Determining Appropriate Cache-size for Cost-effective Cloud Database Queries

Ruchi Nanda
The IIS University
Jaipur, India

Swati V. Chande
International School of Informatics and Management
Jaipur, India

Krishna S. Sharma
The IIS University
Jaipur, India

ABSTRACT
Retrieving results from the cache is one of the prominent techniques to improve the query response time and reducing load on the back-end database servers. One of the important factors that influences the performance of cache system the most, is the size of the cache. In cloud-based systems, memory is scalable and hence, size of the cache is not a critical issue. However, when the cache is overpopulated with queries and their results, in that case, the query response time increases. This is due to the fact that time for searching the cache for the desired results increases. In this paper, an appropriate cache size is calculated in terms of the number of queries, for the database-size under consideration.

This paper also describes the set-up of Virtual Machine Creation (VMC) cloud, using Cloud Virtual Machine Creation (CVMC) algorithm. This facilitates the deployment of database in cloud-based systems. An appropriate cache-size for cloud-based system is determined through experimentation using Apache HBase.

Keywords
Caching, NoSQL datastores, NoSQL database, HBase, Cache-size, Cloud-based systems, cloud datastores, Query Response Time

1. INTRODUCTION
Caching is one of the techniques to reduce the processing time of the queries by storing them in memory. The data retrieval from cache is considerably faster than main memory or disks. Caching reduces the load on servers, which facilitates reduction in the overall response time of queries. It is beneficial if some data is accessed more frequently than the remaining data. In case of uniform access to data, caching is not beneficial [1]. If the cache exceeds its limit, cache replacement algorithm makes space for the new data by evicting old data.

In cloud-based systems, memory is scalable and inexpensive and therefore size of the cache is not considered to be a critical performance issue. Large number of results can be cached economically and the size of the cache and its eviction policy are therefore not critical [2]. However, if cache is overloaded with many queries and their results, it would be difficult to retrieve the results efficiently. Hence the question arises "What is the best-suitable cache-size in case of cloud-based systems?" In [3] the cache-based framework is proposed by the authors that caches query and results. A least recently used queries and their results are evicted, if the cache exceeds its limit, so that a new query and its results could be stored. This paper deals with how to decide the parameter 'cache-size' by conducting experiments.

The cache-size depends on the database-size [1]. For the database-size under consideration, the best suited cache-size can be determined in terms of the number of queries stored in it, by performing two sets of experiments on the underlying database. The cache-size is calculated meticulously by considering the following queries:

- Exact cache-hit queries
- Different percentages of cache-hit queries.

The first contribution of this paper lies in the development of VMC cloud, which allocates infrastructure to the client, by means of a virtual machine. The second contribution, which is equally important, lies in determining the best suitable cache-size for cloud database queries.

The paper is organized as follows:

Section II contains the study of IaaS cloud development. In section III set up of the cloud is explained that includes description and implementation of the CVMC algorithm. The concept of caching is discussed in Section IV followed by related work in Section V. The experimental setup, experiments conducted and results obtained are discussed in Section VI and finally the last section contains the conclusions drawn from the present work.

2. CLOUD SET-UP
With the emergence of cloud computing technology, virtualization becomes a key component in the cloud-based systems. Cloud offers infrastructure, platform and software as three basic services on subscription basis to the clients. These services are provided using virtualization technology, which is implemented by hypervisor or Virtual Machine Monitor. Hypervisor distributes the hardware resources among all the virtual machines.

Infrastructure as a Service (IaaS) is the most important service for accessing the cloud resources, such as memory, network, processing, computing server, storage. In the present work, a Virtual Machine Creation (VMC) cloud has been developed, which is responsible for allocating virtual machine to the client. The Cloud Virtual Machine Creation (CVMC) algorithm, described below, is used for this purpose.

The CVMC algorithm accepts 'UserID' and 'password' as input from the client. If the client is an authorized customer of cloud resources, it allocates Kernel-based Virtual Machine (KVM) image on cloud server.

The client assigns a name to the virtual machine image and also specifies size of the memory, number of virtual CPUs and hard disk space as desired. The VMC cloud automatically creates a virtual machine image on the server, based on the client's requirements. The client can access the machine using

Virtual Network Computing (VNC). Algorithm 1 describes CVMC algorithm.

