J.Natn.Sci.Foundation Sri Lanka 2017 45 (1): 13 - 22
DOI: http://dx.doi.org/10.4038/jnsfsr.v45i1.8033

RESEARCH ARTICLE

Indexing for semantic cache to reduce query matching complexity

Munir Ahmad[*], M. Abdul Qadir and Tariq Ali
Department of Computer Science, Faculty of Computing, Capital University of Science and Technology, Islamabad, Pakistan.

Revised: 04 August 2015; Accepted: 21 January 2016

Abstract: Cache is an important service to enable faster data access for distributed data storage systems. A cache system with a higher hit ratio and a lower query processing time is considered to be an efficient cache system. Semantic cache has a potential to maximise the hit ratio as compared to other cache organisation techniques because it has an ability to answer fully as well as partially overlapped queries. It is a challenge to determine the overlap between incoming queries and stored semantics (semantic matching) with minimum time. To store the semantics, one of the parameters which plays a significant role in the complexity of semantic matching is the indexing scheme. Existing indexing schemes store each disjoint query (on the base of projection) into different segments and the number of segments can grow to exponential (2^n-1, where n is the number of queried attributes) in the worst case. This will lead to an exponential complexity semantic matching scheme. We propose a schema based on semantic indexing that enhances the efficiency of semantic matching process by reducing the time for query matching significantly. In the proposed indexing scheme, we have merged query semantics with the schema. On the basis of the proposed indexing scheme, we have designed a query matching algorithm (*sMatch*). It is proved that the time complexity of *sMatch* is polynomial while the complexity of other matching algorithms is exponential. In addition to reducing the complexity of semantic matching, schema based storage of semantics would reject incorrect queries and enhance the hit ratio in cases where the non-schema based schemes would not find the data from the cache. To show the rejection of incorrect queries and improved hit ratio, complexity based and experimental results are presented. The results show that the proposed scheme performs significantly better than the previous ones.

Keywords: Distributed databases, knowledge management, query matching, query processing, semantic cache, semantic indexing.

INTRODUCTION

Faster information retrieval in a large distributed database system (DDBS) is a challenging task. Slower retrieval of data is the major problem in DDBS, especially when the network or server load is high (Chen & Roussopoulos, 1994; Godfrey & Gryz, 1997; Ahmad *et al*., 2009). The overall retrieval time can be reduced when the same data needs to be accessed frequently. A good caching near the front end (client machine) would definitely reduce the response time, increase the throughput, and provide fault-tolerance (Chen & Roussopoulos, 1994; Dar *et al*., 1996; Godfrey & Gryz, 1997; Ahmad *et al*., 2009).

A cache system does not come without a cost and has management overheads. To keep the overheads at a minimum, one has to manage the cache system intelligently (Ahmed *et al*., 2005). There are many techniques to manage cache systems such as adaptive database caching (Cluet *et al*., 1999; Altinel *et al*., 2003), semantic caching (Ren *et al*., 2003; Ahmed *et al*., 2005; 2008a; b; 2009; 2010; 2012; Ahmad & Qadir, 2009), page caching and tuple caching (Ren *et al*., 2003). A caching technique that stores data as well as semantics of executed queries is referred to as a semantic cache. Semantic cache enhances the performance of conventional cache by answering partially overlapped queries (Dar *et al*., 1996; Ahmad *et al*., 2008b). In semantic cache, the semantics of a newly posed query are matched with the stored semantics of already processed queries. On the basis of semantic matching, a user query is divided into two sub-queries: probe (portion available at cache) and remainder (portion that is not available at cache) queries (Dar *et al*.,

[*] Corresponding author (munirahmad83@gmail.com)

1996; Ren *et al*., 2003). Probe query is processed locally and remainder query is processed at the server side. There are two basic factors to evaluate the performance of a cache system; hit ratio (h_r), and query processing (Q_{pt}) time (Ahmad *et al*., 2008a; b).

Cache is called hit if data is found in it (Cai *et al*., 2005; Ahmad *et al*., 2008a) and hit ratio is the percentage of user posed queries that can be answered locally (partially or fully) from the cache. Therefore, the cache system should be designed in such a way that it increases the hit ratio. Improvement in hit ratio ensures efficient reuse of stored data (Ahmad *et al*., 2008b). Due to efficient reuse of stored data, lesser amount of data is required to be retrieved from remote locations. In this context it can be said that higher hit ratio ensures an efficient cache system.

Query trimming (Q_{tr}) and query rebuilding (Q_{rb}) are two major activities in query processing (Roussopouls, 1991). The query trimming process splits the user query into probe and remainder queries, whereas the query rebuilding process merges the results of probe and remainder queries. Query trimming process is further divided into two basic steps.

First, semantics of the newly posed query are matched with the stored semantics on cache, which is called query matching (Q_m). The query matching process helps out to enquire about the availability of data in cache. An efficient query matching process finds all the hidden semantics out of the stored semantics in cache. These findings of all hidden semantics increase the hit ratio. Optimum query matching ensures optimum query trimming. The query matching process depends on how the semantics of already processed queries are indexed (S_{ind}).

In the second step of query trimming, the user posed query is divided into two sub-queries; probe and remainder. The division process of a posed query depends upon the splitting algorithm as well as on how the semantics are indexed. Hence, the efficiency of query trimming depends on the indexing scheme.

