Performance Optimization for Distributed Database Based on Cache Investment

[1]Sanju Gupta and [2]Dr. Swati V. Chande
[1]*The IIS (Deemed to be) University*
Jaipur, India
ksanjana.khandelwal@gmail.com

[2]*International School of Informatics and Management,*
Jaipur, India.
swatichande@rediffmail.com

Abstract*: *In database query processing there are some key factors which determine performance are, the processor speed, the size of RAM, and the cache memory strategy etc. The focus of this paper is on caching method. Retrieving the results from the cache memory is one of the popular techniques to enhancement the performance of the query and in turn improving the query response time. It reduces the load on the back-end database servers, as required data is available on cache. In this paper we describe how to reduce the number of moving tables to database server round-trips by caching most appropriate table on the application server. We propose a novel caching framework, named Proref, which provide the best caching candidates for long run of queries. Proref work on the concept of Cache investment. This paper describes the proposed architecture and workflow of Proref policy which describe how this policy is designed and calculate the best cache candidate for caching in the distributed environment. The results show how this proposed policy helps in improving the performance of the database, especially relevant for today's "big data" environment.

***Keywords*: Distributed database, Caching, Cache investment, Query processing, Query Performance, Response Time.**

(Article history: Received: 3rd April 2022 and accepted 15th December 2022)

I. INTRODUCTION

As we know that most important part of computer is the central processing unit (CPU). Due to advancement in chip technology, size of CPU chip is getting smaller with improved performance. One aspect which leads to slower processing speed is still the focus point for researcher's i.e., the communication time between system's main memory (random access memory) and CPU chip. This issue is not resolved simply by increasing memory size as major cost (interaction time) is taken by CPU to access the memory. To address this issue, system developers have find a solution to use cached memory, In this case, chip is also having some memory. Although size of cache memory is very small in comparison to main memory, but it can be accessed much faster. To retrieve information in cache memory quickly, cache in CPU keep the information stored somewhere. Due to the implementation of the cache memory, system performance enhanced significantly.

As mentioned in the above paragraph advance computers with improved performance need big memory and fast processor. In the system, where data need to be processed and lot of data/information is transferred between processor and memory and vice versa. In this environment cache become very important and required features of the system due to its effectiveness and small size. We can define cache performance by two ratios: i) hit ratio and ii) miss ratio. Performance enhancement of the queries can be achieved either by hardware-controlled change or software-based approach. This paper explains the proposed policy for the performance improvement of distributed database system. This proposed strategy optimizes the cache performance based on improving the cache hit ratio.

The paper is arranged as below-

- In section II prior related work for caching and cache investment is discussed.
- Section III explain the experimental setup. It includes description and implementation of Proref policy algorithm and formula evolution.
- Section IV contains the results obtained from the mathematical experimental setup and practical implementation.
- the last section contains the conclusions.

II. PRIOR RELATED WORK ON CACHING AND CACHE INVESTMENT

Caching can be applied to many types of databases like a) relational databases for example Amazon relational database system (Amazon RDS) or b) NoSQL databases for example Apache Cassandra, MongoDB and Amazon DynamoDB. Caching does not interfere in implementation and by doing so it helps to achieve scale and improvement in speed of application performance. Most effective strategy for improving application performance is in memory data caching.

system performance can be derived from individual unit performance like i) I/O units, ii) caches, iii) bus, iv) integer, branch and floating point, v) memory systems. Speed of processor and main memory has huge gap and it's increasing exponentially. For the period of 2001-05, CPU speed increased by 55% and memory speed by 7%. This data is published by research group clock frequency. This phenomenon is considered as memory wall. The main purpose of cache to reduce the speed gap and overcome the memory wall (Tarig Ibrahim Osman Ahmed, Elsanosy M. Elamin, 2018).

As cache is proved to effective tool to overcome the memory wall, matrix for its performance is playing big role in selecting parameters like size of cache, replacement policy and associativity. performance also depends on cache hits and cache misses' number ('Sheshappa S.N., Ramakrishnan K.V. and G. Appa Rao, 2012'). Cache hits means the probability of finding the required data in cache and cache miss means not finding the data in cache. AAT (average access time) or AMAT is the average time taken to access the memory is calculated based on count of cache hits and misses. This parameter is very significant for measurement of cache performance. This number depends on speed of processor so as processor speed increases, performance improves.

