Journal of Engineering Science and Technology
Vol. 12, No. 9 (2017) 2476 - 2487
© School of Engineering, Taylor's University

AN INTELLIGENT APPROACH OF QUERY PROCESS OPTIMISATION USING COOPERATIVE SEMANTIC CACHING TECHNIQUE

P. MOHANKUMAR*, BALAMURUGAN BALUSAMY

School of Information Technology and Engineering, VIT University, Vellore, Tamilnadu India
*Corresponding Author: pmohankumar@vit.ac.in

Abstract

Query processing is the key aspect in any of client-server application. Optimality is measured based on various concerns such as architecture base, search engines, query processor, query formalism, query representation languages, IO and storage cost. Researchers strived and provided various solutions for query process optimization in the past. Efficient query processing requires an advanced caching mechanism to reduce the query response time. Semantic caching produce results for optimizing the evaluation of database queries by caching results of old queries and using them when answering new queries. We use a collaboration of semantic caches in a cooperative way process to resolve user queries so-called cooperative semantic cache as an extension. As a novelty in this paper, a new way of cooperative semantic cache organisation is proposed, i.e., a paradigm for reference caching as prefix _index of query information to reduce access latency as well as fault tolerance. Intelligence is used for fetching the relevant cache content and its consistency. Optimality is measured with an empirical test on real time application.

Keywords: Cooperative, Semantic cache, Intelligence, Fault tolerance.

1. Introduction

Query caching allows reuse of answers to previous queries, so reducing the delivery time of answers and the traffic on the net. Semantic caching is based on the representation of cached data as semantic regions and the processing of queries by the construction of probe queries for retrieving cached data and remainder queries for fetching data from remote servers. As an extension by enabling where clients are allowed to share their cached query results in a cooperative manner so called cooperative semantic cache. In general, this allows clients to register queries for which they have cached results and also to search for queries stored in the collaborative cache that subsumes or overlaps new queries for which they want the result. This approach attracts recent researchers due to the benefits such as reduction of access latency, an increase in throughput and hit ratios and quick response time. Research solutions provided for environment base architectural issues and utilisation such as mobile, cloud, the web and distributed as well as peer to peer. Only a few concentrated about cache relevance and query processing issues.

Amidst as novelty, this work focuses on cache content consistency, intelligence among cooperative cache sharing, fault tolerance, and propose as a new approach, reference caching (prefix-index) of query information to extract the relevance to achieving optimality. The roadmap of our workflow follows initially as motivating example, architectural issues, intelligence and fault tolerance approach, cache content consistency, i.e., which content should be in cache during query execution. It allows the comparison of reference point caching in the environment to reference caching prefix index of query information details and empirical test to check optimality.

2. Motivation and Related Works

In Semantic caching [1] applications Clients store the results of old queries, together with their descriptions, where old query results are used when answering new queries. Semantic caching [1-4] is a technique that integrates support for associative access into an architecture based on data-shipping. Sometimes, a subsequent query can be executed only using the cached data. We deploy the academic database scenario followed in universities to gather related information in so called fully flexible credit systems (FFCS). This has been detailed in later session as an empirical test in this paper. Let us consider the following scenario for example Client sends a request (query) to the server (s1: select faculty name from the course where course = ‘DBMS’) to retrieve the entire faculty which handles DBMS during the morning slot. The server returns the result set, and the client stores it in the local cache. After sometime another client requests server (s2: select faculty name from the course where course = ‘DBMS’ and slot = ‘morning’). It observed that the answer for s2 is totally subsumed by s1. Thus, s2 can be answered locally using the cached result set of s1. In general new queries are, not always totally contained in the cached queries. When there is just an overlapping between the new and the cached queries, the query is split into two disjoints parts: one that can be answered using the data contained in the cache (which is called the probe query) and a remainder query that must be execute in the server. This split is done automatically, by analysing the semantics of the queries [5].