The focus of this paper is to optimise the query trimming process. This goal is achieved by proposing an indexing scheme and a query matching algorithm. A critical analysis of existing indexing schemes has been done on defined criteria; hit ratio and query processing time. On the basis of the critical analysis, limitations of the existing schemes such as run time complexity and hit ratio have been highlighted. A schema based hierarchical semantic indexing scheme has been proposed that proved useful to overcome the limitations of the existing schemes. An algorithm for query matching (*sMatch)* is designed for *SELECT* and *PROJECT* queries by using the query splitting algorithm (Sun *et al*., 1989; Guo *et al*., 1996; Tariq *et al*., 2010). Complexity analysis of the proposed algorithm and existing schemes has been done. On the basis of complexity analysis, it has been found that the proposed indexing scheme is instrumental in decreasing the query matching complexity from *0* ($m \times n \times (2^n\text{-}1)$) to *0* ($m \times n$). *sMatch* has an ability to process '*Select**' type queries and due to this the hit ratio is increased. Finally, a comparison was done with the existing well-known algorithms proposed by Ren *et al*. (2003) and Ahmad *et al*. (2010) on the basis of hit ratio and query processing time. From the comparison we conclude that the proposed algorithm reduces the query processing time and increases the hit ratio significantly.

The following presents some definitions by using relational algebraic notations that are used in the paper.

Definition 1: *User query (Q_U)* will be represented by 5-tuple $< D, Q_A, Q_P, Q_R, P_A>$ where D is the name of the database, Q_A is a set of required attributes, Q_R is a relation, Q_P is a condition and P_A is a set of attributes in predicate.

Definition 2: Given a database $D = \{R_i\}$ and its attributes set $A = A_{Ri}$, $1 \leq i \leq n$, *semantic enabled schema,* Q_C will be 6-tuple $< D, S_A, S_P, S_R, S_{SA}, C >$ where D is the name of the database, S_R is the name of relation, S_A is a set of attributes, S_{SA} is a status of attributes, S_P is the predicate (condition) on which data has been retrieved and cached, and C is the reference of contents.

Definition 3: Given a user query $Q_U < D, Q_A, Q_P, Q_R >$ and $Q_C < D, S_A, S_P, S_R, S_{SA}, C >$; Dataset, D_U and D_C will be the retrieved rows in the execution of Q_U and Q_C, respectively.

Definition 4: Given a user query Q_U and cached query Q_C, *probe query (pq)* will be $Q_U \cap Q_C$ and dataset against *pq* will be $D_U \cap D_C$.

Definition 5: Given a user query Q_U and cached query Q_C, *remainder query (rq)* will be $Q_U - Q_C$ and dataset against r*q* will be $(D_U - D_C)$.

Definition 6: Given a user predicate Q_P and cached predicate S_P, *predicate implication* ($Q_P \rightarrow S_P$) holds if and only if $(Q_P \cap S_P) - Q_P = \Phi$.

Definition 7: Given a user predicate Q_P and cached predicate S_P, *predicate satisfiability* holds if and only if $Q_P \cap S_P \,! = \Phi$.

Definition 8: Given a user predicate Q_P and cached predicate S_P, *predicate unsatisfiability* holds if and only if $Q_P - S_P = Q_P$.

Definition 9: Given a user query Q_U and cached query Q_C, *query implications* ($Q_U \rightarrow Q_C$) holds if and only if $Q_A\ S_A$ and $Q_P \rightarrow S_P$.

Definition 10: Given a user query Q_U and cached query Q_C, *query satisfiability* holds if and only if $Q_A \cap S_A\ != \Phi$ and $Q_P \cap S_P\ != \Phi$.

Definition 11: Given a user query Q_U and cached query Q_C, *query unsatisfiability* holds either $Q_P \cap S_A = \Phi$ or $Q_P \cap S_P = \Phi$.

Definition 12: Given a user query Q_U and cached query Q_C, *common attributes* (C_A) is a set of attributes which are common among user and cached queries and will be computed as $C_A = Q_A \cap S_A$.

Definition 13: Given a user query Q_U and cached query Q_C; *difference attributes* (D_A) is the set of attributes, which exists in user query but not in cached query and will be computed as $D_A = Q_A - S_A$.

Semantic caching has been extensively studied by researchers in both relational and XML databases (Lee & Chu, 1999; Luo *et al*., 2000; Chen *et al*., 2002; Sumalatha *et al*., 2007b; 2007c; Sanaullah *et al*., 2008). An effort has been made to build a semantic caching system on the basis of description logic (Tariq *et al*., 2010) but failed to answer the overlapped queries locally. The related work is discussed on the basis of semantic indexing and query (*SELECT* and *PROJECT*) trimming in relational data semantic cache.