Commercial database systems have been using caching to speed up processing for a long time. To decrease disc IO in a distributed setting, all systems typically use one or more-page caches (also known as buffer pools) that store frequently used database pages. The size of buffer pool is in gigabytes in a large database server. Many systems cache execution plans in order to avoid repeatedly optimizing the same query or stored method. In many systems, catalogue data is also regularly cached. Different forms of cache data are employed in different systems during query optimization and query processing, although the specifics vary from system to system. All of these are examples of software caches that are used in conjunction with hardware caches in current CPUs to decrease main memory and hard drive accesses (Themis Palpanas, Per-Åke Larson and Jonathan Goldstein,2002).

Data caching at client workstations has shown to be a successful method for enhancing the performance of a client-server database system (Carey, Michael, Miron, and Eugene ,1991); Michael J. Franklin and Michael J. Carey, 1994; Michael J. Franklin, 1996). Client-server interactions can be decreased by caching data for subsequent reuse, improving response time and reducing network traffic. Because queries can be processed using data cached on the client side, the client CPU, memory, and disc resources are utilized, and the server workload is greatly reduced. As more clients are added to the system, more resources can be used, therefore system scalability is improved. The major research issues in data caching include: i) caching granularity, ii) cache coherency strategy, iii) cache replacement policy and iv) prefetching scheme, etc. (Qun Ren and Margaret H. Dunham, 1998).

(Konard et.al., 2009) adopted a cache investment caching strategy for use in a peer-to-peer database with semantic cache Cache candidates are chosen using this method based on previous query executions. The semantic cache, according to the author, is used to cache composite data such as interim results from prior queries. Result of this paper shows that with proper utilization of semantic caching in peer-to-peer environment, it is possible to decrease data flow and avoid unneeded, costly procedures. When leveraging semantic cache investment, performance has been found to improve.

III. EXPERIMENT SETUP AND OBJECTIVE

In order to boost the efficiency of the distributed database system, the proposed policy (Proref policy) has been developed. It improves the performance of a system by reducing the response time for query execution. This proposed policy assigns Proref value (explained later in this paper) to each database table and helps to identify better cache candidate. Experiments were conducted to investigate and explain the working of this proposed policy.

Objective- The experiment has been conducted with following objectives:

1) To find best candidate for caching using Proref policy in the distributed environment.

2) Design and prove the Hypotheses

The Proref policy is used to boost the performance of queries in a distributed database system. Simple hypothesis method is used to prove it. This hypothesis predicts the relationship between the single dependent variable (response time) and single independent variable (cache hit ratio).

Hypothesis: ***"Proposed Proref policy improves the performance of the distributed database queries by minimizing query response time and increasing the cache hit ratio."***

For mathematical setup to see the improvement of Proref policy, following values have been considered to make the experiment simplified.

a. *Database type: Distributed database*
b. *Size of the cache – 1 MB*
c. *Size of the table – 100 KB (Uniform size for all tables)*
d. *No. of sample database Tables- 50*
e. *Min runs of Query -500*

As we will increase the complexity and history of the work, this policy will give better results. For further work, we can change the values and increase complexity to observe the performance improvement.

A. Proposed Policy method and formula:

The proposed policy work for better cache candidate. To find the same, following methods and formula derived using existing policies (reference counting and profitable) architecture.

As per our Policy To identify the cache candidate following formula is used for table (q) at client (i)

$$Kit(q) = Vit(q) + Rit(q) \quad \text{-----} \quad (I)$$

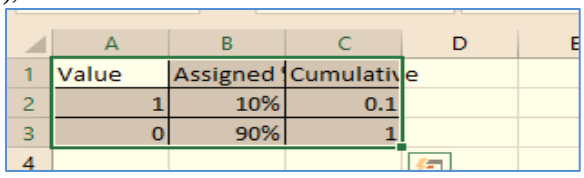
Where

Kit(q) = Proref Policy Value for table (q) following the execution of query (t)

