ISSN No. 0976-5697

Volume 8, No. 3, March – April 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Determining the Effect of Similar Queries in Cache-based Cloud Datastores

Ruchi Nanda
Sr. Asst. Prof., Department of CS & IT
The IIS University
Jaipur, Rajasthan, India

Prof. Swati V. Chande
Head, Department of CS
International School of Informatics & Management
Jaipur, Rajasthan, India

Prof. Krishna S. Sharma
Advisor, The IIS University
Jaipur, Rajasthan, India

Abstract: Caching is one of the techniques that facilitates reduction in number of database accesses, load of queries on the database server, speeds up the data retrieval process, and also reduction in the query processing time and overall response time. With the increase in size of data in cloud environment, the cloud datastores are expected to manage static data as well as dynamic data efficiently. Static databases are crucial to maintain, as all the decisions and analyses are made on the basis of the data available in these databases. A caching scheme is utilized with the aim to enhance the performance of static cloud database queries. The processing time is evaluated over different percentage of cache-hit queries. Experiments performed on HBase show that the time to retrieve the results of cache-miss queries gradually decreases as the cache starts getting populating with more cache-miss queries. There is also reduction in the time taken by the queries when directly executed in succession on non-cache based system. Therefore, an experiment is conducted to check the effect of similar cache-hit queries when run in succession on cache based system. The comparisons are made against the non-cache based system and it shows that successive trips to cache, lead to the performance improvement in cache-based datastore over non-cache based datastore by 48.64%, in terms of the cumulative processing time.

Keywords: Caching, column-based datastores, HBase, Cache-size, Cloud-based systems, cloud datastores, Static Cloud Datastore, Processing Time, Query Response Time

I. INTRODUCTION

With the increase in size of data in cloud-based systems, it is crucial to maintain static databases and dynamic databases. The data generated by social networking sites, forums, blogs and other information sharing websites is increasing very rapidly leading to creation of static data in large volumes. The importance of static data is therefore, paramount as several decisions and analysis are made on the basis of this data. It is hence, a major main concern for organizations to effectively maintain static data.

This study has been carried out with the objective to enhance the performance of static cloud database queries which handle multiple client queries. It stores query and its results on the database-tier of the server. Clients directly access the cache for their query results and only if the results are not found in cache, then it is retrieved from the underlying static datastore. Hence, it reduces the number of direct interactions with the database. When the cache exceeds its maximum size, the least recently used (LRU) replacement algorithm executes, that replaces the least recently used query and its result with the new query and its result. The cloud environment is set by developing a Virtual Machine Cloud (VMC) cloud using Cloud Virtual Machine Creation (CVMC) algorithm [1], which gives a virtual machine access to client on cloud server.

The paper is organized as follows:

Section II contains related work on caching. Section III describes the role of caching in enhancing query performance and the experimental setup, evaluation parameters, experiments conducted and results obtained are discussed in Section IV and finally the last section contains the conclusions drawn from the present work.

II. REVIEW OF RELATED WORK

Caching mechanisms have been developed for caching not only just static data and dynamic data but several authors have also worked on caching part of the database. These authors vary the parameters such as type of queries, database, cache-size, page replacement algorithms, so as to obtain either reduction in the query response time or load on the database server.

Authors mostly focused on range queries [2], and some of them on the most frequently accessed queries [3][4]. Most of the authors used the relational database management system like Oracle and key-value stores as the platform. Fan et al. [3] considered that node selection is based on random data partitioning across all cluster nodes, instead of correlated partitioning, as in case of Bigtable or HBase systems. The caching techniques are applied to structured databases by [4]-[6] or are distinctly provided as a cloud service [7]. The authors believed that data cache should become a cloud service and shared by multiple tenants. Chockler et al. [7] designed a simple multi-tenant data cache scheme, named BLAZE, based on the CLOCK replacement algorithm that uses utility-based cache partitioning between tenants.

Updating the cache data is vital if the database tends to change dynamically. Keeping in mind, the updated data as well as maintaining the indexing and consistency of the data, Dong et al. [4] proposed a cache system for frequently updated data in the cloud (CSFUD). The techniques used by them were applied to update data queries on document-oriented NoSQL databases, and relational databases like Oracle and DB2.