Initially, flat structure (semantics of query was stored without any specific structure) based query trimming techniques were proposed by Dar *et al*. (1996). Flat structure based query trimming process proved to be expensive (Roussopouls, 1991; Ahmad *et al.,* 2009) in terms of runtime complexity due to no indexing strategy. This may work for a very small size cache, when the number of queries stored in it are a few tens. To overcome this limitation the cache was organised into chunks (Deshpande *et al*., 1998; Ren *et al*., 2003) and segments (Godfrey & Gryz, 1997; Makki & Andrei, 2009; Makki & Rockey, 2010). Query trimming process improved up to some extent due to indexing the semantics in the form of segments (Ahmad *et al.,* 2008a; 2009). In the presence of segments, query matching (basic step of query trimming) still has high runtime complexity (Ahmad *et al*., 2009) due to the large number of possible segments (there can be '$2^n - 1$' possible segments for a relation having 'n' attributes). Segment based indexing scheme is used in most of the query trimming techniques (Godfrey & Gryz, 1997; Ren *et al*., 2003; Cai *et al*., 2005; Jonsson *et al*., 2006; Kang *et al*., 2006; Makki & Andrei, 2009; Makki & Rockey, 2010). All of these have higher runtime complexity due to the large number of possible segments. Makki and Rockey (2010) proposed the concept of query visualisation to optimise the query trimming process (Makki & Andrei, 2009; Makki & Rockey, 2010). They claimed and discussed some scenarios to prove that previous schemes were unable to trim the query in an optimal time. They improved the runtime complexity with the help of query visualisation concept. Still, this scheme was expensive because its query matching process depended on the segment based indexing scheme. Query refinement was also used to optimise the query trimming process that enhances semantic caching system (Keller & Basu, 1996; Chakrabarti *et al*., 2000). This scheme was also expensive due to its dependency on segment based indexing scheme. A 3-level indexing scheme (Sumalatha *et al*., 2007a; b) was proposed to overcome the limitation of the segment based scheme. It improved the query matching process but query trimming was not clear in that scheme. Bashir and Qadir (2006) presented a hierarchical scheme (4-HiSIS) for query matching. Although 4-HiSIS is a better approach for query matching there was no scheme defined for 4-HiSIS that covers the query trimming process. A scheme based on content matching was presented by Bashir *et al*. (2007) to improve the efficiency of semantic cache. There were 112 rules defined and merged with 4-HiSIS to split the query into probe and remainder queries (Bashir & Qadir, 2007). This work only coverd simple predicates and failed to split complex (having more than one comparison and / or more than zero conjunct operators) queries. 4-HiSIS was merged with satisfiability/implication, which improved the runtime complexity due to 4-HiSIS (Bashir & Qadir, 2006). Still this scheme was not appropriate to trim the query because it was designed for single relation and was only applicable for single predicate. To improve this, semantic matching process has been enhanced by presenting the graph based semantic indexing scheme (Ahmad *et al*., 2010). This enhanced scheme achieved the required efficiency in terms of runtime complexity but failed to improve hit ratio, because the graph based scheme was unable to handle the '*SELECT* * ' type queries.

If a user pose a new query as given below;

*SELECT * FROM* Books *WHERE* Author = 'Ali'

None of the previous schemes in literature was able to match the semantic of this query with the stored semantics. From the above we conclude that there is a need of an efficient indexing scheme that must achieve the required hit ratio and perform semantic matching efficiently. Therefore, we have presented an efficient semantic indexing that is able to achieve both goals.

METHODOLOGY

One of the challenges in query processing is efficient query matching. Therefore, to make the query processing efficient there is a need to make the process of query matching accurate and faster. Among the five different strategies used for query matching, graph based semantic indexing (Ahmad *et al*., 2010) scheme is the most efficient. This approach was used to determine whether or not the required attributes are available at cache. After finding the status of the required attributes this scheme behaves like a segment based scheme for building probe and remainder queries, which is costly and time consuming.

Schema is required to be stored in semantic cache to process a query (Ahmad *et al*., 2008b; 2009). Therefore, we have merged the query's semantics with schema. The main advantage of this amalgamation is preserving semantics of the query efficiently by using schema. However, in previous studies (Ahmad *et al*., 2008b; 2009) semantics and schema were stored separately in cache, which demands extra memory. In this scheme semantics are associated with stored schema (semantic enabled schema). This is called a schema based hierarchical indexing scheme for semantic cache. Initially, the schema for each database is stored in cache with database names, relation names and attribute names with false status in the form of a tree as given in Table 1. Due to keeping the schema of database its space complexity will be higher than the previous techniques until 'n' attributes are cached or 'n' distinct queries are processed.

Table 1: Schema based hierarchical semantic indexing

DB name	Table name	Fields	Status	Condition	Content
		A_1	False	Null	Null
	R_1	A_2	False	Null	Null
DB		A_3	False	Null	Null
	R_2	A_1	False	Null	Null
		A_2	False	Null	Null

An example of library database is given in Table 2.

Table 2: Schema based hierarchical semantic indexing for library

DB name	Table name	Fields	Status	Condition	Content
	Books	Author	False	Null	Null
		Title	False	Null	Null
Library		ISBN	False	Null	Null
	Journals	Author	False	Null	Null
		Title	False	Null	Null

Now suppose that a user enters a query as given below:

SELECT Author, Title *FROM* Books *WHERE* Author = 'Ali'

After execution of the above query the retrieved contents will be stored (assume the contents will be stored with name 1; just like materialised view) and the semantics will be updated as given in Table 3.