Vit(q) = value of table (q) assigned after the execution of query (t). This value shows how frequent table has been used.
Rit(q)=Improvement in Response time after the execution of query (t)
For Value of table Vit(q) calculation following formula has been used
Vit(q) =Cit(q) + α* Vit-1(q) ----- (II)
Where:
Cit(q) =Component of value for reference count
Cit(q) =0 if table is not used in query
Cit(q) = 1 if table is used in query
α = Aging Factor (0≤ α ≤ 1). We have used α =1 to give equal weightage
To calculate Response time Rit(q) following formula has been used
Rit(q) = Oit(q) - Lit(q) -----
(III)
Where:
Oit(q) = Response time to execute query with present cache condition using optimal plan for table (q)
Lit(q)= Response time to execute query with caching the table (q)

B. Calculation for value of table Vit(q) (Step 1)

Value of the table (Vit(q)) assigned based on the Component value Cit(q).
Where:
Cit(q) =Component of value for reference count
Cit(q) =0 if table is not used in query
Cit(q) = 1 if table is used in query
In the experiment, tables have been randomly selected in query with 10% probability condition. That means out of the 50 tables, only 5 tables have been used in each query.
Formula used for the same in excel is as below:
=INDEX(A$2:A$3,COUNTIF(C$2:C$3,"<="&RAND())+1),Where



	A	B	C	D	E
1	Value	Assigned	Cumulative		
2	1	10%	0.1		
3	0	90%	1		
4					

In the following table, results for 8 tables and 10 queries have been shown.
Table 1: Table to get Cit(q):

Query Number	use of Table 1	use of Table 2	use of Table 3	use of Table 4	use of Table 5	use of Table 6	use of Table 7	use of Table 8
1	1	0	0	0	1	0	0	0
2	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	1	0
8	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

Till 500								

For Value of table Vit(q) calculation following formula used
Vit(q) =Cit(q) + α* Vit-1(q)
Where α = 1 so Vit(q) =Cit(q) + Vit-1(q)
For initial part of the sheet value of the table remains almost in a range for all tables but going forward it reflects the frequency of the table used in queries.

Table 2: Table with calculation for Vit(q) for initial 10 queries are as below:

Query Number	Value of Table 1	Value of Table 2	Value of Table 3	Value of Table 4	Value of Table 5	Value of Table 6	Value of Table 7	Value of Table 8
1	1	0	0	0	1	0	0	0
2	2	0	0	0	1	0	0	0
3	2	0	0	0	1	0	0	0
4	3	0	0	0	1	0	0	0
5	3	0	0	1	1	0	0	0
6	3	0	0	1	1	0	0	0
7	3	0	0	2	1	0	1	0
8	3	0	0	2	1	0	1	0
9	3	0	0	2	1	0	1	0
10	3	0	0	2	1	0	1	0

Where value of the table changes with each use of the table in query. In the simulation 50 tables and 500 instances of query execution have been considered.

Table 3: Sample sheet with calculation for Vit(q) at the end of the experiment range is as below:

Query Number	Value of Table 43	Value of Table 44	Value of Table 45	Value of Table 46	Value of Table 47	Value of Table 48	Value of Table 49	Value of Table 50
491	50	48	39	48	43	52	50	58
492	50	48	39	48	43	53	50	58
493	50	48	40	48	44	53	50	58
494	51	48	41	48	44	53	50	58
495	51	48	41	48	44	53	50	58
496	51	48	41	48	44	53	51	58
497	51	48	41	48	44	53	51	58
498	51	48	41	48	44	53	51	58
499	51	48	41	48	44	53	51	58
500	51	48	41	48	44	53	51	58

C. Calculation for improvement in response time Rit(q) (Step 2)

To calculate improvement in response time Rit(q) following formula used

Rit(q) = Oit(q) - Lit(q) ----- (III)

Where:

Oit(q) = Response time to execute query with present cache condition using optimal plan for table (q)

Lit(q)= Response time to execute query with caching the table (q)

For mathematical environment, random variable number has been arrived at using excel formula "**Randbetween**" considering normal range of response time for reference counting and profitable policy. For profitable policy, it is derived basis on the formula based on continuous improvement with number of queries run.