The work has been carried on static-data caching by [6][8] and on dynamic-data caching by [4][9]. In cloud-based systems as the size of the data is emerging, for important business analysis, static database performance is much more important. Caching algorithms [3][4][5][7] proposed, mainly focused on SQL-based systems and document-based stores. There is no prior study to our knowledge that has applied caching mechanism on column-based stores. Hence, in the area of query processing, caching is one of the fields where there is scope of further research.

The related studies do not consider query processing time as the major factor, instead focus majorly on load balancing and cost factor. As per authors, Kim et al. [10] and Karde and Thakare [11], in highly distributed environment, cost is not the dominant factor. The query response time is the major issue in improving the performance of timely retrieval of data [12][13]. These studies have acted as a motivation to cache queries and results to reduce the processing time of multiple queries.

III. Caching In Enhancing Query Performance

Caching is one of the techniques that reduces the load on the back-end servers as well as reduces the query processing time. The caching framework [14] proves that caching is a viable alternative in NoSQL datastores. In this paper, cache index is created for the queries present in the cache. The server receives the queries from the client using a text file, where it is looked up in an index created for the cache. If the query is found in the index, the query results are directly retrieved from the cache and returned to the client. In this case, the query is processed by the cache. If the query is not found in the index, it is sent to the underlying datastore for the processing. In this case, query is processed and evaluated by the underlying datastore and the query results are then, sent to the client. The query and its results are also sent to the cache, where it is stored and entry in the index is made for that query.

IV. Experiment & Results

A. *Experiment Setup*

The experiment was performed on VMC server [2]. The underlying datastore was Apache HBase-0.94. Python program is executed on HBase using HappyBase-0.3 version. The experiment was run in a steady environment where the system had one virtual machine running.

B. *Evaluation Parameters*

For the evaluation of the caching system different parameters have been used which are discussed below:

The performance improvement provided by the cache is measured in terms of cache-hit and cache-miss ratio [15]. The response time of the queries are calculated by the summation of CPU time, I/O time and the communication time by using (1).

$$\mathrm{RT} = \mathrm{CPU} + \mathrm{I/O} + \mathrm{CommunicationTime} \quad (1)$$

In this, network latency time is not considered, hence communication time is not included for the calculation of query response time. The average response time (ART) of the system is calculated by the difference between the query issuing time and the result receiving time. Raichuria et al. [10] used (2) for the calculation of ART.

$$\mathrm{ART} = \frac{1}{NC}\sum_{q=1}^{NC}(\mathrm{tb}-\mathrm{te}) \quad (2)$$

where, tb is the query-issuing time, te is the time of the result reaching the query-issuer and NC is the total number of completed queries.

The time taken by the CPU to process the queries is calculated by (3) [10]:

$$\mathrm{CPT} = \frac{1}{NC}\sum_{q=1}^{NC}(\mathrm{tcp}) \quad (3)$$

where, tcp is the time taken by CPU to process queries and NC is the total number of completed queries.

The cache-hit query is directly retrieved from the cache and hence the processing time of the query is reduced. The disk is not accessed and the database engine does not execute the cache-hit query, the results are directly retrieved from the memory (i.e. cache). This results in the reduction of CPU time, which in turn, decreases the processing time. If the query is cache-miss, the result is retrieved from the database, after the query completes all the phases of query processing including its execution by the database engine.

In this paper, the processing time in case of cache-hit and cache-miss queries are calculated to assess the behavior of cache with respect to the data retrieval time. The paper also focuses on the calculation of query response time. It includes all the overheads of cache system, such as time taken to make the query as the most recently accessed query and the time taken to build the result to a form that can be sent to the client. This is done to compare the performance of the database system having cache with non-cache database system. Average query response time and average cumulative processing time are used interchangeably in the paper.

The details of processing time and cumulative processing time are given below:

Processing Time (PT) in case of Cache-hit query: The time taken to process the cache-hit query is, the time to look in the index and retrieve the query result from the memory i.e. cache. It is called as cache-retrieval time and denoted by CRT. It is calculated by using (4):

$$\text{Average Processing Time} = \sum_{i=1}^{n}\mathrm{CRT} \quad (4)$$

Processing Time in case of Cache-miss query: It is the summation of the time taken by the system to search the index of the cache and the time taken to retrieve the result from the database. It is calculated by using (5).

$$\text{Average Processing Time} = \sum_{i=1}^{n}(\mathrm{CRT} + \mathrm{DBRT}) \quad (5)$$

where, CRT includes only the time to search a query in an index.