Table 3: Updated indexed semantics

DB Name	Table name	Fields	Status	Condition	Content
	Books	Author	True	Author = "Ali"	1
		Title	True	Author = "Ali"	1
Library		ISBN	False	Null	Null
	Journals	Author	False	Null	Null
		Title	False	Null	Null

Author and Title across books are posed to retrieve in the query so their status will be changed to true. Condition and content reference will also be updated accordingly. After managing semantics, the next step is to use the managed semantics. For content matching, first the database names will be matched exactly; secondly if database name is matched then the relation names will be matched; otherwise processing will be stopped. After

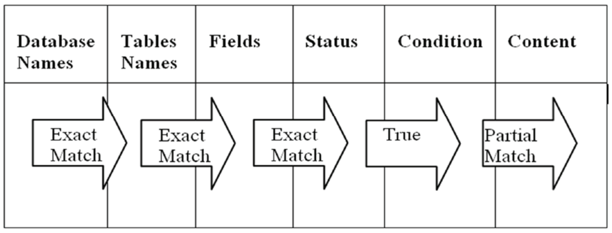

Figure 1: 6HiSIS: Schema based semantic matching

exactly matching the relation name, attribute names are matched. When it is found that attributes are part of the schema, then their status is checked. In case of true status of attributes, data across particular attributes will be available; otherwise it will be retrieved from the server. If status of the attribute is also true then condition across a particular row is matched. Finally probe query is generated with the generated condition from the referenced content. Hierarchical schema based semantic matching is given in Figure 1.

Due to the schema based hierarchical semantic indexing scheme we are able to perform query matching in linear fashion and to handle the SELECT * type queries.

There is a simple driver algorithm (*sMatch*) to perform the query matching as given in Figure 2.

$$sMatch(\mathrm{D_A}, \mathrm{Q_P}, \mathrm{S_P}, \mathrm{C_A})$$
$$= \begin{cases} \mathrm{rq1} := \pi_{\mathrm{D_A}}\sigma_{\mathrm{Q_P}}(\mathrm{Q_R}) \text{ and } \mathrm{pq} := \mathrm{NULL} \text{ and } \mathrm{rq2} := \mathrm{NULL}, & \mathrm{D_A}! = \Phi \text{ and } \mathrm{C_A} = \Phi \\ \mathrm{rq1} := \pi_{\mathrm{D_A}}\sigma_{\mathrm{Q_P}}(\mathrm{Q_R}) \text{ and } \mathrm{pq} := \pi_{\mathrm{C_A}}\sigma_{\mathrm{Q_P}}(\mathrm{Q_R}) \text{ and } \mathrm{rq2} := \mathrm{NULL}, & \mathrm{Q_P} \rightarrow \mathrm{S_P} \text{ and } \mathrm{D_A}! = \Phi \text{ and} \mathrm{C_A}! = \Phi \\ \mathrm{rq1} := \mathrm{NULL} \text{ and } \mathrm{pq} := \pi_{\mathrm{C_A}}\sigma_{\mathrm{Q_P}}(\mathrm{Q_R}) \text{ and } \mathrm{rq2} := \mathrm{NULL}, & \mathrm{Q_P} \rightarrow \mathrm{S_P} \text{ and } \mathrm{D_A} = \Phi \text{ and} \mathrm{C_A}! = \Phi \\ \mathrm{rq1} := \pi_{\mathrm{D_A}}\sigma_{\mathrm{Q_P}}(\mathrm{Q_R}) \text{ and } \mathrm{pq} := \pi_{\mathrm{C_A}}\sigma_{\mathrm{Q_P}}(\mathrm{Q_R}) \text{ and } \mathrm{rq2} := \pi_{\mathrm{C_A}}\sigma_{Q_P \urcorner \mathrm{S_P}}(\mathrm{Q_R}), & \mathrm{Q_P}\wedge\mathrm{S_P} \text{ is satisfiable and } \mathrm{C_A}! = \Phi \text{ and } \mathrm{D_A}! = \Phi \\ \mathrm{rq1} := \mathrm{NULL} \text{ and } \mathrm{pq} := \pi_{\mathrm{C_A}}\sigma_{\mathrm{Q_P}}(\mathrm{Q_R}) \text{ and } \mathrm{rq2} := \pi_{\mathrm{C_A}}\sigma_{Q_P \urcorner \mathrm{S_P}}(\mathrm{Q_R}), & \mathrm{Q_P}\wedge\mathrm{S_P} \text{ is satisfiable and } \mathrm{C_A}! = \Phi \text{ and } \mathrm{D_A} = \Phi \end{cases}$$

Figure 2: Working algorithm *sMatch*

MATHEMATICAL PROOF AND RESULTS

A comparison of the proposed *sMatch* with previous studies by Ren *et al*. (2003) and Ahmad *et al*. (2010) was done. The comparison is given in different aspects; run time complexity, hit ratio, and handling of incorrect and '*SELECT* * ' type queries.

At first, we compared the query matching complexity of the proposed semantic indexing scheme with the segment based semantic indexing scheme. We calculated the complexity of both as follows.

Theorem1: For a relation R having attribute set $A = Ai$, where $1 \leq i \leq n$; query matching complexity for segment based scheme is '$m \times n \times 2^n\text{-}1$' where '$m$' is the number of required attributes in user query and 'n' is the number of total attributes in a relation.

Proof: (Constructive)

Segment based query matching scheme depends on the number of segments and the number of attributes in each segment. The number of segments in cache depends on the number of attributes in a relation. Possible segments for a relation having 'n' attributes are proved in lemma 1.1.

Lemma 1.1: For a relation R having attribute set $A = Ai$, where $1 \leq i \leq n$, the maximum number of segments is $2^n\text{-}1$.

