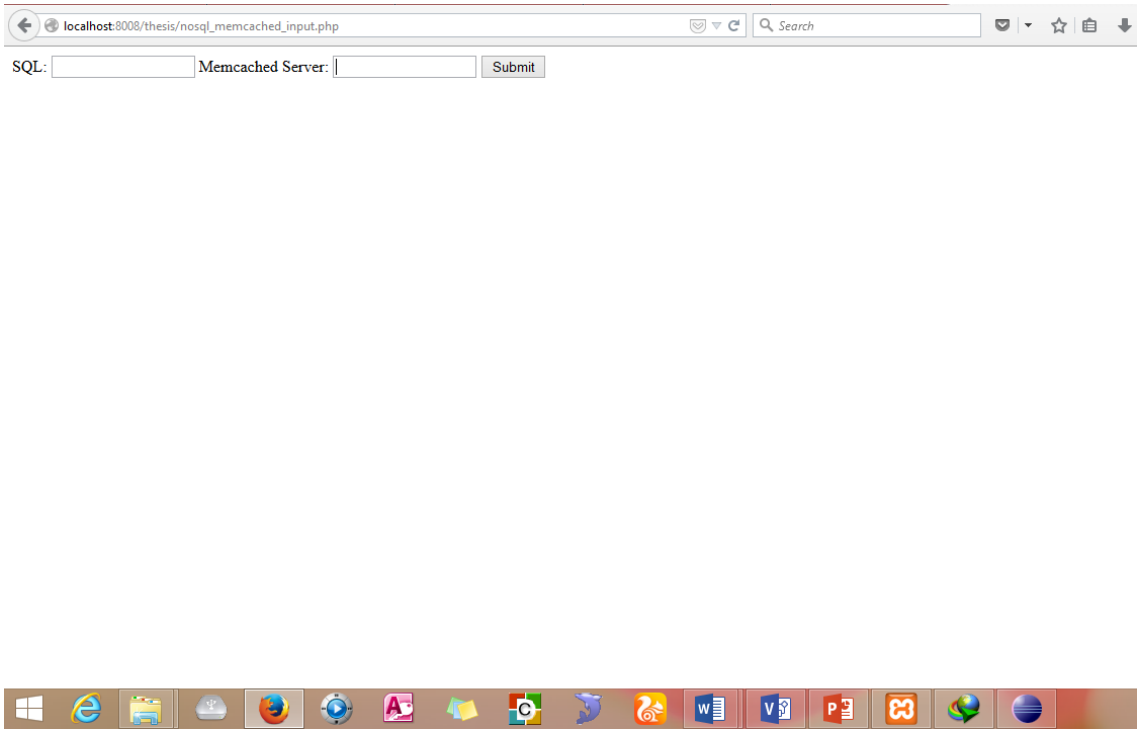

Figure 23: Implementation: Input page to set SQL query and Caching Server Node

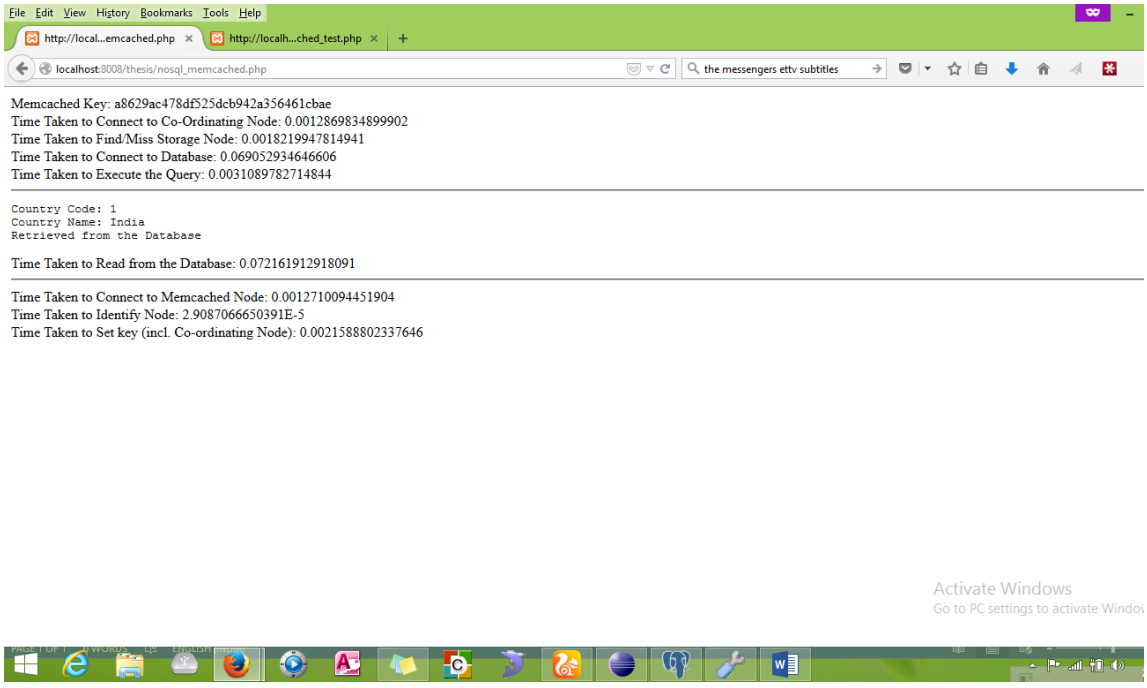

Figure 24: Implementation: Time Statistics for DB Data fetch and Set Key in Cache

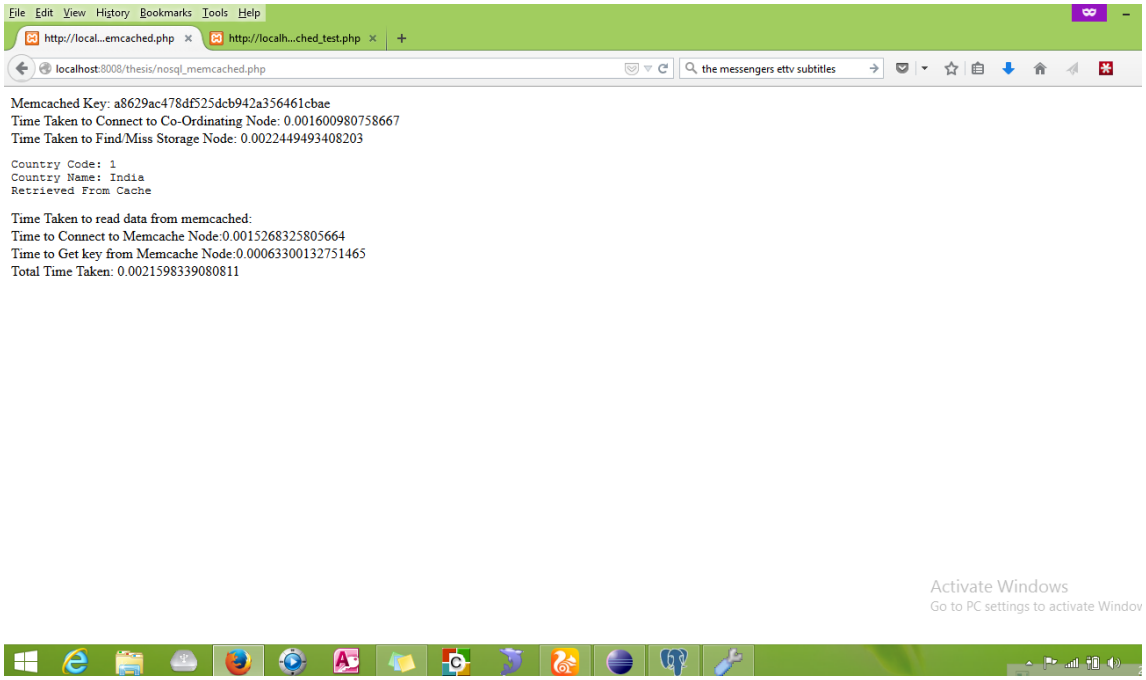

Figure 25: Implementation: Time Statistics for Cache Data fetch

4.3 Performance Evaluation

Below is the performance evaluation of NoSQL Database and Memcached. The evaluation is divided into 4 parts. Every part had 10 trials of NoSQL and Cache node.

All the times are in microsecond.