In our proposed work we highlight intelligence and reference caching index for accessing the query relevance towards which we reach cache consistency and maximum throughput. Intelligence here we inference to identify the client cache in the network either P2P or distributed which stores the query relevance. Intelligence for semantic cache approach is discussed in detail [6]. Apart from this what to be cached, how long it should be retained, either in static or dynamic, during execution alone or how often it can be replaced is considered for cache consistency issue. With respect to a query a relevant semantic needs to be defined and identify how the defined semantic can be viewed as semantic regions. How the semantic regions can be viewed as granularity and coalescence has to be mapped and syntax will be checked for the incoming query to be processed and this process is detailed in [7]. We define the prefix reference index based on semantic of the query and its result. For each and every incoming valid query we keep the reference id integrated with corresponding resultant ID which is defined, and it gets stored in the cache. This may be defined manually or automated by coding so that the query id from system defined may be used to refer its corresponding result which can be indexed in the cache.

3. Intelligence and Fault Tolerance Analysis

Intelligence falls on how the information relevant to the incoming query is identified, i.e., availability in client cache location. Not only for identification as stated in [6] but also the semantic knowledge used to transform a cache miss in a conventional caching to a cache hit. Here we also use intelligence for providing cooperative capabilities conceptually and approximate query answering. In general, our approach is closely about query satisfiability and query containment problem. Query mapping and processing will be discussed in detail in the sections below. Here we view the intelligence on behalf of identifying the relevance where the information exists. Intelligence inspire the mechanism deployed in [7] for query processing, but prior the architectural view is to consider as the cause for involving intelligence, i.e., to find where actually the relevance exists and how it can be collaborated cooperatively.

3.1. Intelligence inference interface as wrapping agent

Detailed architecture view is given in [2, 5]. The generalised architecture for our work is given in as shown in Fig. 1. We further follow the inspiration of the following Fig. 2. For our work as data analysis, cache indexing during adaptation and acquisition finally as intelligence to tune the user output relevant to their level of satisfaction as wrapping agent.

Intelligence is used for fetching the relevant cache content and its consistency in our work. The overview of intelligence in a generalized view is shown in Fig. 2. In most cases intelligence is used for decision making. Here the hierarchy flow is as data to information then information to knowledge and knowledge to intelligence and finally as intelligence decision. The data in the context becomes information, information plus insight becomes knowledge, knowledge with the intuition of foresight becomes intelligence, intelligence in right situation drives better decision based that the process gets executed.

Once the intelligence decision is over the cooperative semantic caching query response workflow starts which is shown in Fig. 3. Here the universal index plays the core role. Actually, it is the virtual process of intelligence (wrapper agent layer) as shown in Fig. 1. In our case as discussed the index is used to identify the query relevance cache as well as the resultant in the log. Here the universal index acts as a look-up table to find out the relevance during execution. In Fig 3.the work flow shows the entire process of cooperative semantic caching mechanism with proposed approach. The details are given in the below algorithm session [7].When a user posts the query, the query received by the intelligence agent, it parses the query and find the details with respect to attribute, value and relation and type of operation either read or write. If its read operation, any node will be assigned to take responsible for execution and acknowledged to the concern. If its write operation, the exact cache where relevant data stored is identified and begins the process of execution.

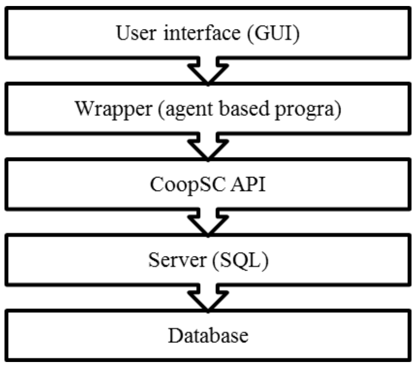

Fig. 1. Generalized architecture.

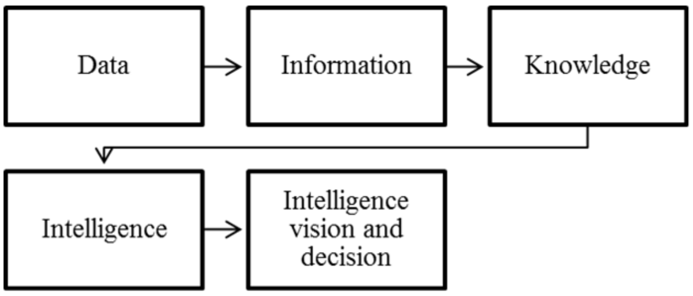

Fig. 2. Intelligence inference work flow.

