National College of Ireland

Improving response time for Geographically distributed caches using query result caching

MSc Research Project
MSc Cloud Computing

Pooja Raghav
Student ID: x17158788

School of Computing
National College of Ireland

Supervisor: Sachin Sharma

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Pooja Raghav.
………

Student ID: x17158788

Programme: MSc cloud computing **Year:** 2018-2019

Module: Research in computing
…….………

Supervisor: Sachin Sharma
…….………

Submission Due Date: ………28/12/2018……………………………………………………………………………………

Project Title: Improving response time for Geographically distributed caches using query result caching

Word Count: 5937 …………………**Page Count**………………… 22 ……………………….……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Pooja Raghav ………

Date: …20/12/2018………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Improving response time for distributed caches using query result caching

Pooja Raghav
X17158788

Abstract

Caching is an important technique to reduce network traffic and increase the performance of the applications. In distributed or client/server database environments caching can introduce the issue of inconsistency and latency. Traditional caching architecture is difficult to meet the needs of the users and business both, where faster responses are expected. So, to improve this technique we have proposed an architecture for a distributed network environment where a local cache interacts with a remotely located cache in case of any data miss and updates the local cache for faster responses. In a scenario where remote cache to misses the queried dataset the request to fetch the records is directed to the Primary data store, which will update both the local and remote caches with the desired data. This approach drastically reduces the response time for the user to fetch the same set of record next time, decreasing the latency time and increasing the overall performance of the infrastructure.

1 Introduction

The concept of caching has been long known and recognized as a powerful technique for performance improvement in computer systems. In large distributed systems which often separates the client machine from the data store by the network, it is desirable to have caches for the most immediate used block of data to avoid delay in communication or most precisely to avoid latency issues.

Thus, this technique heavily reduces the need for an application to fetch the same dataset which can be both expensive and time taking by storing the intermediate information in the local cache. Caching also introduces the most frequent problem of ensuring and maintenance of consistency of data in the cache and the central data store (Primary database). In largely distributed caches it is difficult to ensure consistency with the traffic required which can be a vital factor in ensuring the cache performance. [1]

In this paper, we have focused on the issue of data consistency for users in the distributed environment. We have considered a simple distributed client/server architecture with a centralized database. The caches are distributed in two different regions. For clients in one region, the cache is referred to a local cache and the cache in another region is a remote cache. Unlike previous client /server database architecture where the local cache interacts with the database in case of any cache miss, here the local cache tries to contact remote cache for data. Furthermore, if the remote cache is unable to return the dataset the request for data is directed to the centralized database. The centralized database not only updates the remote cache but also the local cache. Thus, this approach heavily resolves the issue of latency, as both the caches are updated for the missing record and the user can access the record for the same missing dataset in no time. We have focused on improving latency and derived results from various test cases.

Thus, this approach benefits in the following ways:

The research paper is fragmented in seven broad sections. The Abstract is followed by Introduction. Section 2 focusses on the literature review throwing the light of the previous state of the art and traditional client /server cache database architectures for caches in distributed environment Literature review throws an adequate light on the proposed state of the art. Methodology section in detail describes the underline functions and workflow of the proposed client/server and cache implementation. This has been explained in detail with the help of a UML diagram. We have also discussed every

software components used for the implementation of the architecture. The next section of the paper called “Design and Development in details talks about the logical architecture, followed by software and configuration specifications for the system design for the proposed architecture. The design and implementation involved writing of three automation scripts index.php,getCustomerdetails.php and searchCache.php whose purpose and description has been made in the respective section. The evaluation section focuses on six used cases with different conditions and verifies response time for each. This response time is used to evaluate the difference and efficiency with respect to the availability of data in Cache (local and remote). The Last section of the report emphases on the future recommendation and the areas of improvement for the proposed state of the art. Front-end development and development of customized micro-services to record the transactional information maintained by the database to determine the changes are the most important recommendations made. Thus, the paper concludes on the note how caching functionalities can be improved in a distributed environment as Caching technique can drastically improve latency, inconsistency and performance issues of the databases.

2 Related Work

Caching is critical and its main purpose is for improving the performance of many middleware applications and in order for an application to benefit from caching, it must repeatedly use data which is expensive to calculate or to retrieve.

2.1 Query Result Caching

The database systems have often been a subject of concern for performance of the applications. The need for speed is increasing rapidly so we must be able to handle a large amount of data in small time units like seconds or even milliseconds.

The main problem that affects the usage and the performance bottleneck is the intensive usage of the database through unnecessary, and repetitive calls. The database component can be scaled immensely with the approach of cache query but these on the other hand pose a great challenge in maintaining a higher cache hit rate and also has problems in efficiently managing cache consistency while the database is being updated. [8]

A lot of applications have failed in efficiency due to an enormous and large number of redundant database calls, huge network traffic between the database servers and the applications, fetching and loading the results which have been requested from the server. [8]

Today numerous online business applications are deployed and being developed in a distributed environment which involves internet-based clients, databases and web application servers. These applications generate a lot of web pages dynamically which serves the purpose of caching and make it an effective approach to gain high availability, scalability and efficient performance of the infrastructure. [8]

This limitation can be overcome by dynamic data cache. It saves the results of queries that are submitted to the database system. A cache hit is recognized and serviced from the cache if a query is an identical match to a previously submitted query. The advantage of such caching is that it is simple and it caters to access scenarios where the same query is likely to be submitted over and over.

2.2 Query Cache Management

The data amongst the disk and the main storage is generally exchanged through the I/O operations. The data which is required by the system is frequently cached in the cache memory of the systems. Generally, the RAM occupies almost 40 – 50% of the cache memory of the RAM present in the system. The cache memory can decrease over the period of the time as there is an increase in the number of applications being installed and configured on the system. This leads users or applications to establish connections with the database directly as cached memory gets deleted more frequently from the system. [9]

Any Physical connections with the database directly cost and become an expensive affair for the business. On the other hand, access to the cache memory is inexpensive and the same data can be retrieved any number of times without establishing any physical access to the database. The main goal of the proposed system is to reduce the latency of the response time and to decrease the physical access to the Database system. Here we also ensure that data must always come from the cache and not from the database as this reduces the response time and makes the retrieval of the information faster when compared to a database. [9]