Cumulative Processing time (CPT) in case of cache-hit: It comprises of three components:

- Cache-retrieval time (CRT): It is the time taken by the incoming query to look in the index and the time taken to retrieve the result from the cache.
- Time taken to Send (TTS): The time taken by the cache system to build the result to a form that can be sent to the client socket.
- Background Pointer Time (BPT): The time taken to change the pointer of the current query and its result to the rear of the cache, to make it as the most recently accessed query. This

process occurs after sending results to the client. Hence, this is treated as a background time. The formula for its calculation is given by (6).

$$\text{Average Cumulative Processing Time} = \sum_{i=1}^{n} (\text{CRT} + \text{TTS} + \text{BPT}) \qquad (6)$$

where, CRT includes the time to search a query in an index and then the time taken to retrieve the answer from cache.

Cumulative Processing time (CPT) in case of cache-miss: It comprises four components:

- Cache-retrieval time (CRT): The time taken by the incoming query to search the index maintained for cache.
- Database Retrieval Time (DBRT): The time taken by the cache system to call database and retrieve the result from it.
- TTS: The time taken by the caching system to build the result to a form that can be sent to the client socket.
- Cache-Insertion Time (CIT): The time taken to append the cache-miss query and its result in the cache. This process occurs after sending results to the client. Hence, this is treated as a background process.

The formula for its calculation is given by (7).

$$\text{Average Cumulative Processing Time} = \sum_{i=1}^{n} (\text{CRT} + \text{DBRT} + \text{TTS} + \text{CIT}) \qquad (7)$$

where, CRT includes only the time to search a query in an index.

The different sets of times are calculated separately to thoroughly assess the behavior of cache. It helps in identifying the areas that require more time.

C. *Experiments*

The experiments have been carried out keeping in view the objective of evaluating the performance of the caching system in the chosen environment and also in practice. The description is as follows:

Experiment No.1: To evaluate the performance of cache-based datastore against non-cache column datastores

The primary objective of this experiment is to evaluate the performance of the datastore having cache solution against non-cache based column datastores, in terms of time. Previous research shows that caching reduces query processing time in SQL-based and document-oriented datastores [4][6]. This experiment is performed to check the impact on processing time, when different percentage of cache-hit queries increases. Thorough tests are conducted for each percentage of exact hit and miss, i.e., from exact cache-hit 100% to 20% in (intervals of 20%).

The cache-size of 50 queries i.e. 20% of the database size was chosen. The studies reported in literature show that the optimum cache-size varies from 10% to 20% of the database-size [16]. BlogInfo database having 250 records is used. Initially, five queries are chosen for warming the cache. One query set is prepared for 100% cache-hit queries, since the cache is warmed by five queries. For remaining percentage of exact hit, three query sets are prepared, so that the experiment is run thrice for each cache-hit percentage and the possibilities of any variation caused by environmental factors are neglected. The total number of query sets prepared is 13 and the total number of queries processed by the cache is 65. Each query set is generated successively. The experiment for each percentage of cache-hit queries is run thrice by executing the program and each query set is generated successively. Table 1 shows PT and CPT between the two successive percentages of cache-hit queries in each run of the experiment.

Table I. Average processing and cumulative processing time (in seconds) in each run of the experiment

Cache-hit %	***100%***	***80%***	***60%***	***40%***	***20%***
Run - I					
PT (sec)	0.0000	0.0068	0.0097	0.0132	0.0184
CPT: Cache (sec)	0.0005	0.0073	0.0101	0.0136	0.0186
CPT: non-cache HBase (sec)	0.0558	0.0500	0.0434	0.0320	0.0224
Run - II					
PT (sec)	0.0000	0.0075	0.0114	0.0117	0.0115
CPT: Cache (sec)	0.0004	0.0080	0.0117	0.0120	0.0117
CPT: non-Cache HBase (sec)	0.0558	0.0520	0.0418	0.0320	0.0228
Run - III					
PT (sec)	0.0000	0.0106	0.0116	0.0174	0.0132
CPT: Cache (sec)	0.0005	0.0110	0.0120	0.0177	0.0134
CPT: non-Cache HBase (sec)	0.0558	0.0474	0.0434	0.0286	0.0226

Table II shows the average PT and CPT and also relative changes in PT and CPT between the two successive percentages of cache-hit queries. It shows that average processing and cumulative processing time increases as we go from 100% to 20% cache-hit. The relative change, measured in seconds, increases from 100% to 80%, 80% to 60% and decreases from 40% to 20% cache-hit percentage. As the cache gets populated with the new queries, the DBRT of the cache-miss queries starts decreasing.