Algorithm 1: CVMC Algorithm

```
Input:  UserId
        Password
        vmimage_name
        size_of_ram # in MB
        No_of_cpu
        hardDisk_size # in GB

Output: KVM virtual machine image
-----------------------------------------------------------------------------------------------------------------------
Begin
While connect to the server:
        Check the login details of the client
        If client is valid:
                Create a KVM client as vmimage_name
                Specify the server location containing an installable operating system image
                Create disk image space on hard disk as hardDisk_size
                Set path to the assigned disk space to create virtual machine image
                Configure virtual processors as No_of_cpu
                Allocate memory as size_of_ram
        EndIf
End While
End
```

For creating VMC server, some prerequisites [4] are required to be accomplished on the machine, which are as follows:

- Installation of virtualization packages on the system
- Service "libvirtd", for managing different activities of the server
- FTP server, for keeping the operating system files, that are to be installed on the VM
- Creation of kickstart file for Network Installation server, that performs partitioning according to the clients requirements and install OS on client VM.

CVMC algorithm is implemented in Python and the cloud environment is set as shown in Figure 1. It consists of the host machine, with the operating system - Red Hat Enterprise Linux 6 (RHEL6). The hypervisor, Kernel-based Virtual Machine (KVM) is chosen to be installed. The standalone Apache HBase database is deployed on top of hypervisor.

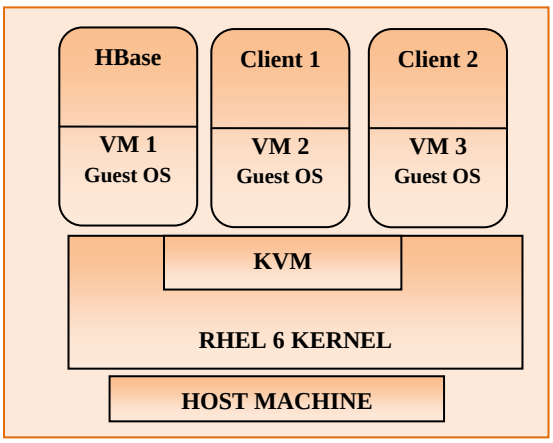

Figure 1: Cloud Environment for the Study

3. PRIOR RELATED WORK ON CACHING

Caching systems have been proposed by the researchers to retrieve data rapidly. Cache stores data, which can be a web page, fragment of a web page, query or results. This paper deals with the caching of query and its results. Size of the cache is one of the important factors that influence the caching mechanism. If the cache is big in size, it will take more time to search results. The small sized cache causes frequent execution of cache replacement algorithms. In both of the cases, performance of the cache is degraded.

Cache-size is an important parameter for determining the caching efficiency and different caching frameworks have been proposed in literature to built for peer-to-peer systems, location-based services, web information retrieval, mobile computing, and cloud databases ([5]-[9]). Packer [1] observed that the optimal buffer cache size is between 10% and 15% of the database size and in some cases up to 20% of the database size. A large number of results can be cached economically and hence the cache size is not a critical issue [2]. Ding et al. [7] varied the cached-entries from 5% to 20% of the total number of moving objects. They experimentally proved that when the percentage of cached-entries is 20% of total number of objects, then the disk page accesses is almost negligible. Chockler et al. [8] preferred to set the cache size equal to the total memory allocated for data cache by the service provider. Ilayaraja et al. [10] set the cache-size to 10% of the database size, which was 50 in number of objects. Dong et al. [9] varied the cache-size based on system administrator's experience.

However, cloud-based caching does not require a limit on the cache capacity, as the size of the cache is infinite. A cache policy was designed with the aim to minimize the cost of a cloud-based system, instead of maximizing the hit ratio [11]. As the unlimited cache size is impractical in deployment scenarios, Kiani et al. [12] focused on cache replacement policies. The authors considered size of the cache in terms of the number of items.

The present paper considers the cache-size in terms of the number of queries. Experiments were conducted for determining an appropriate cache-size for the available database-size. Cache-size was varied from 10% to 30% of the size of the database.

4. INVESTIGATION OF CACHE-SIZE

Caching is one of the techniques which reduces load on database servers and improves query response time. The performance improvement provided by the cache is measured in terms of cache-hit and cache-miss ratio. The cache-hit ratio is defined as the percentage of all the requests handled by cache [13]. It is calculated by using the relation:

$$\text{Cache Hit Ratio} = \frac{\text{Requests handled by cache}}{\text{Total number of requests}} * 100 \quad (1)$$

Cache provides high performance improvement in the following two cases: First, if the ratio of cache hit is high and second, if the service time difference between a cache hit and miss is also high [13]. The calculation of performance improvement provided by the cache is made by using the Eq. 2:

$$PI = CHT - CMT \quad (2)$$

where, PI is the performance improvement provided by the cache.