Since databases are structured, Caches (database) rely heavily upon the traditional method of storing the whole Database data structure as its subset. Often the data is cached based on the location of the cache memory and read and write operations are invalidated based on the memory locations of the caches. The results are not cached on the basis of the actual query. Since the queries can be combined and retrieve results from any location of the database. These queries request are based on a certain logic and is not on the location of the caches. So it becomes extremely difficult to be able to determine if the previous results for the queried data does exist or not due to the changes made to the data.[12]

2.3 Query Response Time

Every user who tries to contact the application tries to establish a request to the browser which is directed to the web browser. The request on the web server is directed and processed to the database. The response to the user is returned by the application through the web browser. Every request that comes to the web server is directed to the database to be able to process. This is independent of the fact that whether it is a new request or the old request. This can create an excessive heavy workload from the webservers and often leads to network congestion, latency, and low system performance and lack inefficiency of the system. [10]

2.4 Content-Blind Cache

The first issue regarding the cache we face while designing is to be able to understand and analyze what data needs to be cached and how it can be stored. For this paper, the possible and less explored design methodology of content-blind caching is considered. Here, the response from every query can only be retrieved if the same query has been cached before for the results. This approach gave many advantages. This leads to mass scaling for higher loads even through the process of checking for trivial queries. This approach drastically reduces the overhead on the databases for the planning and execution of the queries, which eventually takes away the responsibility of edge servers to even run the DBMS functions and which is highly appreciated in the event of high loads on the systems and databases. Replacement of cache is made simple as results are stored in an independent manner. Caching also involves updating of the query results when the database is updated. [11]

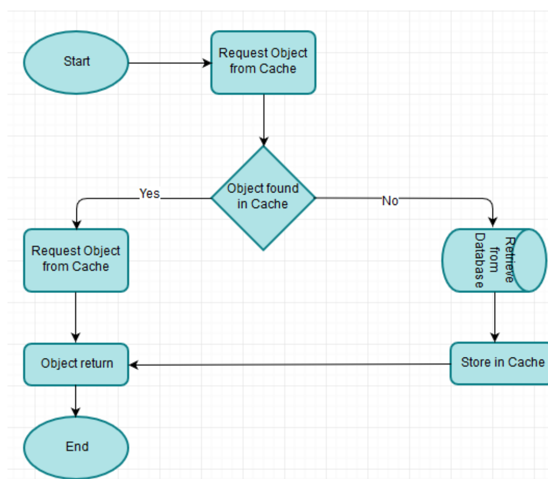

Fig 1. Caching mechanism

3 Research Methodology

In a three-tier architecture, the application software tier and data storage tier can be in different hosts. The throughput of an application can be limited by the network speed. This restriction can be curtailed by introducing a database at an application level. Because commercial database software makes extensive use of system resources, it is not always practical to have the application and the database at the same host. Considering all the scenarios, a more light-weight database application can be used to cache data from the enterprise database management system. [3]

In an enterprise, the volume of data and the number of users grow rapidly, and eventually the access to enterprise database often becomes a bottleneck (i.e. when there is more data, then each query takes longer than the expected time which lead to performance bottleneck). Caching of database data is the solution to this problem. In most cases, the caches are set up in such a way that on every query, it first checks from the cache instead of direct database check, and if it's a hit, then value is returned otherwise it is considered as a miss. In the case of a miss, business logic to be implemented to store the content in both local and remote cache and return it to the client whilst also storing it in the cache for the next time. [3]

The assumption behind caching of the data required for application is that – the data that is recently read have a higherchances of reading it again. Thus, the data retrieved from the database should be stored in a faster memory so that even the next read will be from cache, which resolves the main purpose i.e. it is quicker and reduces the issue of latency. [3]

In the proposed architecture we have considered two Caches which are kept in two different geographical area. One cache is treated as the remote cache while the other cache in different geographical region is treated as the remote cache. These Caches will interact with each other when data is not to be found in the primary cache. In the event when no data is available in the remote Cache the, the request is directed to the centralized Database to update both the local and the remote cache in the regions respectively. This not only resolves the latency issues but also improves the response time of the nearby caches. This approach also reduces the load on the centralized database and heavly reduces the network Traffic.

Below is the Cache Middleware Architecture Diagram for the proposed approach

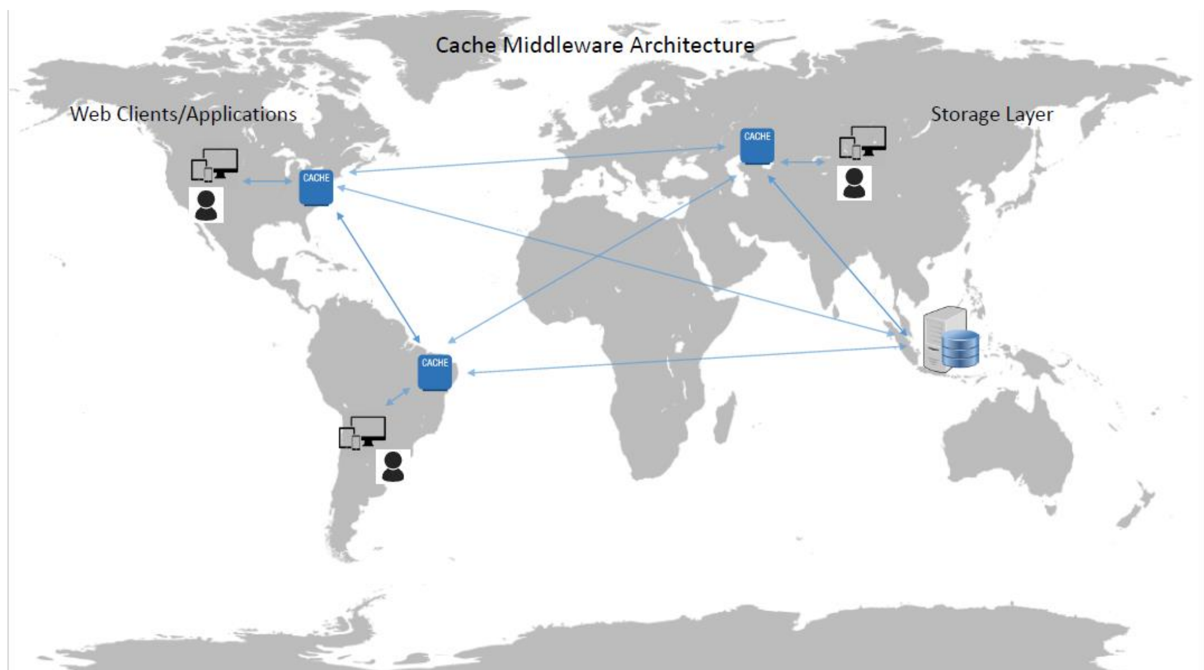

Fig 2. Cache Middleware Architecture

Our proposed logical architecture is distributed across the two geographical regions in the AWS cloud platform. At this point in time we have used the below software for the deployment of client/server database caching.