Observations from Experiment 1:

It is observed that the average PT of the queries increases with the decrease in the percentage of exact cache-hit queries. The change in PT from 40% to 20% is comparatively less. This is due to the fact that as the cache-hit percentage decreases, the cache starts getting populating with more cache-miss queries and the time to retrieve the results of the cache-miss queries gradually decreases.

The average CPT on non-cache HBase decreases from 100% to 20%. The relative change in CPT decreases from 100% to 80%, 80% to 60% and the change is more evident from 40% to 20%. This drop in the cumulative time is due to the partial caching in database system and operating system [6].

Table II. Relative Changes in average processing time and average cumulative processing time (in seconds) over non-cache HBase system

Cache-hit %	***100%***	***80%***	***Relative change from 100%-80% (sec.)***	***60%***	***Relative change from 80%-60% (sec.)***	***40%***	***Relative change from 60%-40% (sec.)***	***20%***	***Relative change from 40%-20% (sec.)***
PT(sec.)	0.0000	0.0083	**0.01**	0.0109	**0.31**	0.0141	**0.29**	0.0144	**0.02**
CPT: Cache (sec)	0.0005	0.0087	**16.40**	0.0113	**0.30**	0.0144	**0.27**	0.0146	**0.01**
CPT: non-cache HBase (sec)	0.0558	0.0498	**-0.11**	0.0429	**-0.14**	0.0309	**-0.28**	0.0226	**-0.27**
Improvement using Cache (in %)	**-99.10**	**-82.53**		**-73.66**		**-53.40**		**-35.40**	

It can be concluded from this experiment that the cache-based system definitely takes more time when the queries are approaching first time in cache, but as the cache starts getting populating, the time to retrieve the results of cache-miss queries also gradually decreases. These observations on average processing time are visible in Figure 1. It shows the average processing time increases more rapidly from 100% to 80% cache-hit queries. There is comparatively less increment from 80% to 60% and from 40% to 20% it is almost constant.

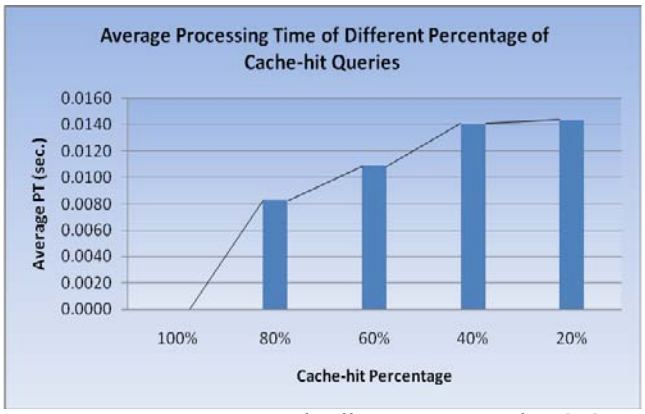

Figure 1. Average Processing Time of Different Percentage of Cache-hit Queries

The observations on the cumulative processing time are shown graphically in Figure 2. The average CPT of non-cache based system decreases with the decrease in the cache-hit percentage and the average CPT of cache based system increases with the decrease in the cache-hit percentage. But the time taken by cache-based system is less as compared with the non-cache based system.

The performance improvements of cache-based system over non-cached HBase in percentages are shown in Figure 3. The improvements in cache-based datastore are 99.1%, 82.53%, 73.66%, 53.40%, and 35.40% at 100%, 80%, 60%, 40% and 20% respectively.