In this experiment, response time for various policies has been documented. In table 4 response time at various queries at different query intervals captured for reference.

Table 4: Sheet with response time in reference counting, profitable and Proref policy is as below:

Refer below figures for response time graph for experiment. Summary of the same as below:

Query Number	Response time (in ms)		
	Reference counting	Profitable	Proref Policy
1	783	765	792
2	781	767	797
3	800	766	801
4	796	761	808
5	790	767	800
6	814	790	798
7	800	776	791
8	781	782	808
9	791	792	800
10	793	783	792
351	819	775	720
352	782	787	739
353	796	798	728
354	786	792	722
355	819	791	734
356	787	776	733
357	810	786	734
358	796	799	735
359	812	783	740
360	802	775	728
480	810	766	705
481	782	760	703
482	791	774	706
483	811	797	712
484	797	775	710
485	812	767	718
486	793	784	710
487	814	762	715
488	813	761	714
489	816	796	719
490	805	800	713

Figure 1 - For reference counting in ms

Figure 2 - For profitable policy in ms

Figure 3 - For Proref policy in ms

Figure 4 - comparison for all three policies in ms

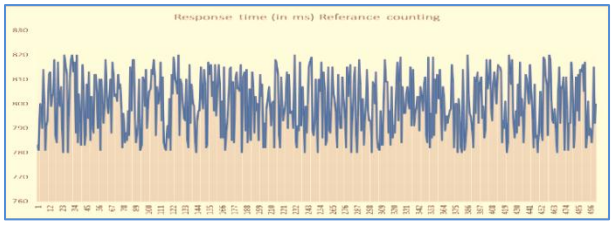


Figure 1: Response time for reference counting in ms

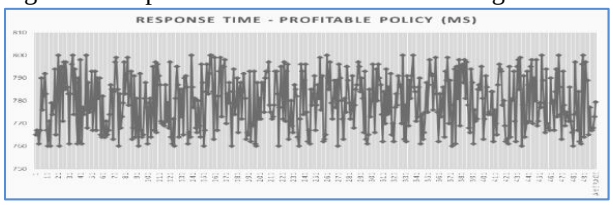


Figure 2: Response time for Profitable Policy in ms

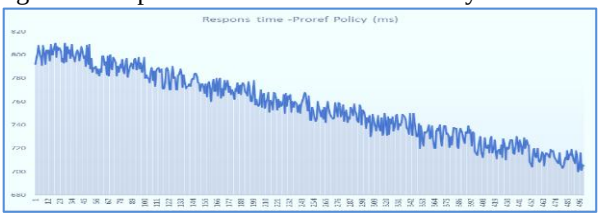


Figure 3: Response time for Proref Policy in ms

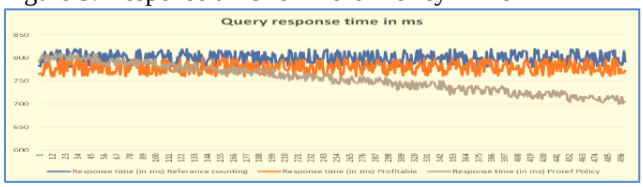


Figure 4: Response time comparison for all 3 policies in ms

D. Calculation for Proref Policy value for table Kit(q) (Step 3)

To identify the cache candidate following formula is used for table (q) at client (i)

Kit(q)= Vit(q) + Rit(q) ----- (I)

Where

Kit(q) = Proref Policy Value for table (q) after the execution of query (t)

Vit(q) = value of table (q) assigned after the execution of query (t)

Rit(q)=Improvement in Response time after the execution of query (t)

Table 5 Sample sheet with calculation for Kit(q) is as below:

Query Number	Value of Table 1	Value of Table 2	Value of Table 3	Value of Table 4	Value of Table 5	Value of Table 6	Value of Table 7	Value of Table 8
1	-8	-9	-9	-9	-8	-9	-9	-9
2	-14	-16	-16	-16	-15	-16	-16	-16
3	1	-1	-1	-1	0	-1	-1	-1
4	-9	-12	-12	-12	-11	-12	-12	-12
5	-7	-10	-10	-9	-9	-10	-10	-10
6	19	16	16	17	17	16	16	16
7	12	9	9	11	10	9	10	9
8	-24	-27	-27	-25	-26	-27	-26	-27
9	-6	-9	-9	-7	-8	-9	-8	-9
10	4	1	1	3	2	1	2	1