- AWS (the cloud service provider)
- Amazon Linux AMI images
- MySQL database
- MySQL Workbench
- Apache Tomcat
- Redis
- Phpredis
- Redis Desktop Manager

3.1 Redis

Today, Redis is ones of the most used database cache software's used by all the big actors of the industry, and when used properly, can handle millions of data operations per second. The Redis cache is one of the best option compared to memcached in the market as it has various advantages over memcached, for example, it is best in handling sessions and the application cache which supports the data guarantee, locking, and key awareness. Redis cache is of the most widely and popular caching system for high performance distributed environment. It is basically a simple key-value lightweight caching system with an extra advantage of persistence and maximum memory usage. [2]

Redis is an open source and in-memory key-value data store that can be used for database, caching, message broker, and queue but for this paper, we have used Redis as a caching layer which stores the query results run against the MySQL database to both local and remote caches. Redis is a in-memory key-value data store, which keeps the data retrieved in memory and it can be accessed faster as compared to query the data from MySQL by the applications. [2]

3.2 MySQL Workbench

MySQL Workbench is a tool for graphical and visual designing of databases. This tool is mostly integrated with MySQL databases, wherein it helps user to perform the SQL transactions (like database creation, manipulation to table data etc.). MySQL Workbench fully supports MySQL server versions 5.6 and higher.

MySQL Workbench is made available to users in two editions – the Community Edition and the Commercial Edition. The Community Edition is available free of cost to the users while the later cost. The Commercial Edition offers additional Enterprise features and for this paper, we have considered only the Community Edition. With the use of such visualization design tools is the MySQL workbench designers can build a good, efficient and fast database. Visualization tools in workbench make ERD (entity relationship diagram) and relational schema mapping easy. [5]

3.3 Phpredis

There are many PHP clients in the market which can communicate with Redis server, out of which the Phpredis is one among the popular and we have considered it as an API for communicating with the Redis, the in-memory key-value data store.

Though MySQL will be able to handle 8 requests per second, we still considered introducing a cache layer via Redis as a best practice for a scalable and highly available design (which improves the latency to a great extent). And to use Redis, we will need to use a library which supports Redis for PHP (for example, phpredis can be used for connecting PHP with redis). And cache the results of a query in red.

3.4 Redis Desktop Manager

Redis Desktop management tools provide an easy way to manage Redis database RDM maintains an in-memory data store which can be used for the purpose of database cache and a message broker. RDM can handle the request as enormous as 100 rpm and supports complex data structures like list, hashes, strings, sets. With RDM we get access to Execute, Watch, Discard and Multi-functions for transactional control [6]. These tools provide features so that every key can be checked and a cache can be cleared easily. One default tool comes with Redis called Redis-client. A visual tool that is great and easy to use, called Redis Desktop Manager (RDM).

Think of the application logic as the following 3 layers:

- Application code
- Redis Cache
- MySQL Data Store

When the user requests for a data, the request is routed to the local cache first. The client expects to retrieve the dataset from the local data store known as the cache. But in scenarios where data is missing in the local cache, the subsequent request is placed to fetch and cache the latest information from the nearest or the remotely located cache. If the data is missed in the remote cache a request is invoked to the centralized database for the missing dataset, thus the Database thus ensures to update both the local and remote cache in the distributed environment for the missing key-value. This allows the users (which are dispersed across different geographically regions) in fetching the data directly from the nearby cache servers which are distributed across and the visibility where the data is fetched is unknown to end users.

When there is a need to read the enterprise data very frequently then the database cache supplements the primary database by removing performance bottleneck and unnecessary pressure. Cache serves as the primary data store for the application and thus reduces the latency in fetching of the same data upon next hit drastically.

Fig 3. UML diagram of the workflow from Client/Cache -- Data Store

Based on the above workflow we have drawn six use cases and whose results will be evaluated in the later section as one of the most important parameters for predicting and calculating the decrease in latency. The Use cases are mentioned below

#	Use Cases
1	When the user tried to search for pattern Alpha when both local and remote cache is empty

2	The user tried to search for a pattern from local cache first, Data exists in local cache and Remote cache is empty
3	When the key is deleted from local cache and the remote cache is empty, then user tried to search for pattern Alpha
4	Data exists in the remote cache but not in the local cache
5	Data is deleted from both local and remote cache and then user tried to search for pattern Alpha
6	After data exists in both local and remote cache, the user tried to search for pattern Alpha

Table – 1 Use cases

STRING: sql_cache_Alpha Cognac TTL: -1 Rename Delete Reload Value Set TTL

Value: size: 319.00 bytes View as: JSON


```
{
  "customerNumber": "242",
  "customerName": "Alpha Cognac",
  "contactLastName": "Roulet",
  "contactFirstName": "Annette ",
  "phone": "61.77.6555",
  "addressLine1": "1 rue Alsace-Lorraine",
  "addressLine2": null,
  "city": "Toulouse",
  "state": null,
  "postalCode": "31000",
  "country": "France",
  "salesRepEmployeeNumber": "1370",
  "creditLimit": "61100.00"
}
```


Fig 4: Keyword Alpha search from MySQL workbench

4 Design Specification

4.1 Logical Architecture

Here we consider a remote cache stored on dedicated servers and are typically built on a key/value store. With remote caches in place, the application or the processes that use are responsible for the data validation and they are also responsible in orchestration between caching of the data. The application don't directly connect to database for queries but connected to cache to reduce the latency. Here, the average latency of a request to local or remote cache over a disk-based database is faster and is on the sub-millisecond timescale.