In case if the particular request is a failure, neighbouring node is assigned and allowed to process and execute. After failure recovery it is acknowledged and updated as the modification. The cache can be viewed as resultant cache, query log and data cache, a universal index which holds the information about the query-relevant cache details. The caching architecture prior process follows the pros and cons discussed in [8, 9] and while processing in [10-12]. The execution is detailed in the below-proposed method.

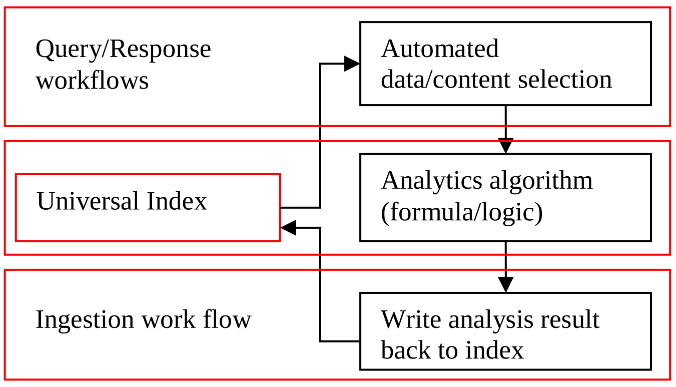
Query/Response workflows
Automated data/content selection
Universal Index
Analytics algorithm (formula/logic)
Ingestion work flow
Write analysis result back to index

Fig. 3. Cooperative semantic caching query response workflow.

3.2. Fault tolerance analysis

In general most of the client-server architecture, the client is a mediator or an interface and for some applications the client machines will be the processing element; It will process the incoming request appropriately relevant to the process before giving it to the server for execution. This is mostly preferred in parallel, distributed, peer to peer, multiuser and agent base application environments. In our case since we wish to provide fault tolerance as well as intelligence, we prefer the client to be the processing element. Few autonomous processing elements (CPU+ storage cache) are allowed to form a client network. They may be connected or remote, centralized or distributed form satisfying peer to peer environment. Their access must be based on the IP address apart from content access. Once the client environment is confirmed the basic issue is how the collaboration is achieved cooperatively to get the query relevance. As we infer the cache reference point mechanism as well as fault tolerance, either the peer to peer environment must be used as a group-server approach or any one of the processing elements amidst the client machine must be the organizer to process the necessary (centralized approach). In our case, we deploy group server mechanism, i.e., any free client machine can respond to the incoming query request and pass over to the relevant in the group. In certain cases, it can even process the request and update later the result to the relevant machine. If its read-only operation, irrespective of the relevant member in groups it can process and respond to the user. Every operation is acknowledged to the host element which holds the actual data irrespective of read or write. This satisfies fault tolerance, i.e., irrespective of failures and misbehaviour of few processing elements the actual job will not get effected .In our experimental set up how this, is detailed. We keep all the original schemas in individual machines and corresponding copy in all other elements in the group.

4. Proposed Method

Based on the above analysis and discussion we propose a method to process query through cooperative semantic caching technique and analyse the optimization measure using intelligence approach. Here intelligence is viewed as wrapping agent (a set of code) strives as preliminary to bring out optimization. We divide our approach into two sessional procedures: one as an intelligence agent and other cooperative semantic caching processor. Intelligence agent plays a parallel role one to communicate with user and server and also as a mediator between server and the client. Cooperative semantic caching based on the result given by intelligence agent gets operational. All the details from user query entry to exit are explained in various sessions in this paper appropriately. More over the obtained optimistic output is shown in Fig. 5.based on the empirical test.

Cooperative semantic caching approach

Cooperative semantic caching server approach was deployed in order to reduce access latency, improve processing time and performance. Here the cooperative semantic cache is allowed to spread over P2P distributed network. The universal index was deployed and used to look up a reference for the cache content identification during execution which supports the additional feature to the system. Cooperative semantic caching mechanism comes under query processing which was core concept. The intelligence plays as a covering agent for the processor to the user interface. It also assesses the user input and associate cooperative caching server for query processing by providing the cache location where the relevance exists by making use of the universal index. This was discussed in Fig. 3.