Proof: Number of attributes in a relation $R = n$

Number of subsets, P, for n attributes can be computed as follows;

$$|P(n)| = 2^n \qquad ...(1)$$

Number of disjoint queries (D_{Qu}) on relation will be equal to the subsets except the empty set.

$$|D_{Qu}| = P(n)\text{- } \Phi \qquad ...(2)$$

There will be only one empty subset in $P(n)$. By replacing values in equation 2 from equation 1 we get;

$$|D_{Qu}| = 2^n - 1 \qquad ...(3)$$

As we know (Ren *et al*., 2003) that the number of segments $|S|$ on cache will be equal to the number of disjoint queries. i.e.

$$|S| = |D_{Qu}| \qquad ...(4)$$

By replacing the value into equation 4 from equation 1, we get;

$$|S| = 2^n - 1$$

Hence lemma 1.1 proved.

Comparisons required to find a single attribute over a number of segments $|S|$ in worst case are $2^n - 1$ as proved in lemma 1.2.

***Lemma 1.2**:* For a relation R having attribute set $A = Ai$, where $1 \leq i \leq n$, $n \times 2^{n-1}$ number of comparisons (NoC_n) are required to find a single attribute Ai over a number of segments $|S|$.

***Proof**:* Finding a single attribute Ai over a number of segments $|S|$ required to visit each and every attribute in each segment.

For a single attribute in a relation, there can be only one segment (according to lemma 1.1) and the number of comparisons (NoC_n) required for query matching will also be one, which can be computed as follows:

$NoC_n = 2^1 - 1$

For two attributes, there will be three segments (according to lemma 1.1). The number of comparisons (NoC_n) required for query matching in this case will be four and can be computed as,

$$\begin{aligned} NoC_n &= 2^2 - 1 + (2^1 - 1) \\ &= 2^2 - 1 + 2^0(2^1 - 1) \\ &= 2^2 - 1 + 2^0(2^{2-1} - 1) \end{aligned}$$

For three attributes, there will be three segments and the number of comparisons (NoC_n) required for query matching is 12 and can be computed as follows:

$$\begin{aligned} NoC_n &= 2^3 - 1 + 2^2 - 1 + 2^1 - 1 + 2^1 - 1 \\ &= 2^3 - 1 + 2^2 - 1 + 2(2^1 - 1) \\ &= 2^3 - 1 + 2^0(2^2 - 1) + 2^1(2^1 - 1) \\ &= 2^3 - 1 + 2^0(2^{3-1-0} - 1) + 2^1(2^{3-1-1} - 1) \end{aligned}$$

For four attributes, there will be three segments and the numbers of comparisons (NoC_n) required for query matching is 32 and can be computed as follows:

$$\begin{aligned} NoC_n &= 2^4 - 1 + 2^3 - 1 + 2^2 - 1 + 2^1 - 1 + 2^1 - 1 + 2^2 - 1 + 2^1 - 1 \\ &\quad + 2^1 - 1 \\ &= 2^4 - 1 + 2^3 - 1 + 2^2 - 1 + 2^2 - 1 + 2^1 - 1 + 2^1 - 1 + 2^1 - \\ &\quad 1 + 2^1 - 1 \\ &= 2^4 - 1 + 2^3 - 1 + 2(2^2 - 1) + 4(2^1 - 1) \\ &= 2^4 - 1 + 2^0(2^3 - 1) + 2^1(2^2 - 1) + 2^2(2^1 - 1) \\ &= 2^4 - 1 + 2^0(2^{4-1-0} - 1) + 2^1(2^{4-1-1} - 1) + 2^2(2^{4-2-1} - 1) \end{aligned}$$

Similarly for five attributes the number of comparisons can be computed as follows:

$$\begin{aligned} NoC_n &= 2^5 - 1 + 2^0(2^4 - 1) + 2^1(2^3 - 1 + 2^2(2^2 - 1) + 2^3(2^1 - 1) \\ &= 2^5 - 1 + 2^0(2^{5-0-1} - 1) + 2^1(2^{5-1-1} - 1 + 2^2(2^{5-2-1} - 1) \\ &\quad + 2^3(2^{5-3-1} - 1) \end{aligned}$$

For 'n' attributes the number of comparisons (NoC_n) can be computed as follows:

$$\begin{aligned} NoC_n &= 2^n - 1 + 2^0(2^{n-0-1} - 1) + 2^1(2^{n-1-1} - 1 + 2^2(2^{n-2-1} - 1) \\ &\quad + \ldots\ldots + 2^{n-3}(2^{n-n-3-1} - 1) + 2^{n-2}(2^{n-n-2-1} - 1) \end{aligned}$$

We can write it as

$$\begin{aligned} NoC_n &= 2^n - 1 + \sum_{r=0}^{r=n-1} (2^r\,(2^{n-r-1} - 1)) \\ &= 2^n - 1 + \sum_{r=0}^{r=n-1} (2^r \,.\, 2^{n-r-1}) - \sum_{r=0}^{r=n-1} 2^r \\ &= 2^n - 1 + \sum_{r=0}^{r=n-1} (\,2^{n-r+r-1}) - \sum_{r=0}^{r=n-1} 2^r \\ &= 2^n - 1 + \sum_{r=0}^{r=n-1} (\,2^{n-1}) - \sum_{r=0}^{r=n-1} 2^r \qquad \ldots(1) \end{aligned}$$

By using geometric series we know that

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

So, equation 1 becomes

$$\begin{aligned} NoC_n &= 2^n - 1 + \sum_{r=0}^{r=n-1} 1 - (2^{n-1+1} - 1) \\ &= 2^n - 1 + 2^{n-1}.n - 2^n + 1 \\ &= n.\,2^{n-1} \end{aligned}$$

Hence lemma 1.2 proved.