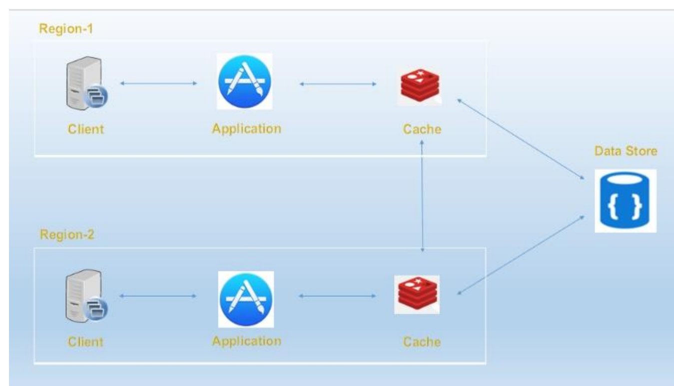

Fig 5 Logical Architecture of the system

4.2 Software and configuration Specifications

At a high level below are details of the software configurations:

- Install the Operating System (RHEL)
- Installation of MySQL database
- Install Java 8
- Install Tomcat

- ❖ Install Redis
- ❖ Configure Redis Java driver(s)
- ❖ Configure, activate and test the Cache
- ❖ Import sample data
- ❖ Install Redis Desktop Manager to manage cache both local and remote cache.

4.3 Software Configurations and Implementation Steps

4.3.1 IP and configuration details

We have considered two Linux AMI compute machines in AWS cloud platform running in two geographical locations EU-west and EU-east respectively. Below are the private and Public IP along with Domain name server mentioned:

#	Private IP	Public IP	Public DNS
1	172.31.0.198	52.209.226.243	ec2-52-209-226-243.eu-west-1.compute.amazonaws.com
2	172.31.2.76	34.244.240.245	ec2-34-244-240-245.eu-east-1.compute.amazonaws.com

Table 2: IP address of EC2 Instances

- ➢ The next step involves installation of MYSQL 5.6 on Linux AMI

sudo yum install -y httpd24 php70 mysql56-server php70-mysqlnd

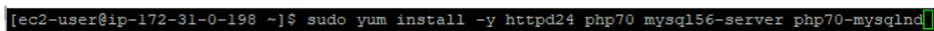

Fig 5. Screenshot for yum install

- ➢ We start the MySQL server for the database and login as MySQL as root to verify the database installation

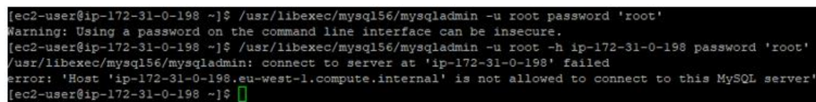

Fig 6. MySQL Data Store

4.3.2 ER diagram of MySQL database

The **classicmodels** database holds the details of models of classic cars. This database is configured in MySQL which contains the business data such customers information, products details, sales orders, sales order line items, etc. [7]

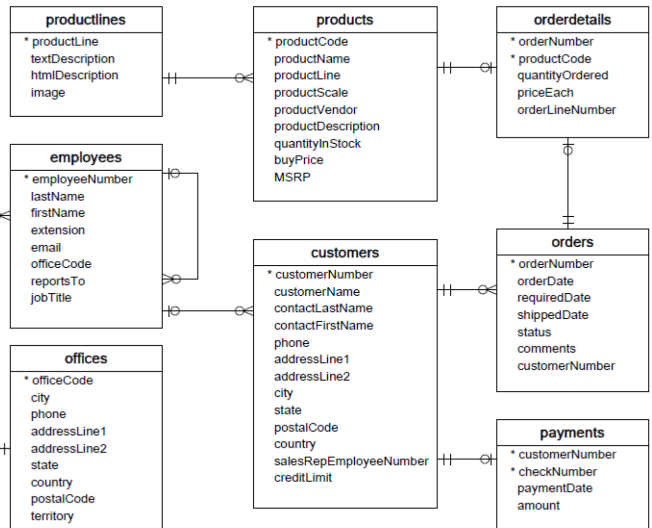

Fig 7: classicmodels database used in the implementation [7]

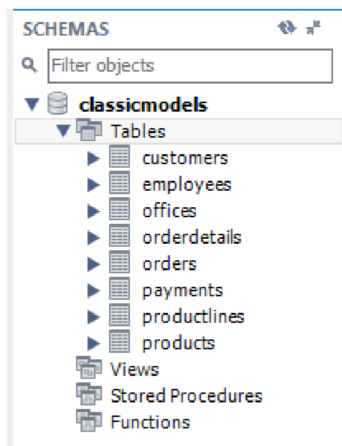

Fig 8 Tables used in classic models [7]

4.3.3 High level Component Installations

4.3.3.1 MySQL

http://www.mysqltutorial.org/mysql-sample-database.aspx

- Install the MySQL workbench
- Installation of JAVA on Linux instance. To install the single package (and its dependencies) on an Amazon AMI Linux instance, run the yum update command with the name of the package.
  sudo yum install java-1.8.0

```
[ec2-user@ip-172-31-0-198 opt]$ sudo /usr/sbin/alternatives --set java /usr/lib/jvm/jre-1.8.0-openjdk.x86_64/bin/java
[ec2-user@ip-172-31-0-198 opt]$ sudo /usr/sbin/alternatives --set javac /usr/lib/jvm/jre-1.8.0-openjdk.x86_64/bin/javac
[ec2-user@ip-172-31-0-198 opt]$
```

Fig 9 yum install for java 1.8

4.3.3.2 Tomcat

We use the yum tool to install tomcat8 package.
sudo yum install tomcat8 tomcat8-web apps tomcat8- admin-web apps tomcat8-docs-web app

This will install:

1. WebApps folder
2. Tomcat documentation (Gives local documentation for quick reference)
3. Admin App – through which one can install /export/start/ stop / reload an application (without having to log in)

```
[ec2-user@ip-172-31-0-198 ~]$ sudo yum list installed | grep tomcat
apache-tomcat-apis.noarch                0.1-1.6.amzn1                @amzn-main
tomcat8.noarch                           8.5.32-1.78.amzn1            @amzn-updates
tomcat8-admin-webapps.noarch             8.5.32-1.78.amzn1            @amzn-updates
tomcat8-docs-webapp.noarch               8.5.32-1.78.amzn1            @amzn-updates
tomcat8-el-3.0-api.noarch                8.5.32-1.78.amzn1            @amzn-updates
tomcat8-jsp-2.3-api.noarch               8.5.32-1.78.amzn1            @amzn-updates
tomcat8-lib.noarch                       8.5.32-1.78.amzn1            @amzn-updates
tomcat8-servlet-3.1-api.noarch           8.5.32-1.78.amzn1            @amzn-updates
tomcat8-webapps.noarch                   8.5.32-1.78.amzn1            @amzn-updates
[ec2-user@ip-172-31-0-198 ~]$
```

Fig 10 checking for tomcat install

➢ So Tomcat8 has now been successfully installed and started on Amazon Linux instance.
sudo service tomcat8 start

```
[ec2-user@ip-172-31-0-198 ~]$ sudo service tomcat8 start
                                                           [  OK  ]
[ec2-user@ip-172-31-0-198 ~]$
[ec2-user@ip-172-31-0-198 ~]$
```

Fig 11. starting tomcat service

The IP Address is 52.209.226.243 and the hostname is ec2-52-209-226-243.eu-wes1.compute.amazonaws.com

In the browser open either of the links - http://52.209.226.243:8080 or http://ec2-52-209-226-243.eu-west-1.compute.amazonaws.com:8080

You will see the Apache Tomcat startup page as shown below.