4.3.1 Time taken to connect to server

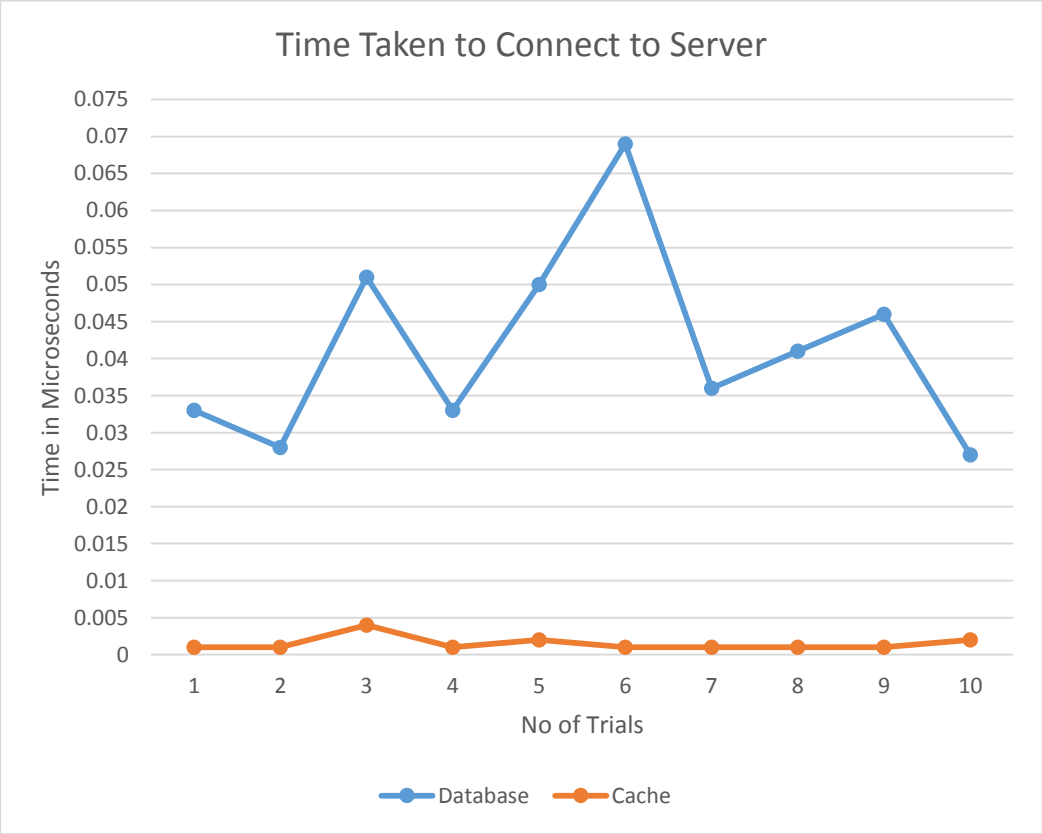

Figure 26: Graph: Connect time: NoSQL vs Cache Node

Table 2: Connect time: NoSQL vs Cache Node

Trial No.	Database	Cache
1	0.033	0.001
2	0.028	0.001
3	0.051	0.004

4	0.033	0.001
5	0.05	0.002
6	0.069	0.001
7	0.036	0.001
8	0.041	0.001
9	0.046	0.001
10	0.027	0.002
Average	0.0414	0.0015

Result:

On an average, connecting to the memcached server is 27.6 times faster than connecting to NoSQL server.