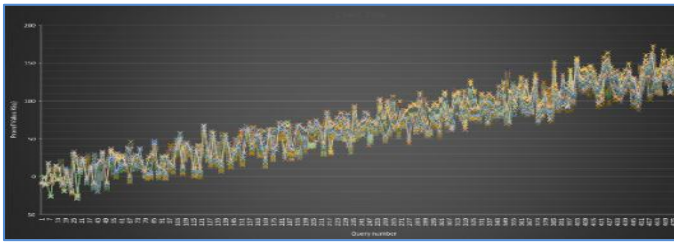


Figure 5: Proref Policy value for queries executed in experiment

IV. Result for Mathematical Evolution

After execution of the experiment, it is observed that Proref Policy improves the response time gradually as the number of queries increases. Refer Table 6 for average response time after running of 500 queries. Average response time improves in case of Proref policy over comparison to reference counting and Profitable policy. Fig. 6 is show trend of how response time improves gradually in case of Proref policy. Table 7 and Fig. 7 show average response time improvement in Proref policy.

Table 6: Average response time for all policies after executing 500 queries is as below:

	Policy	Avg. Response Time
1	Reference Counting Policy	799.7 ms
2	Profitable Policy	779.5 ms
3	Proref Policy	755.5 ms

Response time Improvement due to Proref policy over reference counting and profitable policy is as below:

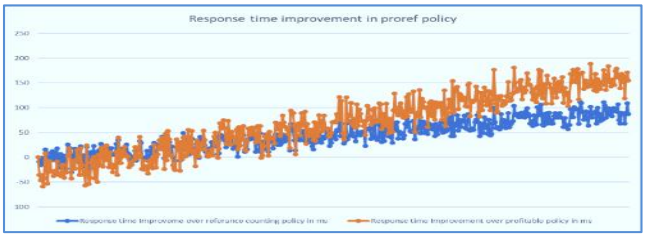


Figure 6 Response time Improvement due to Proref policy over reference counting and profitable policy

Table 7 Average response time improvement

Description	Reference counting	Profitable
Average Response time improvement in Proref policy over other history-based policy (in ms)	44.15	23.932

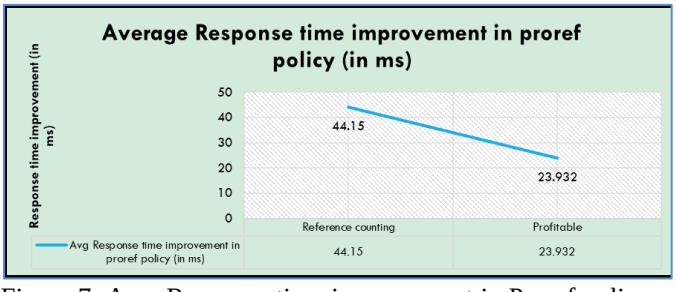


Figure 7: Avg. Response time improvement in Proref policy (in ms)

A. *Comparison of proposed policy with existing History-based policies*

In addition to mathematical experiment, proposed policy compared with existing work done by Donald Kossmann, Michael J Franklin and Gerhard Drasch (Kossmann et.al, 2000). In this comparison same environment has been considered in the calculation. In process for calculation of response time, parameters considered in existing work have been considered here and value for each policy derived based on the same.

Parameter	Value	Description
NumClients	1 or 10	number of clients
NumServer	1 or 10	number of servers
Mips	50	CPU speed of a site (10^6 inst/sec)
NumDisks	1	number of disks per site
ClMemory	2 - 40	client's main memory (% of database)
ServMemory	2 or 40	server's main memory (% of database)
NetBw	100	network bandwidth (Mbit/sec)
PageSize	4096	size of one data page (in bytes)
Compare	2	instructions to apply a predicate
HashInst	9	instructions to hash a tuple
MoveInst	1	instructions to copy 4 bytes
MsgInst	20000	instructions to send or receive a message
PerSizeMI	12000	instructions to send or receive 4096 bytes
DiskInst	5000	instructions to read a page from disk