In lemma 1.2 we have proved that the query matching complexity for finding a single attribute over $|S|$ is $n.\,2^{n-1}$.

It is obvious that this complexity will be '$m.n.2^{n-1}$' to determine the 'm' attributes over relation R having 'n' attributes. The above calculated comparisons are derived from the stored segments in cache. Now, if there are 'm' attributes required in user query then the total comparison will be '$m \times n \times 2^{n-1}$'. Hence the computation complexity of segments based will be '$m \times n \times 2^{n-1}$'; where '$m$' is the number of attributes in user posed query and 'n' is the number of attributes in a relation.

Hence theorem 1 proved.

Corraly 1.1: In best case the query matching complexity for segment based scheme will be '$m.n.2^{n-1}$'. In best case each required attribute will be determined at first location of segment. According to lemma 1.1 the total numbers of segments ($|S|$) will be 2^n-1. For each attribute 'm' we have to visit 2^n-1 segments. In this case query matching complexity will be '$m.n.2^n$-1'.

Theorem 2: For a relation R having attribute set $A = UAi$, where $1 \leq i \leq n$, query matching complexity for schema based scheme is 'm x n' where 'm' is the number of required attributes in user query and 'n' is the number of total attributes in a relation.

Proof: This proof is very straightforward, because it indexes the semantic enabled schema instead of semantic segments. The number of attributes will always be 'n' irrespective of the number of disjoint queries. So in worst case NoC_n will be 'm x n' to determine 'm' attributes over R.

Hence, the worst case complexity of the segment based scheme is exponential while the worst case complexity of the schema based scheme will be polynomial, which can be plotted as given in Figure 3.

Similarly, best case complexity analysis of the schema based and segment based scheme is given in Figure 4. We assumed that each comparison will take one millisecond to compute.