CHT is the service time of a cache hit

CMT is the service time of a cache miss

The objective of the paper is, to evaluate the cache size where, all the queries approaching the cache are answered in reasonable time. Since, the queries can hit as well as miss the cache, the best-suitable cache-size is calculated after examining response time of the exact 100% cache-hit queries and varying percentages of cache-hit and cache-miss queries.

For determining cache-size, client and server side scripts are written using Python socket programming [3]. The client script generates client queries from a text file, which are sent to the server socket, where the query is checked in the cache. If it is found there, then its results are retrieved from cache, otherwise the results are retrieved from the underlying cloud HBase datastore, using HappyBase API. New query and its results are stored in the cache for future access. In case, cache exceeds its maximum limit, the least recently used query and its result are evicted.

5. EXPERIMENTS & RESULTS

5.1 Experimental Set-up

The experiment was performed on VMC cloud server having Intel Core 2 Duo processor, 2.40 GHz CPU and 4 GB RAM, running RHEL 6 operating system. The underlying datastore used for testing the framework was Apache HBase-0.94 [14]. HappyBase-0.3 version was used to interact with Apache HBase. It offers a rich set of Pythonic APIs, on using which a Python program is executed on Apache HBase.

5.2 Experiment Evaluation

In order to evaluate cache-size, a subset of 1000 records of popular dataset, BookCrossing (BX) prepared by Cai-Nicolas Ziegler, DBIS Freiburg [15], was used and converted into HBase schema.

Two experiments were conducted to investigate the best suitable cache-size:

Experiment 1: To investigate the best suitable cache-size when Exact Cache-hit queries approach cache

The purpose of this experiment was to determine experimentally the suitable cache-size, by using a set of 100% cache-hit queries. The cache-size, ranging from 10% to 30% of the database size is taken, in order to observe the variations in query response time with respect to cache-size. A set of 10 queries that are 100% cache-hit, are prepared. The experiment is executed first, for cache-size having 100 queries, when the cache is full. The test is run 3 times, so as to minimize the environment factors [16]. For each run of the test, the response time of the queries is calculated. Similarly, for each cache-size, the experiment is conducted. Table 1 shows the response time of the queries in run one, two and three for different cache sizes.

Table 1: Query Response Time of Exact Cache-hit Queries using Different Cache sizes in Run 1, 2 and 3 of the experiment

	100	150	200	250	300
QRT (Run 1)	0.00029	0.00026	0.00028	0.00035	0.00036
QRT (Run 2)	0.00031	0.00028	0.00027	0.00035	0.00034
QRT (Run 3)	0.00026	0.00024	0.00028	0.00031	0.00047

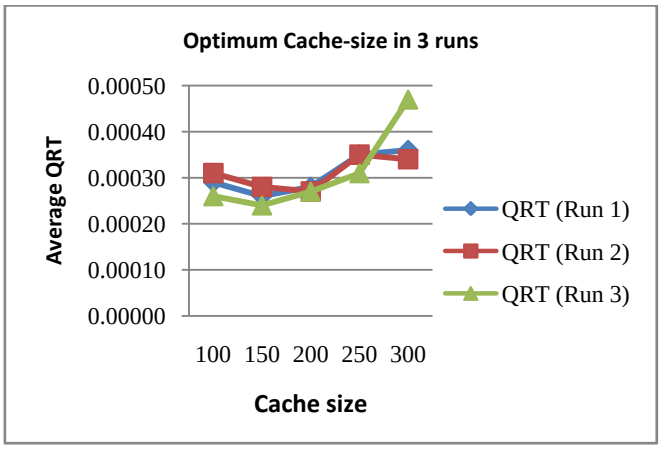

Figure 2: Query Response Time of Exact Cache-hit Queries using different Cache-sizes

Figure 2 shows the graphical representation of the query response time of exact cache-hit queries in each run of the experiment. The QRT is higher comparatively for 100 queries in all the runs of the experiment. For cache-size of 150 queries, it is reduced in all the three runs. Afterwards it starts increasing, but thereafter it increases in the first and third run, whereas in second run it is observed to decrease up to 200 cache-size and then increases for higher cache-sizes.