This comparison validates the response time improvement due to proposed policy. Results of the experiment are as below:

Table 8: Comparison of proposed policy with existing history-based policies

	Reference counting	Profitable	Proref Policy
Average Response Time (in ms)	907.5	823.2	742.7
Response time improvement in Proref policy (in ms)	164.88	80.5	0 (Base data for comparison)

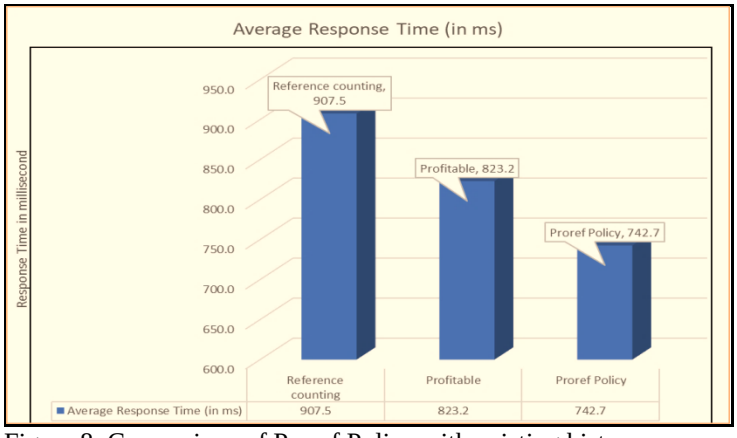


Figure 8: Comparison of Proref Policy with existing history-based policies (in ms)

B. Results from practical implementation:

In this experiment Load check parameter for dashboard and charts measured before implementation of Proref policy and after implementation of Proref Policy. In both parameters query time (benchmark time) and page load time has been measured.

Load check is used for testing of software to measure service quality of website based on expected behavior of user by modelling the usage and simulating the software program by number of users simultaneously.

Result of the Experiment are as below:

Pre- Implementation Report of Proref Policy

1. Dashboard Load Check
   a. Benchmark time (query time) - 0.073489 seconds
   b. Page load time - 2.465 seconds
2. Charts Load Check
   a. Benchmark time (query time) - 0.142767 seconds
   b. Page load time - 1.343 seconds

After Implementation of Proref Policy:

1. Dashboard Load Check
   a. Benchmark time (query time) - 0.047980 seconds
   b. Page load time - 1.585 seconds
2. Chart Load Check
   a. Benchmark time (query time) - 0.073033 seconds
   b. Page load time - 837 milliseconds

It is evident that after Implementation of Proref policy, performance of the system increased.

V **Conclusion:**

This paper presents a history-based hybrid Proref policy. Which works on the concept of cache investment to increase the performance gains in distributed environment due to caching the most appropriate cache candidate. Experiment results show that implementation of Proref policy improves the response time of queries in distributed environment. The policy has been simulated mathematically and results are explained in section IV which are in line with our hypothesis.

"REFERENCES

[1] Donald Kossmann , Michael J. Franklin and Bjorn Thor Jonsson, "Performance Tradeoffs for Client-Server Query Processing" , ACM – SIGMOD Conference on Management of Data , New York,1996.

[2] Sheshappa S.N., Ramakrishnan K.V. and G. Appa Rao," Enhancing Cache Performance Based on Improved Average Access Time", International Journal of Scientific and Research Publications, Volume 2, Issue 11, November 2012 1 ISSN 2250-3153.

[3] Ugah John Otozi , Abakaliki Chigozie-Okwum Chioma , Owerri Ezeanyeji Peter C." Virtual and Cache Memory: Implications for Enhanced Performance of the Computer System", International Journal of Computer Applications (0975 – 8887) Volume 181 – No. 20, October 2018.

[4] M. Kowarschik and C. Weiß, "An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms," Lecture Notes in Computer Science Vol. 2625, pp. 213-232, Springer, 2003.

[5] 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

[6] Kaladhar Voruganti ,M. Tamer Ozsu , Canada Ronald C. Unrau," An Adaptive Hybrid Server Architecture for Client Caching Object DBMSs", Proceedings of the 25th VLDB Conference Edinburgh, Scotland, 1999.