4.3.2 Time taken to fetch data

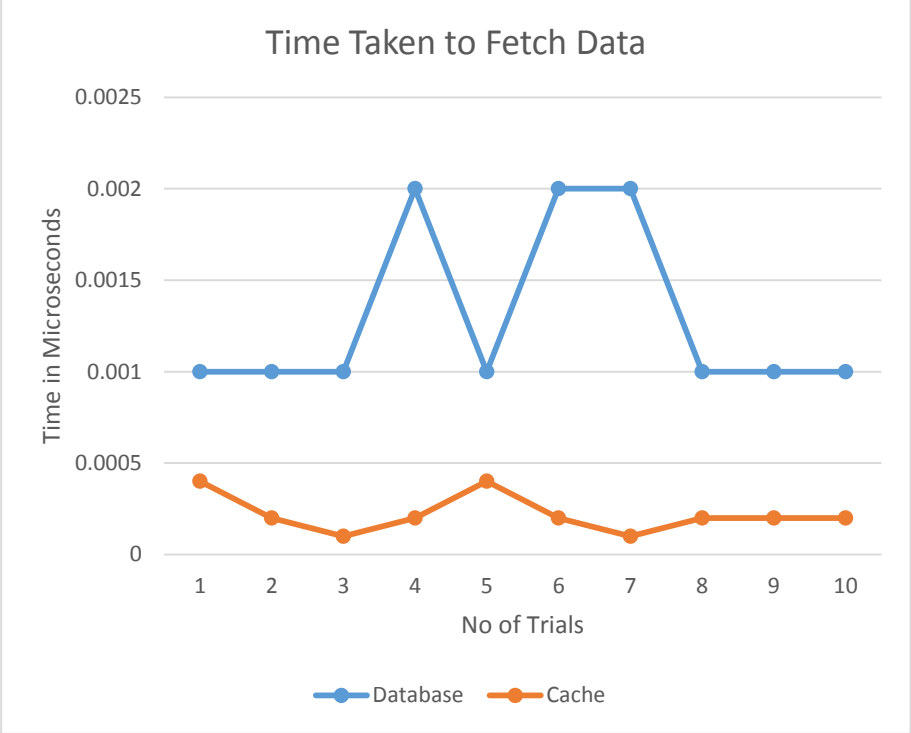

Figure 27: Graph: Data fetch time: NoSQL vs Cache Node

Table 3: Data fetch time: NoSQL vs Cache Node

Trial No	Database	Cache
1	0.001	0.0004
2	0.001	0.0002
3	0.001	0.0001
4	0.002	0.0002
5	0.001	0.0004
6	0.002	0.0002
7	0.002	0.0001
8	0.001	0.0002
9	0.001	0.0002
Average	0.001	0.0002

Result:

On an average, fetching data from memcached server is 5 times faster than NoSQL server.

4.3.3 Total time taken (without coordinating node)

Table 4: Total time (without coordinating node): NoSQL vs Cache Node

Trial No	Database	Cache
1	0.034	0.0014
2	0.029	0.0012
3	0.052	0.0041
4	0.035	0.0012
5	0.051	0.0024
6	0.071	0.0012
7	0.038	0.0011