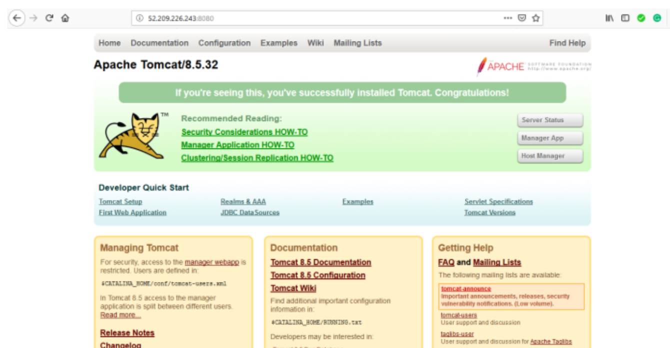

Fig 12. Tomcat Apache web page

➢ Also along with Tomcat, we require to configure the httpd to serve the files

4.3.3.3 Redis

The configuration of Redis to AWS EC2 Amazon Linux instance.

To have a successful installation of Redis, it is mandatory to compile additional software prior to Redis installation. For example, we need to install several Development Tools, such as make, and gcc, etc.

- Installation of Redis 5.0.3
- Configure Redis Server

After the compilation of Redis software, the src directory which is created as part of the installation will hold the different of Redis:

- redis-server = Radis Server
- redis-client = the command line interface utility which can help to talk with Redis Server

```
[ec2-user@ip-172-31-0-198 redis-stable]$ sudo mkdir /etc/redis
[ec2-user@ip-172-31-0-198 redis-stable]$ sudo mkdir /var/redis
[ec2-user@ip-172-31-0-198 redis-stable]$
```

```
[ec2-user@ip-172-31-0-198 redis-stable]$ sudo cp src/redis-server src/redis-cli /usr/local/bin
[ec2-user@ip-172-31-0-198 redis-stable]$ cd /usr/local/bin/
[ec2-user@ip-172-31-0-198 bin]$ ls -lrt
total 12852
-rwxr-xr-x 1 root root 8288456 Dec 7 20:45 redis-server
-rwxr-xr-x 1 root root 4867560 Dec 7 20:45 redis-cli
[ec2-user@ip-172-31-0-198 bin]$
```

4.3.3.4 Phpredis

- ➢ Installation of the Phpredis, verifying its connection with the Redis
- ➢ Next, we need to add an extension with PHP configuration so that PHP can enable this module

4.3.3.5 Redis Desktop Manager

- ➢ Installation of the Redis Desktop Manager version 0.9.3.817

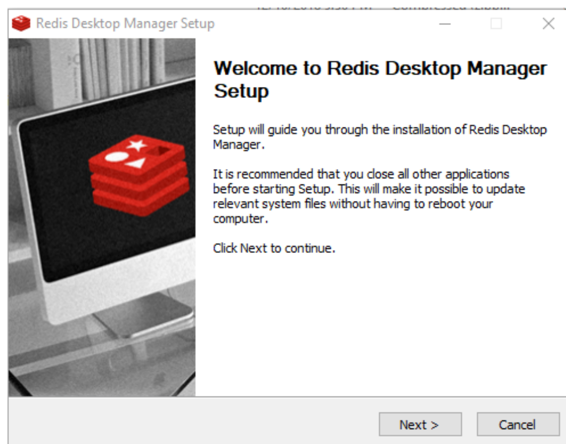

Fig 13. Redis setup window

4.3.3.5.1 Configure RDM

After the successful installation of Redis Desktop Manager, the next activity is to configure instances to connect to local or remote caches.

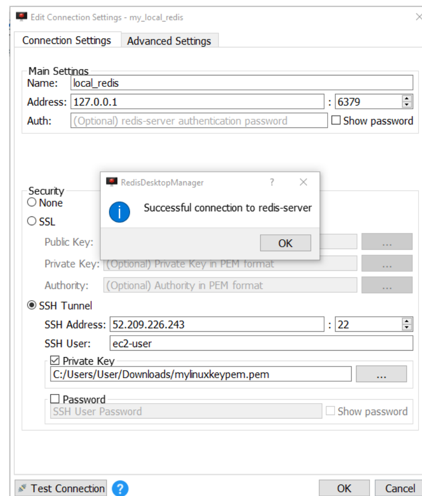

Fig 14 Configuration of RDM with EC2

➢ Storing MySQL queries results into the Redis Cache

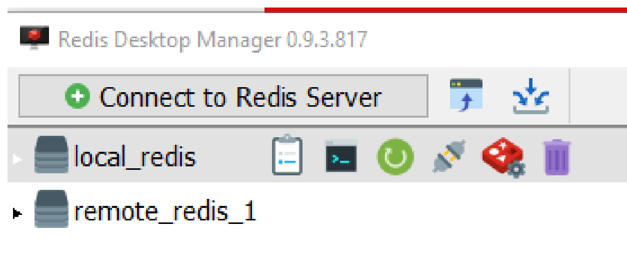

Fig 15 Local and Remote Redis cache in RDM

5 Code Implementation

5.1 Application Code

5.1.1 Script -1: index.php

The flow of the index.php script is as follows:

1. The script first checks for the count of keys available in Redis cache.
2. When the count is 0, then another script getCustomerDetails.php is invoked
3. In the getCustomerDetails.php script, we first call the executeQuery() function by passing the SQL query to be executed.
4. Then we pass the fetched rows resulted in Step-3 to save in Redis cache; during this, we also encode the array of results to JSON string

The script (index.php) will fetch the records available in classicmodels.customers table to Redis cache when the keys count is 0.