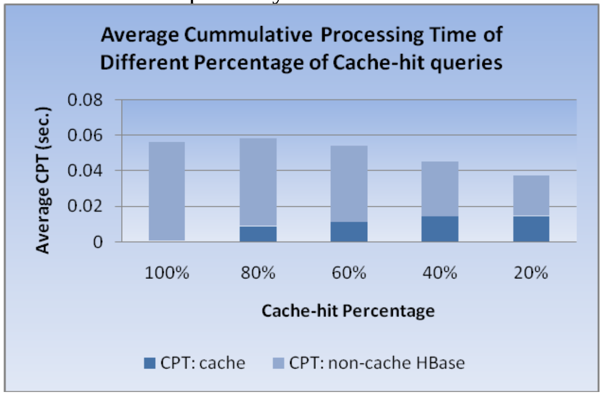

Figure 2. Performance of cache against non-cache column datastores

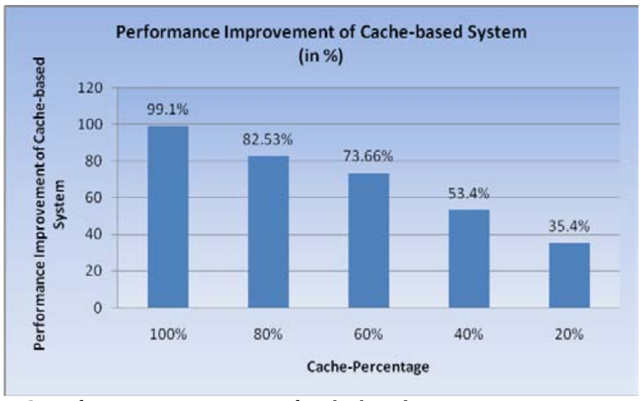

Figure 3. Performance improvement of cache-based system

Experiment No. 2: To determine the effect of similar successive queries on the cache-based system and the non-cache based system

The previous experiment shows that the processing time of the cache-hit queries remains almost negligible. This results in almost negligible cumulative processing time. In case of non-cache HBase, time tends to decrease when a set of queries are executed, which is clearly seen in Table II. This experiment checks whether for the similar queries executed in succession, also time of the cumulative processing of cache-based system decreases. Since the cloud database is shared by many clients, there is always a great possibility of similar queries from different clients. To determine the affect of successive similar queries, the test is performed on a set of ten similar queries. The cache is initially empty. The test is run five times in succession, so that we can easily compare the cumulative processing time at different runs. In each execution of the experiment, the cumulative processing time of the cache system is calculated. Similarly the same sets of queries are executed on non-cache HBase and their time is shown.

Table III shows the change in CPT in seconds, between the two successive runs of the experiment. The average change in CPT (cache-based) in all run is -0.0124 seconds and relative change from Run 1 to Run 5 (in percentage) is -99.40%. The average change in CPT (non-cache HBase) in all run is -0.0242 seconds and relative change in percentage from Run 1 to Run 5 is -62.56%. The negative value shows the decrease in the average CPT. The percentage improvement of cache over non-cache based HBase is 48.64%.

Table III. Percentage improvement in cumulative processing time in successive trips to cache v/s non-cache HBase

Avg. CPT: (sec.)	*Run1*	*Run2*	*Change in CPT from Run 2 to Run 1 (sec.)*	*Run 3*	*Change in CPT from Run 3 to Run 2 (sec.)*	*Run 4*	*Change in CPT from Run 4 to Run 3 (sec.)*	*Run5*	*Change in CPT from Run 5 to Run 4 (sec.)*	*Avg change in CPT in all runs (sec.)*
cache	0.0499	0.0003	-0.0496	0.0003	0.0000	0.0003	0.0000	0.0003	0.0000	-0.0124
non-cache	0.1544	0.0605	-0.0939	0.0535	-0.0070	0.0671	0.0136	0.0578	-0.0093	-0.0242

Relative change in CPT from Run 1 to Run 5 (%):

-99.40% (cache)

-62.56% (non-cache)

% improvement of cache over non-cache HBase: 48.64%

Observations from Experiment 2:

It is observed that there is vast reduction of 99.93% in the overall cumulative processing time from run one to run two. The overall percentage reduction is found to be 99.40%. The result shows that the cache-based system, takes negligible time to process queries in each run. The cumulative processing time is therefore, almost constant after Run 2.