8	0.042	0.0012
9	0.047	0.0012
10	0.028	0.0022
Average	0.0427	0.00172

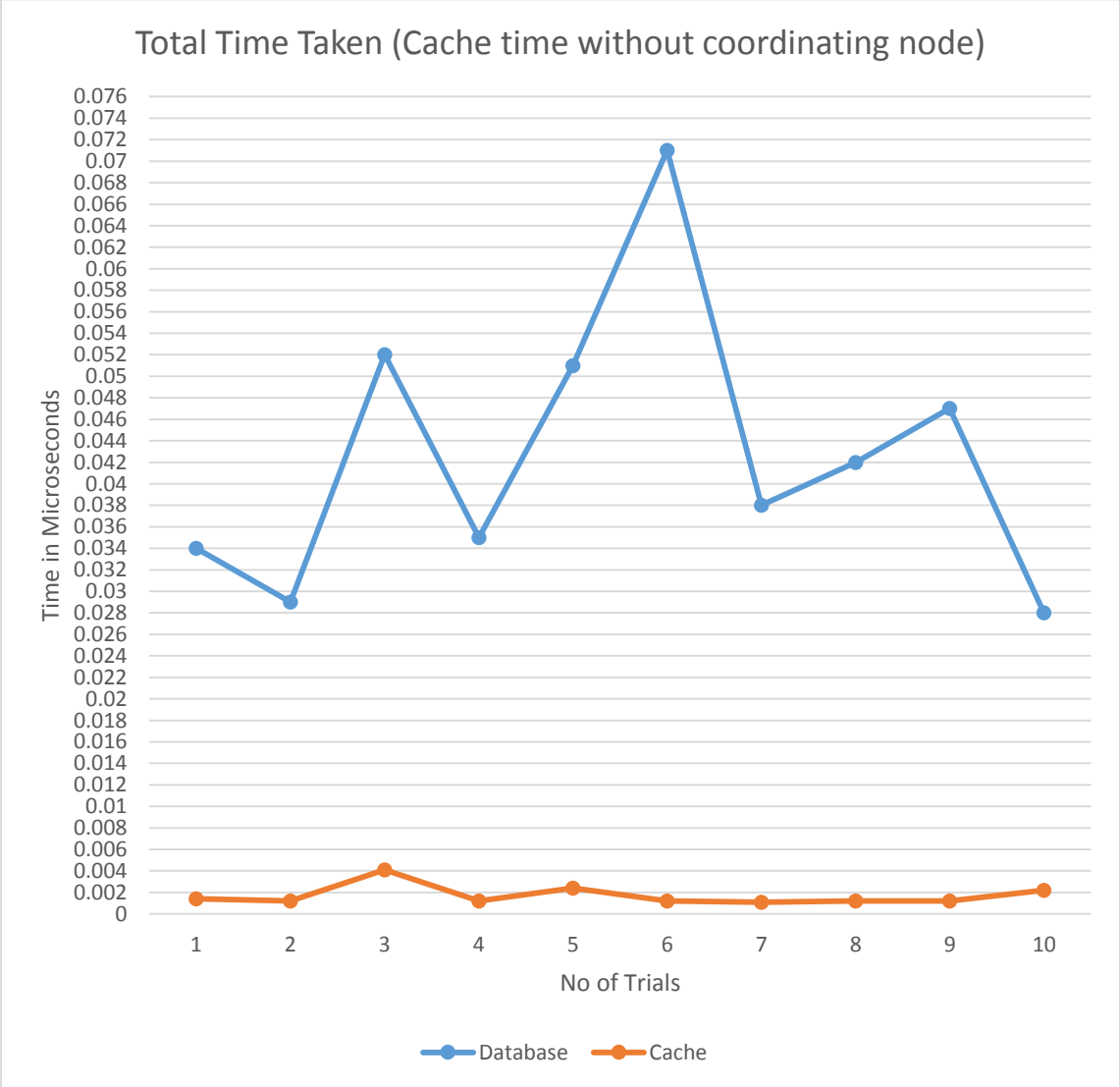

Figure 28: Graph: Total time (without coordinating node): NoSQL vs Cache Node

Result:

On an average, in total, fetching data from memcached is 24.8 times faster than NoSQL.