Below is the snippet of **the index.php** script:

```
<?php

    // Initiate redis instance
    $redisObj = new Redis();

    /* Functions for redis operations starts here   */

    function openRedisConnection( $hostName, $port ) {
        try{
            global $redisObj;

            // Opening a redis connection
            $redisObj->connect( $hostName, $port );

            return $redisObj;
        }catch( Exception $e ) {
            echo $e->getMessage();
        }
    }

    openRedisConnection( 'localhost', 6379 );

    // Returns the number of keys in the currently selected database
    $objects = $redisObj->dbSize();
    echo "DB size: ". $objects. "\n";

    if ( $objects == 0 ) {
        // Calling getCustomerDetails.php script
        echo "Fetching data from customers table and save into redis cache\n";
        include '/opt/thesis/getCustomerDetails.php';
    } else {
        echo "DB size is not empty\n";

        // Get list of all keys in redis. This creates an array of keys from the redis-cli output of "KEYS *"
        $allKeys = $redisObj->keys('*');

        // Sort Keys alphabetically
        sort($allKeys);

        echo "Displaying all keys in redis:\n";
        foreach ($allKeys as $key) {
            $value = $redisObj->get($key);
            echo "Key: $key --> Value: $value\n";
        }
    }

    $executionTime = microtime(true) - $_SERVER["REQUEST_TIME_FLOAT"];
```

Fig 16   index.php script

- **getCustomerdetails.php**

This script is called only when the keys count in redis is 0. In this script, an SQL query (select * from customers) is executed and the results are then stored in Redis cache.

```
<?php
    /* Functions for redis operations starts here   */

    function setValueWithTtl( $key, $value, $ttl ) {
        try{
            global $redisObj;

            // setting the value in redis
            $redisObj->setex( $key, $ttl, $value );
        }catch( Exception $e ) {
            echo $e->getMessage();
        }
    }

    function getValueFromKey( $key ) {
        try{
            global $redisObj;

            // getting the value from redis
            return $redisObj->get( $key );
        }catch( Exception $e ){
            echo $e->getMessage();
        }
    }

    function executeQuery( $query ) {
        global $redisObj;

        // Create connection
        $mysqli = new mysqli( 'localhost', 'root', 'root', 'classicmodels' );

        if ( $mysqli->connect_errno ) {
            echo "Failed to connect to MySql: "."(".mysqli_connect_error().")".mysqli_connect_errno();
        }

        $result = $mysqli->query( $query );

        if ( $result ) {
            echo "Execution of SQL query was successfull\n";
            $num_rows = mysqli_num_rows($result);
            echo "SQL query returned $num_rows Rows\n";

            if ( mysqli_num_rows($result) == NULL ) {
                echo 'No results found.';
            }

            $resultArray = array();

            while ( $rows = mysqli_fetch_assoc($result) ) {
                $key = "sql_cache_". $rows['customerNumber'];
                // echo $key;
                $resultArray[] = $rows;

                // Inserting the value into redis cache
                // setValueWithTtl( 'sql_cache_' . $rows['customerNumber'], json_encode($rows), 3600);
                setValueWithTtl( $key, json_encode($rows), 3600);
            }

            $objects = $redisObj->dbSize();
            echo "DB size: ". $objects. "\n";
        }

        return $resultArray;
```

```
[ec2-user@ip-172-31-0-198 thesis]$ php index.php
DB size: 0
Fetching data from customers table and save into redis cache
Execution of SQL query was successful!
SQL query returned 122 Rows
DB size: 122
Execution time : 0.0050139427185059 seconds
[ec2-user@ip-172-31-0-198 thesis]$
```

Fig 17   getCustomerdetails.php script

The above clearly describes the output of the index.php, fetching the results from the SQL query and saving it into the Redis cache. Then another script getCustomerdetails.php checks the key count and executes the SQL statement. The select statement can be verified from the MySQL, also we can verify the key_value pair being loaded into the Redis cache with the execution of the script. The same can be viewed from the Redis command line (Redis-CLI) interface.

The cache layer introduced for this project holds the temporary data store, which stores the results of SQL queries and is always faster than the application making a direct connection to the database. The application first make a check in local cache based on a key pattern (say Alpha). If the data is available, then it sends the details to the application. But in case of cache miss, then the application make a check on remote cache and subsequently make a direct connection to database when data not available in both caches (local and remote). At a high level, every request response is either cached, or retrieved from the cache, and as a result, the load to our server and database is reduced.

Here we are calling getCustomerdetails.php - you should first check in the Redis to see if there is any data already cached. You can check this by using EXISTS command in Redis. If a particular key exists, it means the data is cached. So, we should just go ahead and fetch it and send it to the user. Otherwise, you should hit the MySQL DB, fetch data & serve the user and then store it in Redis for future access. You can also set an expiry time while storing a key in Redis in which case the key will be deleted after the specific period.

5.1.2 Script -2: searchCache.php

SearchCache.php is written to verify different scenarios/use cases to calculate the response time of cache (local, remote and the Primary Database) to return a record from Database.

The script searches for a key pattern “ALPHA” in accordance with various use cases. The use cases are listed below:

The user tried to search for pattern Alpha

- Initially both the local and remote cache are kept empty.
- User tries to search for pattern/Keyword from the local cache when Data exist in the local cache and Remote cache is kept empty.
- When the key is deleted from local cache and the remote cache is empty, then user tried to search for pattern Alpha
- Data exists in the remote cache but not in local cache.
- Data is deleted from both local and remote cache and then user tried to search for pattern Alpha, here the updating request is sent to the Primary Database for updating of records in both local and remote cache.
- After data exists in both local and remote cache, the user tried to search for pattern Alpha

```
<?php

  if (empty($argv[1])) {
     echo "No arguments passed\n";
     exit;
  }

  $password = "dublin@ireland";
  $PATTERN = "$argv[1]";
  echo "Pattern to be verified = $PATTERN\n";

  // Initiate redis instance
  $redisObj = new Redis();

  /* Functions for redis operations starts here   */

  function openRedisConnection( $hostName, $port ) {
     try{
        global $redisObj;

        if ( $hostName == "localhost" ) {
           echo "Connecting local cache...\n";

           // Opening a redis connection
           $redisObj->connect( $hostName, $port );
        } else {
           echo "Connecting remote cache...\n";

           // Opening a redis connection
           $redisObj->connect( $hostName, $port );
           $redisObj->auth($GLOBALS['password']);
        }

        return $redisObj;
     }catch( Exception $e ) {
        echo $e->getMessage();
     }
  }

  openRedisConnection( 'localhost', 6379 );

  // Returns the number of keys in the currently selected database
  $objects = $redisObj->dbSize();
  echo "DB size of local cache: ". $objects . "\n";
```

```
  if ( $objects == 0 ) {
     // Calling getCustomerDetails.php script
     echo "Fetching data from data store and saving into Redis cache\n";
     $id = "ALL";
     include '/opt/thesis/backup/15122018/getCustomerToCache.php';
  } else {
     // Get the list of all keys/value matching the pattern in Redis (local cache)
     $it = null;

     do {
        // $redisObj->setOption(Redis::OPT_SCAN, Redis::SCAN_RETRY);
        $allKeys = $redisObj->scan($it,"*$PATTERN*",$objects);

        // Redis may return empty results, so protect against that
        if ( $allKeys != FALSE ) {
           // Sort Keys alphabetically
           sort($allKeys);

           echo "Displaying the keys/value from local cache for the matching PATTERN $PATTERN:\n";

           foreach ($allKeys as $key) {
              $value = $redisObj->get($key);
              echo "Key: $key --> Value: $value\n";
           }
        } else {
           echo "No key/value found in local cache for the matching PATTERN $PATTERN\n";
           openRedisConnection( '172.31.2.76', 6379 );

           // Returns the number of keys in the currently selected database
           $objects = $redisObj->dbSize();
           echo "DB size of remote cache: ". $objects . "\n";

           if ( $objects == 0 ) {
              // Calling getCustomerToCache.php script
              echo "Fetching data from data store and saving into Redis cache\n";
              $id = "ALL";
              include '/opt/thesis/backup/15122018/getCustomerToCache.php';
           } else {
              $temp = null;

              do {
                 // Get the list of all keys/value matching the pattern in Redis (remote cache)
                 $allKeys = $redisObj->scan($temp,"*$PATTERN*",$objects);

                 // Redis may return empty results, so protect against that
                 if ( $allKeys != FALSE ) {
                    // Sort Keys alphabetically
                    sort($allKeys);

                    echo "Displaying the keys/value from remote cache for the matching PATTERN $PATTERN\n";

                    foreach ($allKeys as $key) {
                       $value = $redisObj->get($key);
                       echo "Key: $key --> Value: $value\n";
                    }
                 } else {
                    echo "No key/value found in remote cache for the matching PATTERN $PATTERN\n";
                    echo "So fetching data from the data store and saving into Redis cache for the matching PATTERN
$PATTERN\n";
                    $id = $GLOBALS['PATTERN'];
                    include '/opt/thesis/backup/15122018/getCustomerToCache.php';
                 }
              } while ( $temp > 0 );
           }
        }
     } while ( $it > 0 );
  }
```

```php
<?php
  /* Functions for redis operations starts here */

  function setValue( $key, $value, $CHECK ) {
      try{
          global $redisObj;

          // setting the value in redis
          $redisObj->set( $key, $value );

          if ( $CHECK != 'ALL' ) {
              $redis = new Redis();
              $redis->connect('localhost',6379);
              $redis->set( $key, $value );

              // Returns the number of keys in the currently selected database
              $objects = $redis->dbSize();
              echo "DB size of local cache : " . $objects . "\n";
          }
      }catch( Exception $e ) {
          echo $e->getMessage();
      }
  }

  function executeQuery( $query , $CHECK ) {
      global $redisObj;

      // Create connection
      $mysqli = new mysqli( 'localhost', 'root', 'root', 'classicmodels' );

      if ( $mysqli->connect_errno ) {
        echo "Failed to connect to MySql: "."(".mysqli_connect_error().")".mysqli_connect_errno();
      }

      $result = $mysqli->query( $query );

      if ( $result ) {
        echo "Execution of SQL query was successfull\n";
        $num_rows = mysqli_num_rows($result);
        echo "SQL query returned $num_rows Rows\n";

        if ( mysqli_num_rows($result) == NULL ) {
          echo "No results found.\n";
        }

        $resultArray = array();

        while ( $rows = mysqli_fetch_assoc($result) ) {
            $key = "sql_cache_" . $rows['customerName'];
            // echo $key;
            $resultArray[] = $rows;

            // Inserting the value into redis cache
            setValue( $key, json_encode($rows), $CHECK );
        }

        $objects = $redisObj->dbSize();
        echo "DB size of local/remote cache : " . $objects . "\n";
      }

      return $resultArray;
  }

  // Main program starts here
  $CHECK = $GLOBALS['id'];
  echo "Argument received $CHECK\n";

  if ( $CHECK == 'ALL' ) {
    /* Fetch all data from customers table */
    $query = "select * from customers order by customerName";
```

Fig 18. SearchCache.php script

6 Evaluation

We have tried to analyze different scenarios/use cases to calculate the time taken by the user to query results from caches (local and remotely located).

Though we have evaluated different scenarios we will take into consideration mainly three to prove the relative decrease in the response time of the user

1. **The first case**: We have left both the caches [local and remote] empty which means there is no data cached. In this scenario, the user is trying to search for a particular keyword. At first, a script called index.php is triggered to establish the connection with the Redis cache and check for the key count. This script verifies the availability of data in the caches, upon not found connects to the MySQL database and updates both the caches. We have considered the response time as the time taken by the script to execute and return the response.

   So, for case one **time taken** is 0.0055859088 microseconds considered as T

2. **Second case**: User is trying to search for a Particular Keyword when it is missing from both the local and remote cache. In this scenario, the script searchCache.php calls another script (getCustomerToCache.php) connects to the local cache and search for the keyword and when not found connects to remote cache, when data is missing in the remote cache, it contacts the database and updates both remote and the local cache.

   Time taken is = 0.0053539276123047 microseconds considered as T1

3. **Third Case**: When the local cache is updated previously with the missing data.
   Time taken is = 0.0011179447174072 microseconds considered as T2

   To calculate the decrease in the response time we will simply subtract response time of cache when the keyword is missing in both local and remote cache from response time when data exist in local cache upon the previous update.