The cooperative semantic caching mechanism was discussed in below algorithm. The designed setup and utilization was detailed and discussed in section empirical test. Here the core concept will be detailed. In cooperative semantic caching mechanism the clients are allowed to share their cached query results in a cooperative manner as well as to register queries for which they have cached results and also to search for queries stored in the collaborative cache that subsumes or overlaps new queries for which they want the result. This was based on the principle of query caching which allows reuse of answers to previous queries, reducing the delivery time of answers and the traffic on the net and semantic caching representing cached data as semantic regions and processing queries by construction of probe queries for retrieving cached data and remainder queries for fetching data from remote servers. Thus by enabling semantic caching mechanism where clients to share their local semantic caches in a cooperative manner the concept was designed. When executing a query, the content of both the local semantic cache and entries stored in caches of other clients can be used.

A new query will be split into the probe, remote probe, and remainder sub-queries using a query rewriting process. The probe retrieves the part of the answer, which is available in the local cache. Remote probes retrieve those parts of the query which are available in caches of other clients. The remainder retrieves the missing tuples from the server. Here the critical task is query rewriting. If the query is invalid or if it is already executed the system just uses the index reference, the cache and gets concluded but the when the query is partial, or the query is similar to be written in other relevance or equivalence form then the actual work starts. In this paper, the work is limited for rewriting by replacing the alternative queries as well as for select and project operations alone. The setup was made appropriately and tested result was discussed.

In general, in order to execute the query rewriting, cache entries of all clients will be indexed in a distributed data structure built on top of a Peer-to-peer (P2P) overlay that is formed by all clients which are interrogating a particular database server. The query rewriting process determines parts of a given query that can be answered using the local cache (probe), caches of other clients (remote probe) or database server (remainder) and the way in which they are combined in order to return the final query result. This process is executed by a component, running on the client side, called query rewriter. The result of query rewriting process is a query plan tree, which describes how the query is to be executed. Initially, the query rewriting checks entries stored in local cache (Local Rewriting). Afterward, the distributed index is interrogated in order to determine remote cache entries which can be used for answering given query (Distributed Rewriting) which was a broad theory to be implemented for large scale applications.

Procedure intelligence agent ( )

```
Input: user query {keyword/sentence/query}
Output: Expected output.
Start
Step 1 submits input in GUI
Step 2 Invoke Intelligence Agent session
           i) Analyze the input query
           ii) Scan, Parse, and Validate.
          iii) Assign the request to the concern processor
                Else
                     To any of the free process in the P2P network.
Step 3   The processor checks the incoming request whether belongs
            to local or neighbor
               // using the relation and attributes
                If (request==local)
   100:         {       Check the type of query operation (read/write)
                        If (read)
                             {fetch the relevance in the index cache and display}
                           Else
                               {execute the query, update database and display}
                           Else
                             Invoke coopsc approach ();
                             Return;
                  } Else
                 If (request == neighbor)
                  {//identify the neighbor using IP address and in index
                   If (neighbor== free)
                    {   Assign request to neighbor;
                           Goto 100;}
                      Else
                     {Execute the request locally and acknowledge the
                        neighbor ;}
                   return ();
                 }
End;
```

Procedure coop SC ( );

```
// Input partial or new write type query ();
// Output Expected output.
   Start
   Step 1 Scan the query, parse and split based on
             relation  relevance.
   Step 2 Assign the mapped relevance, matched as
            probe and UN matched as remainder.
   Step 3  Pass the remainder to neighbor and fetch
             the related relevance .
            // neighbor is identified using the cache reference point
            index_id (query_id, result_id)
   Step  4 Update the remainder relevance to the
             existing probe information and display.
   Step 5 Repeat steps 3 and 4 until the exact relevance
            Found, i.e., all the neighboring involved
             is checked.
            else
            Assign the query to the processor
            (database server) to execute.
  Step 6 Update the resultant in database as well as all
            the caches in the network
End.
```