The average query response time of the queries is calculated from the results of three runs. Table 2 shows the average query response time of the queries processed in the cache. It also depicts the relative change (RC) in percentage, in the QRT on cache-based system as compared to the QRT taken on non-cache datastore. The average QRT of non-cache HBase is 0.11400 seconds.

Observations from Experiment 1:

The query response time is slightly higher at cache-size of 100. At cache size 15%, it is reduced and remains almost constant till cache size is 20%. Thereafter, it starts increasing

from cache size 20% to 25%. There is almost a reduction of 99.76% in the response time when the cache size changes from 150 to 200. Figure 3 depicts observations pictorially. The x-axis of this diagram represents the different cache-sizes and the y-axis shows the average query response time. The lowest average QRT is observed at cache-size 150 and 200. After that, it starts increasing for cache-sizes 250 and 300.

Table 2: Average Query Response Time for Different Cache sizes

	Cache-size: 100 (10%)	**Cache-size: 150 (15%)**	**Cache-size: 200 (20%)**	**Cache-size: 250 (25%)**	**Cache-size: 300 (30%)**
Avg. QRT : with cache (sec.)	0.00029	0.00026	0.00027	0.00034	0.00039
RC in QRT for cache as compared to non-cache(in %)	-99.75	-99.77	-99.76	-99.70	-99.70

Hence the cache size of 150 (i.e., 15%) or 200 (i.e., 20%) of the database-size can be chosen for cloud-based systems. This experiment considered a set of exact cache-hit queries. But, practically it is not possible that 100% cache-hit queries approach the cache. Hence, in the following experiment the cache-size for different percentage of cache-hit queries is investigated.

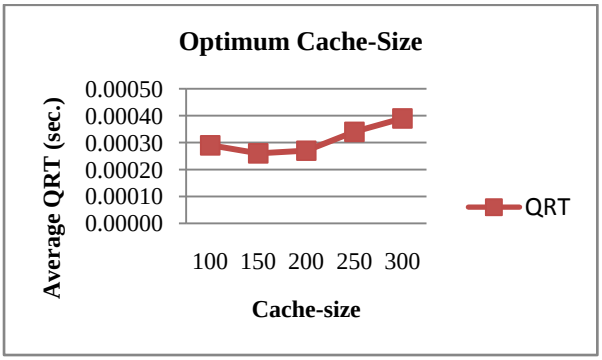

Figure 3: Average Query Response Time of Exact Cache-hit Queries, using different Cache-sizes

Experiment 2: To investigate the best suitable cache-size when Different Percentages of Cache-hit queries approach cache

The purpose of this second experiment was to assess the cache-size obtained in the first experiment, using the queries having different percentages of exact cache-hit, i.e., from 100% to 10%. Hence, the experiment that meticulously checks the differences in the query response time, was executed on the tests conducted on three different cache-sizes 150, 200 and 250. The cache-size, for which the overall query response time is the least, is considered as an appropriate cache-size for cloud-based systems.

For the experiment taking cache-size of 150 queries, the cache-hit queries ranges from 100% to 10%. 10 queries are chosen to warm up the query cache. For each percentage of exact cache-hit, 5 query sets are prepared, except for 100%. For 100% cache-hit queries, only 1 query set is prepared. In all 46 query sets were prepared.

Each query set contains 10 queries that match in percentage with some exact hit and the remaining in the percentage with cache miss. In all 460 queries were processed by the framework having cache-size 150. The experiment was run 5 times for each percentage of exact cache-hit queries, so as to minimize environmental factors. The system was restarted in each run of the experiment, so that there will be no influence of operating system and internal database cache [17]. The cache starts populating from 90% cache-hit queries and when reached 20% cache-hit, the least recently used algorithm starts execution.

The query response time is calculated in each run of the experiment for different cache-hit percentages. Similar procedure is followed for conducting experiments using the cache-size 200 and 250. In cache having size 200, cache eviction started from the first run of 10% cache-hit run. However in, cache having size 250, no replacement was observed. In all, 1380 queries were processed in the experiment.

Table 3 shows the average query response time and the relative change in QRT, in percentage, from cache-size 150 to 200 and from cache-size 200 to 250 for each percentage of cache-hit queries. It provides a comparative analysis on the different percentages of exact cache-hit queries.

Table 3: Average Query Response Time and Percentage Change from cache 150 to 200 and for Percentage Change from 200 to 250.