[7] Donald Kossmann, "The State of the Art in Distributed Query Processing", ACM Computational Surveys, vol. 32, Dec. 2000.

[8] White Paper,"Server side cache invalidation through Oracle push notification," External Document © 2015 Infosys Limited.

[9] Sanju Gupta, Swati V.Chande," A Hybrid Cache Investment Strategy for Distributed Database Queries", International Journal of Computer Applications (0975 – 8887) Volume 145 – No.5, July 2016

[10] Abhijit Gadkari, "Caching in Distribute Environment", The Architecture Journal,2009.

[11] Shaina,Anshu Kamboj, " High Performance E-Business using Application Level Caching", International Journal of Advanced Research in Communication Engineering, vol3,issue sep.2014.

[12] Mantu Kumar,Neera Batra and Hemant Aggarwalo, "Cache Based Query Optimization Approach in Distributed Database",IJCSI,Vol.9, Nov.2012.

[13] Konard G.Beiske,Jan Bjorndalen,Jon Olav Hauglid, "Semantic Cache Investment" , NIK-2009 conference.

[14] Ruchi Nanda, Swati V. Chande, Krishna S. Sharma, "Determining Appropriate Cache-size for Cost-effective Cloud Database Queries", International Journal of Computer Applications (0975 – 8887) Volume 157 – No 6, January 2017.

[15] Norvald H. Ryeng, Jon Olav Hauglid, and Kjetil Norvag , "Site-Autonomous Distribted Semantic Cachig", SAC,2011 .

[16] Donald Kossmann , Michael J. Franklin, "Cache Investment Strategies", Univ.of MD Technical CS-TR3803 and UMIACS-TR - 97-50,May 1997.

[17] Ideh Azari," Efficient Execution of Query in Distributed Database Systems", 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE).

[18] Donald Kossmann , Michael J. Franklin,Gehard Drasch, "Cache Investment : Integrating Query Optimization and Distributed Data Placement," ACM Transaction on Database System (TODS), Dec. 2000.

[19] ,"Local Disk Caching for Client-Server Database Systems *", Computer Science Department University of WisconsinMadison,1994.

[20] Doshi P. and Raisinghani V., "Review of Dynamic Optimization Strategies in Distributed Database", Electronics Computer Technology (ICECT), 3rd International Conference, April 2011.

[21] Yan T,IacobesnM,Garcia-Mo Lina H,"Introduction of Query optimization of distributed database", WAM Press, I 999.

[22] Alaa Aljanaby, Emad Abuelrub, and Mohammed Odeh,"A Survey of Distributed Query Optimization", The International Arab Journal of Information Technology, Vol. 2, January 2005.

[23] Elmasri R. and Navathe S. B.," Fundamentals of Database Systems, Reading", MA, Addison-Wesley, 2000.

[24] Donald Kossmann , Michael J.Franklin,Gehard Drach,"Cache Investment for Indexes",VLDB Conference,Feb,1998.

[25] Hua-Ming Liao, Guo-Shun Pei, "Cache-Based Aggregate Query Shipping: An Efficient Scheme of Distributed OLAP Query Processing", JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 23(6): Nov. 2008.

[26] Ruby Bhati ,Nitika Bansal, S K Jha," Distributed Database System:The Current Features And Problems?", International Journal of Computer Science and Management Research, Vol 2 , March 2013

[27] Laura M. Haas, Donald Kossmann, Ioana Ursu ," Loading a Cache with Query Results.", Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[28] Tarig Ibrahim Osman Ahmed1 , Elsanosy M. Elamin2 ," Design Strategy of Cache Memory for Computer Performance Improvement", International Journal of Research Studies in Electrical and Electronics Engineering(IJRSEEE) Volume 4, Issue 3, 2018, SSN 2454-9436."

AUTHOR PROFILE

Mrs. Sanju Gupta
Specialization: DBMS, Big Data, ERP, Cache investmentPh.D. scholar from The IIS University, Jaipur.

Dr. Swati V Chande Professor and HOD, Computer Science
Specialization: Big Data, DBMS, Soft Computing,
Qualification: Ph.D, M.S. (Software Systems), M.Sc. B.Sc (Hons.)