5. Relevance Reference Caching Mechanism

In order to improve the access latency, we propose a novelty mechanism called relevance reference caching towards the prefix-indexed query information. This is actually inspiration based on reference point caching mechanism which had been deployed for reducing web latency. Reference point caching mechanism deploys a free number of proxies in client and server side bypassing the intermediates with indexing the IP-address in the cache and retrieving the documents from the server. In our case since we limit to the database (homogeneous or heterogeneous) to reduce the query latency and processor response time. A P2P network as in [13, 14] and intelligence along with fault tolerance was deployed. Directly the relevance query location was identified. That is user can interface with any client in the P2P network (post his request), every machine will act as a server (master) depending on the need, i.e., just to identify the query relevance and assigning the concern. If not write operation and if the concerned node or machine is busy then the user posted query will get executed on the local machine itself, and the output is displayed, later it is acknowledged, to the concern.

The concern is identified by using the reference caching technique to fetch the relevant if it is new information and its validity will be passed to the processor for execution. Analysing the user input, scanning, parsing, identifying the query type of operation, identifying the relevance, whether the query is partial, complete or new is the role of an intelligence agent. Here in P2P [14] network every machine is implicitly assigned to the exclusive database and explicitly generalized with the replicated information of the neighbouring databases. Here the question arises how about the storage cost, to overcome we are not actually storing the database it is on the secondary disk, we use a query prefix index along with the concern result ID. The data structure deployed is shown below in Tables 1 and 2.

Table 1. Reference client index.

Client index	**Relevance Client ID**		
IpAddress	IP-Address	Prefix_query index	Relevance_result Index
10.1.2.102	10.1.2.107	1001	101R

Table 2. Reference relevant query cache.

	User query [1]	**User query [5]**
Prefix query ID	SL(p1,p2)<p1=100>(R)	SL count(p1)(R)
Query index	Select p1,p2 from project where p1=100	Select p1 from project
Result ID	101 R	102 R
Result	Research	20

6. Empirical Test

The proposed work was implemented and tested over the real-time academic database called FFCS (Fully flexible credit system). This is a user-friendly academic environment which was widely used across universities in ASIA and US. Here the students are allowed to choose their course type, exam mode, faculty, class timing, and evaluation methodology, based on their choice corresponding to their level of credits to be obtained in the specified duration. Similarly, the faculty can choose his timing, type, of courses he or she wishes to handle, year of students, evaluation mode. Everything is allowed prior to the beginning of a semester. The sample database of more than thirty thousand students is taken from VIT University. The experimental workflow is viewed in four phases user interface, wrapping intelligent agent, cooperative semantic caches and database server. In detail information can be found in below URL in reference.www.vit.ac.in. A sample case was taken from this environment for testing the performance of proposed system. A student form CSE branch may wish to register course software maintenance. After authentication of the concerned user the system display option in view course page of the course displayed based on course ID, Course Title or Faculty who is going to teach. Most probably students may not be aware of courses or Faculty list. They tend to enter a title to enter database system, and then they get the displays of all the existing information in relevance. Tasks: one user with shared request, one user with one request, many users with a single request and many users with many requests.

Request means here a query relate to their departments. For example, consider the query one student want to register for university common subject, e.g., UNIX program his query of form select course from academics where course_name =' Unix programming' and slot ='d1'Whenever the user post a query in the system GUI, the intelligence agent checks the validity, based on his/her registration number as login, the user will have a concerned department, year of studying, the related course is identified For, e.g., 14CSE0200 here the 14 indicates the year the person joined cse indicates the branch and remaining the roll of the student. Firstly the query will be passed on to the concern department's processor and executed, if possible the data gets modified, and the relevant data gets stored in the log and displayed to the user. Since the request is universal, it will be passed on to the neighboring cache and gets checked for existences if possible, the data updated and acknowledged to the user relevance native record. Considering another case, if a student wants to register theory and lab under the same faculty.

Select Fname from course where course name ='Database' and Fname

IN (Select Fname from course where course name = 'Database lab').