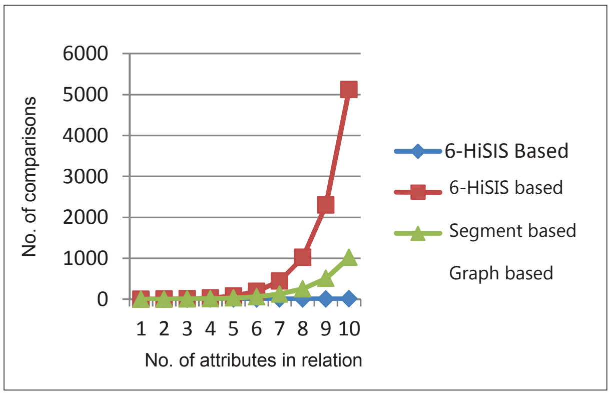

Figure 3: Worst case complexity analysis

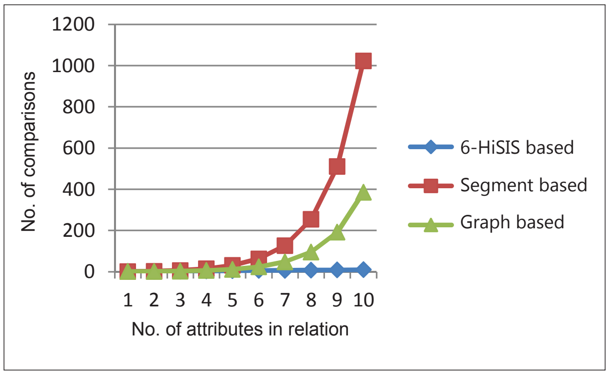

Figure 4: Best case complexity analysis

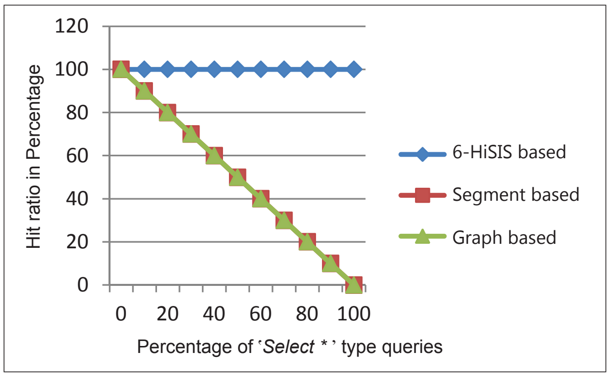

Figure 5: Hit ratio comparison by increasing Select * queries

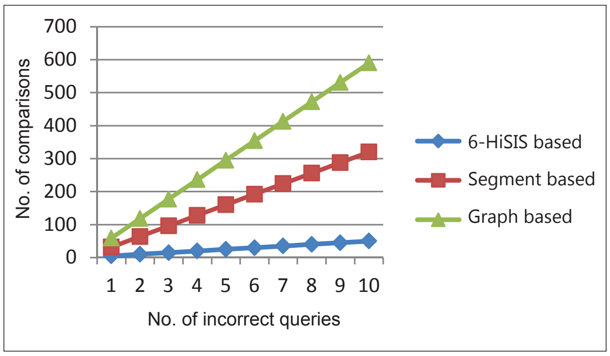

Figure 6: Time comparison on the base of incorrect queries

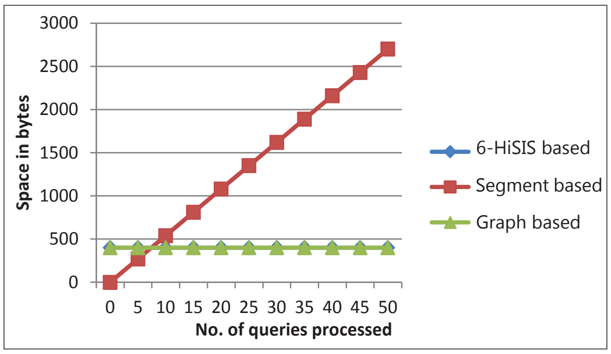

Figure 7: Space complexity analysis

In Figure 5, the comparison between segment based (Ren *et al*., 2003), graph based (Ahmad *et al*., 2010) and 6-HiSIS based schemes on the basis of hit ratio is given. In this comparison we are assuming that all of the data is available (hit ratio should be 100 %). Now, assume that a user poses queries with '*' operator (Select * type queries), which indicates that the user needs all of the attributes. As soon as we increase this type of queries, the hit ratio for graph based and segment based will be decreased, while the hit ratio for 6-HiSIS based scheme will remain at 100 %. This happens because the previous (segment and graph based) techniques are unable to handle 'Select *' type queries and retrieve all the data from the server. The 6-HiSIS based scheme evaluate this type of queries by using schema. Therefore, the hit ratio by 6-HiSIS based scheme remains at 100 %.

In Figure 6, time comparison between segment based (Ren *et al*., 2003), graph based (Ahmad *et al*., 2010) and 6-HiSIS based schemes on the basis of incorrect user queries is given. In Figure 6 we are assuming that the user is going to pose incorrect queries. As soon as we increase the number of incorrect queries, the computing time for graph based and segment based schemes will be increased while the hit ratio for 6-HiSIS based scheme will remain at 100 %. This happens because the previous techniques are unable to reject incorrect queries. In contrast, 6-HiSIS rejects incorrect queries at the initial level. Due to this, the proposed techniques perform better.

In Figure 7, space complexity for segment based (Ren *et al*., 2003), graph based (Ahmad *et al*., 2010) and 6-HiSIS based schemes is given. As we have discussed earlier, space complexity will be higher than the previous schemes until '*n*' attributes are retrieved or '*n*' distinct queries have been processed and their semantics are cached in semantic cache.

CONCLUSION

Query processing has a major influence on the efficiency of the caching system. One of the major activities of query processing is query matching. Efficient query matching ensures efficient query processing. Query matching depends on the semantic indexing scheme. An efficient semantic indexing scheme ensures efficient query matching. In this paper we have presented an efficient semantic indexing scheme (6-HiSIS), which reduces the runtime complexity from exponential to polynomial.

REFERENCES

1. Ahmad M., Asghar S., Qadir M.A. & Ali T. (2010). Graph based query trimming algorithm for relational data semantic cache. *MEDES'10 - Proceedings of the International Conference on Management of Emergent Digital Ecosystems*, Bangkok, Thailand, pp. 47 – 52.
2. Ahmad M. & Qadir M.A. (2009). Query processing and enhanced semantic indexing for relational data semantic cache. *MSc thesis*, Mohammad Ali Jinnah University, Islamabad, Pakistan.
3. Ahmad M., Qadir M.A., Ali T., Abbas M.A. & Afzal M.T. (2012). Semantic cache system. *Semantics in Action – Applications and Scenarios* (ed. M.T. Afzal), chapter 4. INTECH Open Access Publisher, Rijeka, Croatia - European Union.
DOI: https://doi.org/10.5772/38862
4. Ahmad M., Qadir M.A. & Sanaullah M. (2008a). Query processing over relational databases with semantic cache: a survey. *IEEE International Multitopic Conference, INMIC 2008*, 23 – 24 December, Karachi, Pakistan, pp. 558 – 564.
DOI: https://doi.org/10.1109/INMIC.2008.4777801
5. Ahmad M., Qadir M.A., Razzaque A. & Sanaullah M. (2008b). Efficient query processing over semantic cache. *Intelligent Systems and Agents, ISA 2008,* indexed by IADIS digital library (*www.iadis.net/dl*). Held within IADIS Multi Conference on Computer Science and Information Systems (MCCSIS 2008), Amsterdam, The Netherlands, pp. 22 – 27.
6. Ahmad M., Qadir M.A. & Sanaullah M. (2009). An efficient query matching algorithm for relational data semantic cache. *2nd IEEE Conference on Computer, Control and Communication, IC409,* 17 – 18 February, Karachi, Pakistan, pp. 1 – 6.
7. Ahmed M.U., Zaheer R.A. & Qadir M.A. (2005). Intelligent cache management for data grid. *Proceedings of the Australasian Workshop on Grid Computing and E-Research*, New South Wales, Australia, pp. 05 – 12.
8. Altınel M., Bornhövd C., Krishnamurthy C., Mohan C., Pirahesh H. & Reinwald B. (2003). Cache tables: paving the way for an adaptive database cache. *Proceedings of the 29th International Conference on Very Large Databases,* Berlin, Germany, pp. 718 – 729.
DOI: https://doi.org/10.1016/b978-012722442-8/50069-0
9. Bashir M.F. & Qadir M.A. (2006). HiSIS: 4-level hierarchical semantic indexing for efficient content matching over semantic cache. *IEEE International Multitopic Conference, INMIC 2006*, 23 – 24 December, Islamabad, Pakistan, pp. 211 – 214.
DOI: https://doi.org/10.1109/INMIC.2006.358165
10. Bashir M.F. & Qadir M.A. (2007). *ProQ-Query Processing Over Semantic Cache for Data Grid*. Centre for Distributed and Semantic Computing, Mohammad Ali Jinnah University, Islamabad, Pakistan.