T1-T2 = 0.005353 – 0.00111 microseconds = 0.004243 microseconds / 4.24 s
Which is 80% times faster response rate

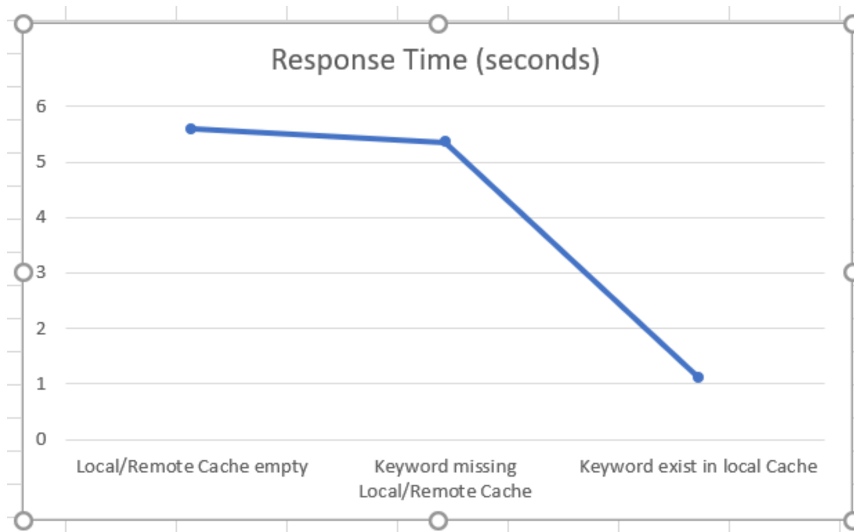

Fig 19. Graph representing Use Cases against response time of the query

```
[ec2-user@ip-172-31-0-198 15122018]$ php searchCache.php Alpha
Pattern to be verified = Alpha
Connecting local cache...
DB size of local cache: 122
Displaying the keys/value from local cache for the matching PATTERN Alpha:
Key: sql_cache_Alpha Cognac --> Value: {"customerNumber":"242","customerName":"Alpha
Cognac","contactLastName":"Roulet","contactFirstName":"Annette ","phone":"61.77.6555","addressLine1":"1 rue Alsace-
Lorraine","addressLine2":null,"city":"Toulouse","state":null,"postalCode":"31000","country":"France","salesRepEmployeeNum
ber":"1370","creditLimit":"61100.00"}
Execution time : 0.0011179447174072 seconds
[ec2-user@ip-172-31-0-198 15122018]$
```

Fig 20. SearchCache.php script showing the execution time for the query

#	Use Cases	Response Time (micro seconds)
1	The user tried to search for pattern Alpha Both local and remote cache is empty	0.0055859088897705
2	The user tried to search for a pattern from local cache first Data exists in the local cache Remote cache is empty	0.001086950302124
3	When the key is deleted from local cache and the remote cache is empty, then user tried to search for pattern Alpha	0.05279803276062
4	Data exists in the remote cache but not in the local cache	0.0035989284515381
5	Data is deleted from both local and remote cache and then user tried to search for pattern Alpha	0.0053539276123047
6	After data exists in both local and remote cache, the user tried to search for pattern Alpha	0.0011179447174072

Table 3. Use Cases and response time calculation

7 Conclusion and Future Work

Caching query results increases the scalability of the back-end database by serving a large part of the queries at the dynamic data cache. This reduces average response time when the back-end server is experiencing high load. It offloads origin backend system and provides better client response time.

Based on our analysis and the problems encountered, an efficient caching techniques to be proposed for an overall improvement of data accessibility and to reduce the query latency further.

The most problem with cache solution is – the data to be pushed to cache for the first time and this fetch is always slow. So a micro service to be developed in such a way that it will ensure the data is in the cache before the application request even for the first time. The future scope is to focus on the development of a micro services that can compute every cache entry, put them in a cache distributed key-value store and serve them with minimal latency. [3]

To avoid the number of cache misses, a customized micro service to be developed and implemented at or on the system of the database to directly detect changes to the MySQL database table data. This micro service should be monitoring the transactional information maintained by the database itself to determine when changes to the database occur. This service will interact with an API to pull all recent changes in the database and then store the results in Redis cache. During this, the necessary modules required will be loaded and also make a connection to the central database. Once the data is fetched, the query results were added to the cache by running Redis commands like dbSize, set, get etc...

In the current setup, the scripts developed for the thesis were executed from the command line to capture the results. But in future, some front-end development work should be carried out ensuring the backend scripts (searchCache.php, getCustomerToCache.php) were invoked automatically and accessible from the browser.

References

[1]. Gray, C., & Cheriton, D. (1989). Leases: an efficient fault-tolerant mechanism for distributed file cache consistency.

[2]. Tiago Macedo, & Oliveira Fred (2011). Redis Cookbook: Practical Techniques for Fast Data Manipulation O'Reilly media Publications, pp. 50

[3]. Martin Kleppmann (2012). Rethinking caching in web apps

[4]. Dedi Iskander Inan & Ratna Juita (2011). Analysis and Design Complex and Large DataBase using MySQL Workbench: International Journal of Computer Science & Information Technology (IJCSIT) Vol 3,

[5]. Li Yang & Li Cao (2016): The effect of Mysql workbench in teaching entity-relationship (ERD) to relational schema mapping: I.J.Modern Education and Computer Science, pp. 10-12

[6]. Songhuan Li, Hong Jiang & Mingkang Shi (2013).Redis-based Web Server Cluster Session Maintaining Technology: 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery, pp .1-2

[7]. Mysqltutorial.org (2018). How to load sample database into MySQL database server

[8]. M. A. Ramteke, Prof. S. S. Dhande & Prof. H. R. Vyawahare (2014). A Review on Query result Caching using dynamic data Cache: International Journal of computer science and Information Technologies, Vol 5, pp. 1-2

[9]. H.V chainani ,S.S kale,R.R Sarad & R.R Karwa (2018).Database Cache Management System: International Journal for Scientific Research & Development, Vol. 6,pp 1-2

[10]. Arnaldo Marulitua Sinaga & Poppy Sibarani (2015). The implementation of Caching Database to reduce query's response time: MATEC Web of conferences, pp 1-2

[11]. Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, Gustavo Alonso (2006). GlobeCB: Content-blind Result Caching for Dynamic Web Application: Technical Report IR-CS-022, pp. 1-5

[12]. Nathaniel D. Harward,San Francisco,Andrew R Geweke & Alexander Voskoboynik (2006) .Database Caching and invalidation using Database Provided Facilities For Query Dependency Analysis: United States Patent Application Publication,pp. 13-14