In this case, the system first needs to filter the faculty who assigned for theory and lab corresponding to the same subject. Further, if there is a vacancy in the slot specified by student .if not it should check for any similarities from neighbouring set if exist then execute and display else process based on partiality else as new query execute and update result else any invalid reject the query. Like this, more than hundred students of five different departments with each of twenty-five queries were tested as samples. The observed status with proposed system is plotted in graphs. The metrics based on the query and its response time, user unsatisfied and satisfied, data validity and invalidity based, based on read-only and shared, read and write with exclusive were found in the defined system reaches the significance comparatively to the existing environment. Coming to cooperative semantic cache, the setup was made for implementation and evaluation of a database server and some clients that execute, in parallel, single and double indexed attribute selection queries with following three scenarios cooperative semantic caching approach, classic semantic caching and no caching approach. Every query is a range selection on either the unique attribute or more.

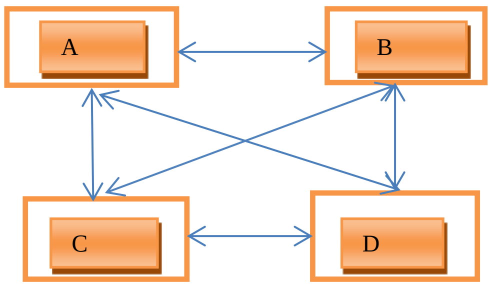

Fig. 4. P2 player approach.

For the experimental approach, the sample P2P layer approach deployed was shown in Fig.4. Here A, B, C, D are nodes in the network, to test the sample queries similar shown in above session cases this approach is used. The user queries from various schools (departments) were taken for testing. The user query is of the form as one user with shared request, one user with one request, many users with a single request and many users for many requests. The sample discussed was to show how the query gets processed by CoopSC. Wrapping agent as intelligence (a set of code) strives as preliminary to bring out optimization. It shows how the information relevant to the incoming query is identified, i.e., availability in client cache location. Not only for identification but also for the semantic knowledge used to transform a cache miss in a conventional caching to a cache hit as well as for providing cooperative capabilities conceptually and approximate query answering.

The index is used to identify the query relevance cache as well as the resultant in the log, here the universal index act as a look-up table to find out the relevance during execution. When a user posts the query, the query received by the intelligence agent, it parses the query and finds the details of attribute, value and relation and type of operation either read or write. If its read operation, any node will be assigned to take responsible for execution and acknowledged to the concern. If its write operation the exact cache where relevant data stored is identified and process to execute. In case if the particular is failure neighbouring node is assigned and allowed to process and execute. Later after failure recovery it is acknowledged and updated the modification. Thus satisfying fault tolerance.

The cache utilization was viewed as resultant cache, query log and data cache, a universal index which holds the information about the query-relevant cache details. The performance is analyzed based on a) query response time and (b) amount of data sent by the database server, c) tuples' origin (d) and hit rates based on the size of cache utilization with respect query. Amidst improvement on various aspects, we find a difference in load balancing during cooperative access while executing the query relevant to more than one relation with the write operation. Here actually the proposed work comes into play, i.e., cache reference with query index and the IP address. These give exactly which cache consist the query-relevant information. However, the problem which we observed in practical is during the initial request if the specified machine or processor is busy the how it can be executed. Though we deploy the fault tolerance mechanism it under goes a sort of delay, i.e., waiting state. For overcoming this issue, we the chord mechanism [15] in assigning the request to other free machine based on the algorithm deployed in [16] in order to balance the load of request and avoid unnecessary delay and over loading.

7. Graphical Representation of the Observed Resultant Significance

Figure 5 shows the observed results under semantic cache base system, cooperative semantic cache base system, and intelligence inference coop semantic cache system we observed and plotted with a difference of significance. The plotted by taking the number of user queries, that is a client with the response time inn minutes and tested with three type we observed that the response time relevant to intelligence inferred semantic query is minimum. Here in the graph, we can see that the initial time begins with little variants that 0.2, 0.3, and 0.5 instead of zero. It is a tolerance delay during the initial active time which can be covered while execution as due to intelligence inference time.