4.3.4 Total time taken (with coordinating node)

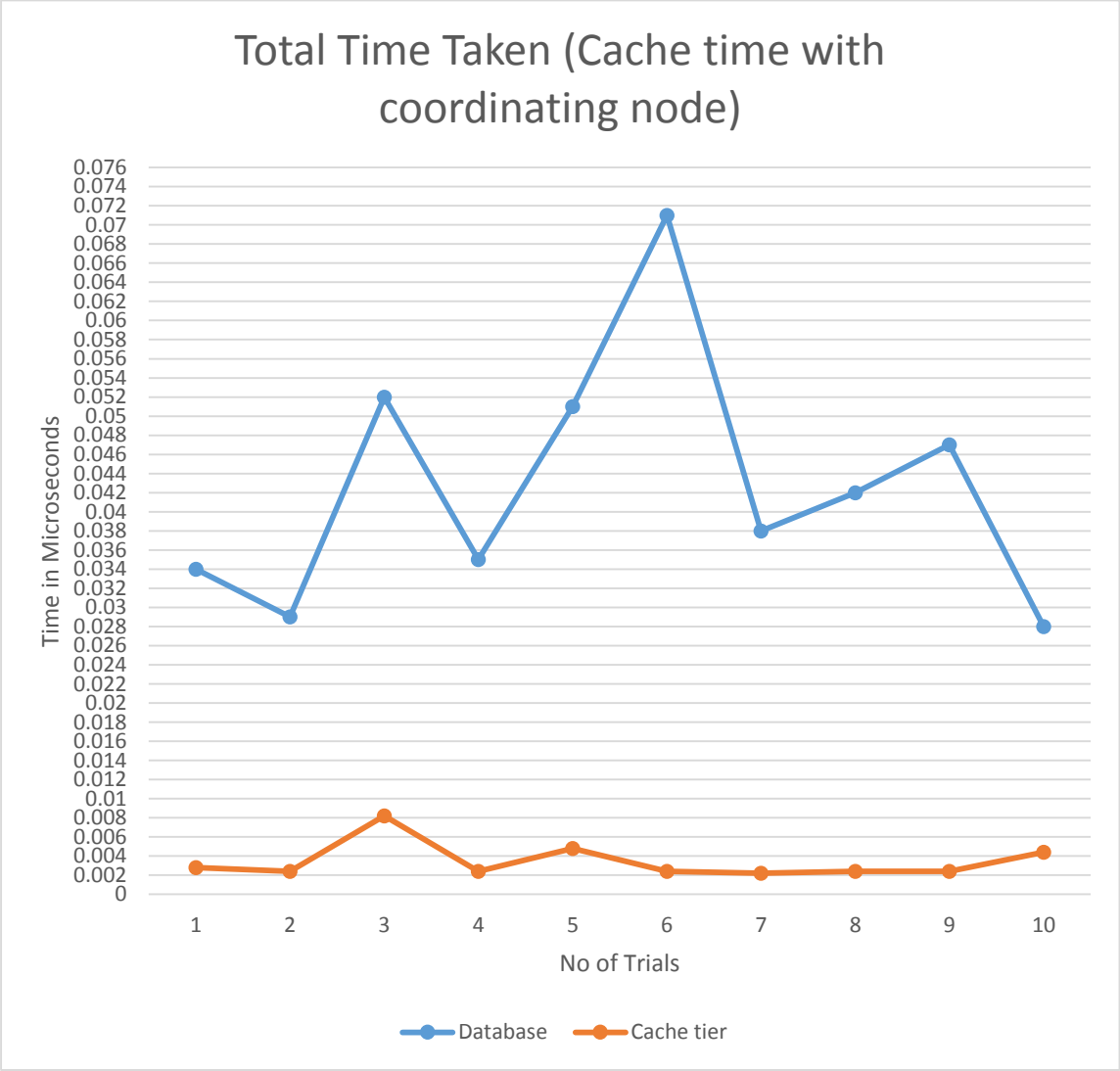

Figure 29: Graph: Total time (with coordinating node): NoSQL vs Cache tier

Table 5: Total time (with coordinating node): NoSQL vs Cache Tier

Trial No	Database	Cache tier
1	0.034	0.0028
2	0.029	0.0024
3	0.052	0.0082
4	0.035	0.0024

5	0.051	0.0048
6	0.071	0.0024
7	0.038	0.0022
8	0.042	0.0024
9	0.047	0.0024
10	0.028	0.0044
Average	0.0427	0.00344

Result:

Even with the coordinating node, fetching data from memcached is 12.41 times faster than NoSQL.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The experiments show that memcached considerably improves the read performance of the NoSQL databases. Also, by implementing a cache tier, using a coordinating node to cache the location of the cached content is deployed. Even by using this library, the retrieval is approximately 12.5 times faster.

Also, the INDICANO library provides an interface to clients having the caching server requirements. The library provides a standardized mechanism for the cache distribution function so that the client does not have to handle distribution transparency separately. Also, the resource utilization is achieved by sharing the NoSQL node's memory for caching purposes.

5.2 Future Scope

The INDICANO library implements the cache retrieval functions. Although, the library is capable to set data in cache, but it needs an explicitly mentioned cache server to set cache data to. In future, the server to cache data to should be decided automatically.

Also, the resources are shared between the NoSQL and cache node. The amount of resources being allocated to cache at any given point of time should be decided automatically.

CHAPTER 6
REFRENCES

Appuswamy, R., Gkantsidis, C., Narayanan, D., Hodson, O., & Rowstron, A. (2013). Scale-up vs Scale-out for Hadoop: Time to rethink? ACM SOCC, 13(October).

Arcot, R., Kum, H.-C., Carsey, T., Lander, H., & Sankaran, S. (2012, September). DataBridge - A Sociometric System for Long-Tail Science Data Collections.

Eclipse - Wikipedia, the free encyclopedia. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Eclipse_(software)

Eclipsepedia. (n.d.). Retrieved from http://wiki.eclipse.org/Main_Page

EMC Corporation. (2011). Big Data and Hadoop-101, Information Dynamics.