	100%	**90%**	**80%**	**70%**	**60%**	**50%**	**40%**	**30%**	**20%**	**10%**
QRT(150)	0.00004	0.08741	0.09525	0.10081	0.10786	0.10877	0.11966	0.11634	0.12420	0.12232
QRT(200)	0.00003	0.08744	0.09732	0.10399	0.10727	0.11736	0.12483	0.11854	0.12295	0.12273
QRT(250)	0.00002	0.08832	0.09483	0.10776	0.10949	0.11497	0.11958	0.11942	0.12363	0.12367
Relative change in % from 150 to 200	-25.00	0.03	2.17	3.15	**-0.55**	7.90	4.32	1.89	**-1.01**	0.34
Relative change in % from 200 to 250	-33.33	1.01	**-2.56**	3.63	2.07	**-2.04**	**-4.21**	0.74	0.55	0.77

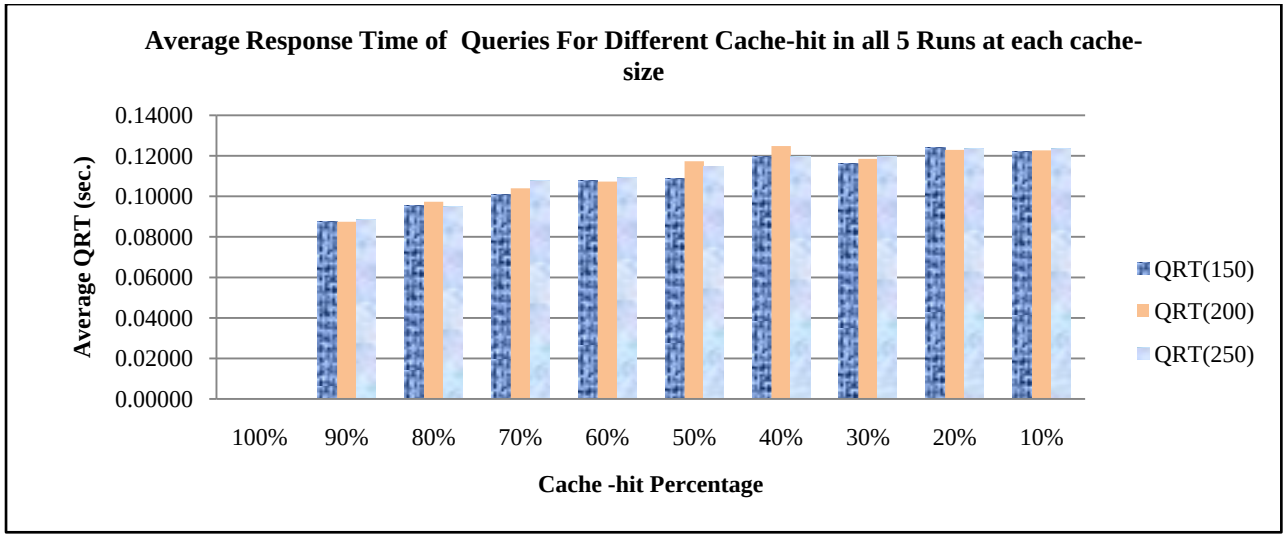

Figure 3: Average Query Response Time of Different Cache-hit Percent Queries using different Cache-sizes

Observations from Experiment 2:
It is observed that the average query response time (QRT) almost linearly increases as the percentage of exact cache-hit queries decreases from 100% to 40%. The reason is, with the increase in cache-miss percentage, the cache takes more time to retrieve the cache-miss query results from the database for the first-time. For cache-hit queries below 40% the QRT remains almost constant.

The change in query response time, from cache-size 150 to 200 and then from 200 to 250, shows, that query response time increases when the size of the cache increases. For cache-size 150, the average query response time is lower in every percentage of cache-hit, except for 60% and at 20%. This reduction is due to the fact that cache eviction starts from 20% cache-hit, and hence, it takes slightly higher time as compared to other cache-size.

The average QRT for cache-size 150 is comparatively less as compared to the average QRT in case of cache sizes 200 and 250. The percentage difference between cache-sizes 150 and 200 shows that only at three hits QRT is decreased. From cache-size 200 to 250, it is observed that QRT is increased almost in every percentage hit.

Hence, this experiment shows that cache-size of 15% of the database may be considered as the best-suitable cache size for cloud-based systems having the database-size under consideration. The average values of query response time in all the five runs of the experiment for the three cache-sizes are plotted in the graph shown in Figure 3.