The non-cache based HBase results illustrate speed up of cumulative processing time in second run and during third and fifth run. It is due to the partial caching of results by operating system and internal HBase database [6]. It is not close to the processing time of cache. The overall percentage reduction in non-cache HBase is around 62.56% only.

The conclusion from the experiment is that percentage improvement (reduction in cumulative processing time) of cache in comparison to non-cache HBase is around 48.64%.

The graph represented in Figure 4 clearly depicts the above observations. It shows that the curve showing average CPT in case of cache-based HBase decreases rapidly from Run 1 to Run 2 and remains constant after Run 2. In case of non-cache based HBase, the curve rapidly decreases from Run 1 to Run 2, but not as close to cache curve. It decreases from Run 2 to Run 3 and slightly increases from Run 3 to Run 4. It again decreases from Run 4 to Run 5.

Experiments 1 and 2 show that, the processing time definitely reduces when more cache-hit queries approach the cache. The overall performance of cache-based system is also observed.

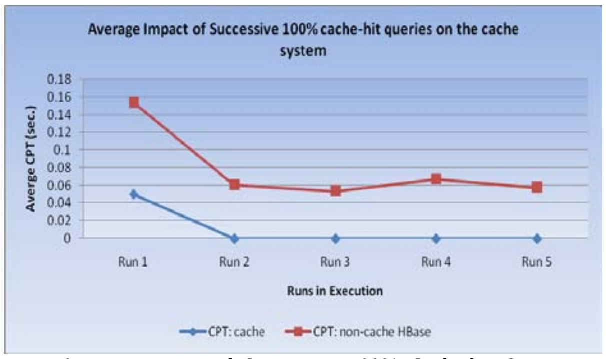

Figure 4. Average Impact of Successive 100% Cache-hit Queries on the database having cache in three Executions

V. Conclusions

The performance enhancement of static cloud datastores queries is paramount. This paper evaluates the performance of static cloud datastore queries. The impact of different percentage of cache-hit queries are experimentally determined and observed that the average PT of the queries increases with the decrease in the percentage of exact cache-hit queries. The change in PT from 40% to 20% is comparatively less. This is due to the fact that as the cache-hit percentage decreases, the cache starts getting populating with more cache-miss queries and the time to retrieve the results of the cache-miss queries gradually decreases. The improvements in cache-based dataastore are 99.1%, 82.53%, 73.66%, 53.40%, and 35.40% at 100%, 80%, 60%, 40% and 20% respectively. It also shows that the successive trips to cache, lead to the performance improvement in cache-based datastore over non-cache based system by 48.64%, in terms of the cumulative processing time.

VI. References

[1] R. Nanda, K. S. Sharma, S. Chande 2017. Determining Appropriate Cache-size for Cost-effective Cloud Database Queries. International Journal of Computer Applications, Volume 157, Issue 6 (2017), 29-34, 0975 – 8887.

[2] 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.

[3] B. Fan, H. Lim, D. G Andersen, and M. Kaminsky, “Small cache, big effect: Provable load balancing for randomly partitioned cluster services,” in *Proc. of the 2nd ACM Symposium on Cloud Computing, 2011*, p. 23.

[4] 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.

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

[6] 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.

[7] 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.

[8] Bu, Y., B. Howe, M. Balazinska, and M. D. Ernst (2012) The HaLoop approach to large-scale iterative data analysis. *The VLDB Journal—The International Journal on Very Large Data Bases*, **21(2)**, 169-190.

[9] 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.

[10] W. Kim, D. S. Reiner, and D. Batory (Eds.) 2012. Query processing in database systems. Springer Science & Business Medi, Springer-Verlag, Berlin, Heidelberg.

[11] P. Karde, and V. Thakare 2010. Selection of materialized views using query optimization in database management: An efficient methodology. International Journal of Management Systems, vol. 2, pp. 116-130.

[12] Feng - Xiao, S. Y. J. M. 2013. A Survey of Query Techniques in Cloud Data Management Systems [J]. Chinese Journal of Computers, vol. 2, pp. 001.

[13] K. Ma, and B. Yang 2015. Introducing extreme data storage middleware of schema-free document stores using MapReduce. International Journal of Ad Hoc and Ubiquitous Computing, vol. 20(4), pp. 274-284.

[14] 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.

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

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