Flash Memory Summit 2014. (2014). Ultra Massive application storage capacity for real-time applications - 20140806_G22_Hassidim.pdf. Retrieved from Flash Memory Summit: http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2014/20140806_G22_Hassidim.pdf

Han, J., E, H., Le, G., & Du, J. (2011). Survey on NoSQL database. 6th International Conference on Pervasive Computing and Applications (ICPCA) (pp. 363-366). Port Elizabeth: IEEE.

Help - Eclipse Platform. (n.d.). Retrieved from http://help.eclipse.org/luna/index.jsp

Help - Eclipse Platform. (n.d.). Retrieved from http://help.eclipse.org/luna/topic/org.eclipse.php.help/html/index.html

Li, Y., & Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) (pp. 15-19). Victoria: IEEE.

memcached-a distributed memory object caching system. (n.d.). Retrieved from http://www.memcached.org

NoSQL - Wikipedia, the free encyclopedia. (n.d.). (The Wikimedia Foundation) Retrieved from https://en.wikipedia.org/wiki/NoSQL

NoSql Databases. (n.d.). Retrieved from http://nosql-database.org/

PHP - Wikipedia, the free encyclopedia. (n.d.). Retrieved from https://en.wikipedia.org/wiki/PHP

PHP: What can PHP do? - Manual. (n.d.). Retrieved from http://php.net/manual/en/intro-whatcando.php

PHP: What is PHP? - Manual. (n.d.). Retrieved from http://php.net/manual/en/intro-whatis.php

Plattner, P. D., Dimitropoulos, D. X., Stuedi, D. P., & Droz, D. P. (2012). An In-Memory RDMA-Based Architecture for the Hadoop Distributed Filesystem. Zurich: IBM Research.

PostgreSQL - Wikipedia, the free encyclopedia. (n.d.). (Wikimedia Foundation) Retrieved from https://en.wikipedia.org/wiki/PostgreSQL

PostgreSQL at a glance | CUBRID Blog. (n.d.). (CUBRID.org) Retrieved from http://www.cubrid.org/blog/dev-platform/postgresql-at-a-glance

PostgreSQL: The world's most advanced open source database. (n.d.). (The PostgreSQL Global Development Group) Retrieved from http://www.postgresql.org

Schindler, J. (2013). Profiling and Analyzing the I/O Performance of NoSQL Dbs. SIGMETRICS'13 (pp. 389-390). Pittsburgh: ACM. Retrieved June 2013

Shah, H., & Bohara, M. h. (2014). A Brief Introduction to Memcached with its Limitation. International Journal of Engineering Research & Technology (IJERT), 3(2), 1941-1943.

Shrivastava, M., & Bischof, D. H.-P. (2013, July 24). Hadoop-Collaborative Caching in Real Time HDFS. International Conference on Parallel and Distributed Processing Techniques and Applications, p. 2013.

Singh, G., Chandra, P., & Tahir, R. (2012). A Dynamic Caching Mechanism for Hadoop using Memcached. Urbana Champaign: University of Illinois.

Soliman, A. (2013). Getting Started with Memcached. Packt Publishing Ltd.

Tudorica, B. G., & Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. Roedunet International Conference (RoEduNet), 2011 10th (pp. 1-5). Iasi: IEEE.

Xie, J., Yin, S., Ruan, X., Ding, Z., Tian, Y., Majors, J., . . . Qin, X. (2010). Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters. 19th International Heterogeneity in Computing Workshop. Atlanta, Georgia: IEEE.

Xu, C., Huang, X., Wu, N., Xu, P., & Yang, G. (2010). Using Memcached to Promote Read Throughput in Massive Small-File Storage System. Grid and Cooperative Computing (GCC), 2010 9th International Conference (pp. 24-29). Nanjing: IEEE.

Zhang, J., Wu, G., Hu, X., & Wu, X. (2012, September). A Distributed Cache for Hadoop Distributed File System in Real-time Cloud Services. ACM/IEEE 13th International Conference on Grid Computing, pp. 12-21.

Zhang, W., Rajasekaran, S., & Wood, T. (2013). Big Data in the Background: Maximizing Productivity while Minimizing Virtual Machine Interference. Third Workshop on Architectures and Systems for Big Data (ASBD 2013).