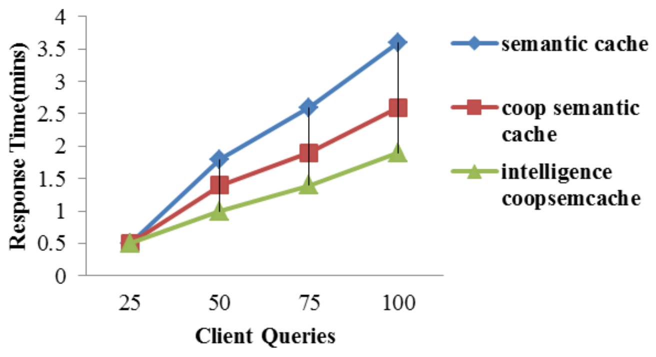

Fig. 5. Evaluation results.

8. Conclusions

Query processing is the key role in database applications irrespective of type and architectures. Researchers proved and provided various solutions from traditional to trends, we proposed our work for query processing optimality towards intelligence inference cooperative semantic caching approach as a novelty by using the relevance reference caching as the prefix queries in order to reduce the user access latency and minimize processor response time. We proposed an algorithm for our idea and performed an empirical test with the real-time academic application so called FFCS. Further, we plotted the difference of significance with our approach which is shown in the graph. Apart from this, we achieved fault tolerance also on behalf of improving system performance and user convenience. As a conclusion, we found an outcome with a considerable significance for query pace response time.

References

1. Dar, S.; Franklin, M.J.; Jónsson, B.P.; Srivastava, D.; and Tan, M. (1996). Semantic data caching and replacement. *Proceedings of the 22nd International Conference on Very Large Data Bases* (*VLDB*‘ 96), Mumbai (Bombay), India.
2. Chidlovskii, B.; Roncancio, C.; Schneider, M.-L. (1999). Semantic cache mechanism for heterogeneous web querying. *Xerox Research Centre Europe*, Grenoble Laboratory, France tutorials, 269-282.
3. Stuckenschmidt, H. (2004). Similarity-based query caching. *Proceeedings of International Conference on Flexible Query Answering Systems*, 295-306.
4. d'Orazio, L.; Roncancio, C.; Labbé, C.; and Jouanot, F. (2008). Semantic caching in large scale querying systems. *Revista Colombiana De Computación*, 9(1), 33-57.
5. Vancea, A.; and Stiller, B. (2009). Answering queries using cooperative semantic caching. *IFIP International Conference on Autonomous Infrastructure, Management and Security*, AIMS 2009: Scalability of Networks and Services, 203-206.
6. Lee, D.; and Chu, W.W. (2001). Towards intelligent semantic caching for web sources. *Journal of Intelligent Information Systems*, 17(1), 23-45.
7. Vancea, A.; and Stiller, B. (2010). CoopSC: A cooperative database caching architecture. *Proceedings of the 19th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2010)*, 223-228.
8. Xiaohui, L.; and Torsten, S. (2005). Three level caching for efficient query processing in large web search engines. *Proceedings of the 14th International Conference on World Wide Web WWW‘*05, 257-266.
9. Carey, M.J.; Franklin, M.J.; Livny, M.; and Shekita, E.J. (1991). Data caching tradeoffs in client-server DBMS architectures. *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data SIGMOD* '91, 357-366.
10. Denko, M.K.; and Tian, J. (2008). Cross-layer design for cooperative caching in mobile ad hoc networks. *Proceedings of 5th IEEE Consumer Communications and Networking Conference*, 375-380.
11. Yin, L.; and Cao, G. (2006). Supporting cooperative caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(1), 77-89.
12. Chand, N.; Joshi, R.C.; and Misra, C. (2006). Efficient cooperative caching in ad hoc networks communication system software and middleware. P*roceedings of First International Conference on Comsware*, 1-8.
13. Vaithianathan, S.K.; and Aravind (2001). *CHORD - P2P lookup protocol*. A tutorial. University of Central Florida, 15-20.
14. Khanaa, V.; and Thooyamani, K.P. (2014). Traffic balancing on co-operative proxy caching for peer to peer network. *World Applied Sciences Journal*, 29, 214-217.
15. Mohankumar, P.; and Vaideeswaran (2012). An easy-chair mechanism based algorithmic generalized approach for load balancing in database servers. *International Journal of Advanced Research in Computer Science*, 3(3), 54-57.
16. Chandranmenon, G.P.; and Varghese, G. (2001). Reducing web latency using reference point caching. *Proceedings of the IEEE Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM* 2001, 3, 1607-1616.