The figure 3 depicts that average response time is very small for 100% cache-hit and starts increasing with the decrease in the percentage of cache-hit queries. It is almost constant for cache-size 15% of the database-size.

6. CONCLUSIONS

In cloud-based systems, memory is scalable and therefore it is not a critical issue. However, if the cache becomes overloaded with queries and their results, the search time increases. This results in performance degradation of the cache system. This paper provides a technique to find out the best-suitable cache-size in terms of the number of queries. The performance of cloud database with cache is most cost effective when the cache-size is kept at about 15% of the database size. The work can be further extended by including parameters like:

- types of queries: the considered operation in the present work is scan queries. The queries involving 'AND' operator are treated as the most expensive queries. Hence, these are more suitable candidates for caching.
- database-size: as the size of the data is enormous in almost all application domains, therefore, larger database size can be taken up for further investigations.
- query-set size: the number of queries in each set can be varied to obtain optimum performance.

The paper also presents CVMC algorithm, using which clients can access the cloud infrastructure through virtual machine. The clients can further install software or deploy database on cloud-based systems.

7. REFERENCES

[1] A. N. Packer 2001. Configuring and tuning databases on the Solaris platform. Prentice Hall PTR.

[2] X. Long, and T. Suel 2006. Three-level caching for efficient query processing in large web search engines. World Wide Web, 9(4), 369-395.

[3] R. Nanda, K. S. Sharma, S. Chande 2016. Enhancing the Query Performance of NoSQL Datastores using Caching Framework. International Journal of Computer Science and Information Technologies, Volume 7, Issue 5 (September-October 2016), 2332-2336, 0975-9646.

[4] Red-Hat Inc., libvirtd(8) - Linux man page. Online Available: http://linux.die.net/man/8/libvirtd.

Retrieved on 18 July 2016.

[5] K. Raichura, N. Padhariya, and K. Atkotiya 2014. Cache-Based Query Optimization In Mobile Ad-Hoc Networks," International Journal of Technology Enhancements and Emerging Engineering Research, vol. 3(2), pp.226-232, 2014.

[6] O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi 2004. A peer-to-peer framework for caching range queries. In Proc. Data Engineering. IEEE, pp. 165-176.

[7] H. Ding, A. Yalamanchi, R. Kothuri, S. Ravada, and P. Scheuermann 2006. QACHE: query caching in location-

based services. Progress in Spatial Data Handling. Berlin, Heidelberg: Springer, pp. 99-116.

[8] G. Chockler, G., Laden, and Y. Vigfusson 2010. Data caching as a cloud service. In Proc. of the 4th International Workshop on Large Scale Distributed Systems and Middleware ACM, pp. 18-21.

[9] F. Dong, K. Ma, and B. Yang. 2015. Cache system for frequently updated data in the cloud. WSEAS Transactions on Computers, vol. 14, pp. 163-170.

[10] N. Ilayaraja, F. M. Jane, I. Thomson, C. V. Narayan, R. Nadarajan and M. Safar 2011. Semantic Data Caching Strategies for Location Dependent Data in Mobile Environments. In International Conference on Digital Information and Communication Technology and Its Applications (pp. 151-165). Springer Berlin Heidelberg.

[11] N. Le Scouarnec, C. Neumann and G. Straub 2014. Cache policies for cloud-based systems: To keep or not to keep. In IEEE 7th International Conference on Cloud Computing (pp. 1-8). IEEE.

[12] S. L. Kiani, A. Anjum, N. Antonopoulos, K. Munir and R. McClatchey 2012. Context caches in the Clouds. Journal of Cloud Computing: Advances, Systems and Applications, 1(1), 1.

[13] D. Wessels 2001. Web caching, O'Reilly Media, Inc.

[14] Userguide HappyBase [Online]Available: http://happybase.readthedocs.io/en/latest/user.html

[15] C. N. Ziegler, S. M. McNee, J. A. Konstan and G. Lausen 2005. Improving recommendation lists through topic diversification. In Proceedings of the 14th international conference on World Wide Web, ACM, 22-32.

[16] M. Perrin 2015. Time-, Energy-, and Monetary Cost-Aware Cache Design for a Mobile-Cloud Database System. Doctoral dissertation, University of Okalahoma.

[17] B. J. Sandmann 2014. Implementation of a Segmented, Transactional Database Caching System. Journal of Undergraduate Research at Minnesota State University, Mankato, vol. 6(1), pp. 21.