11. Bashir M.F., Zaheer R.A., Shams Z.M. & Qadir M.A. (2007). SCAM: semantic caching architecture for efficient content matching over data grid. *Advances in Web Intelligence*, pp. 41 – 46. Springer, Heidelberg, Berlin, Germany.
DOI: https://doi.org/10.1007/978-3-540-72575-6_7
12. Cai J., Jia Y., Yang S. & Zou P. (2005). *A Method of Aggregate Query Matching in Semantic Cache for Massive Database Applications*, pp. 435 – 442. Springer, Heidelberg, Berlin, Germany.
DOI: https://doi.org/10.1007/11573937_47
13. Chakrabarti K., Porkaew K. & Mehrotra S. (2000). Efficient query refinement in multimedia databases. *Proceedings of the 16th IEEE International Conference on Data Engineering*, 03 March, San Diego, USA, pp. 196.
DOI: https://doi.org/10.1109/icde.2000.839410
14. Chen C.M. & Roussopoulos N. (1994). The implementation and performance evaluation of the adms query optimizer: integrating query result caching and matching. *Proceedings of the International Conference on Extending Database Technology*, Cambridge, UK, 28 – 31 March, pp. 323 – 336.
DOI: https://doi.org/10.1007/3-540-57818-8_61
15. Chen L., Rundesteiner E.A. & Wang S. (2002). XCachea semantic caching system for XML queries. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data,* Madison, USA, pp. 618.
DOI: https://doi.org/10.1145/564691.564771
16. Cluet S., Kapitskaia O. & Srivastava D. (1999). Using LDAP directory caches, *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems,* Philadelphia, Pennsylvania, USA, 31 May – 03 June, pp. 273 – 284.
DOI: https://doi.org/10.1145/303976.304003
17. Dar S., Franklin M.J. & Jonnson B.T. (1996). Semantic data caching and replacement. *Proceedings of the 22nd International Conference on Very Large Databases*, San Francisco, USA, 03 – 06 September, pp. 330 – 341.
18. Deshpande P.M., Ramasamy K. & Shukla A. (1998). Caching multidimensional queries using chunks. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, Washington DC, USA, 01 – 04 June, pp. 259 – 270.
DOI: https://doi.org/10.1145/276304.276328
19. Godfrey P. & Gryz J. (1997). Semantic query caching for heterogeneous databases. *Proceedings of the 4th KRDB Workshop on Intelligent Access to Heterogeneous Information*, Athens, Greece, 30 August, pp. 61 – 66.
20. Guo S., Sun W. & Weiss M.A. (1996). Solving satisfiability and implication problems in database systems. *ACM Transactions on Database Systems* **21**(2): 270 – 293.
DOI: https://doi.org/10.1145/232616.232692
21. Jonsson B.T., Arinbjarnar M., Thorsson B., Franklin M. & Srivastava D. (2006). Performance and overhead of semantic cache management. *ACM Transactions on Internet Technology* **6**(2): 302 – 331.
DOI: https://doi.org/10.1145/1151087.1151091
22. Kang S.W., Kim J., Im S., Jung H. & Hwang C.S. (2006). Cache strategies for semantic prefetching data. *7th International Conference on Web-Age Information Management Workshops,* Washington DC, USA, 17 – 19 June, pp. 7.
DOI: https://doi.org/10.1109/waimw.2006.7
23. Keller A.M. & Basu J. (1996). A predicate-based caching scheme for client-server database architectures. *The International Journal on Very Large Databases* **5**(1): 35 – 47.
DOI: https://doi.org/10.1007/s007780050014
24. Lee D. & Chu W.W. (1999). Semantic caching *via* query matching for web sources, *Proceedings of the 8th International Conference on Information and Knowledge Management*, Kansas City, USA, 02 – 06 November, pp. 77 – 85.
DOI: https://doi.org/10.1145/319950.319960
25. Luo Q., Naughton J.F., Krishnamurthy R., Cao P. & Li Y. (2000). Active query caching for database web servers. *3rd International Workshop WebDB200 on The World Wide Web and Databases*, London, UK, 18 – 19 May, pp. 92 – 104.
26. Makki S.M. & Andrei S. (2009). Utilizing semantic caching in ubiquitous environment. *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing*, Leipzig, Germany, 21 – 24 June, pp. 1213 – 1217.
DOI: https://doi.org/10.1145/1582379.1582644
27. Makki S.M. & Rockey M. (2010). Utilizing semantic caching in ubiquitous environment. *IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*.
28. Ren Q., Dunham M.H. & Kumar V. (2003). Semantic caching and query processing. *IEEE Transactions on Knowledge and Data Engineering* **15**(1): 192 – 210.
DOI: https://doi.org/10.1109/TKDE.2003.1161590
29. Roussopoulos N. (1991). An incremental access method for view cache: concept, algorithms, and cost analysis. *ACM Transactions on Database Systems* **16**(3): 535 – 563.
DOI: https://doi.org/10.1145/111197.111215
30. Sanaullah M., Qadir M.A. & Ahmad M. (2008). SCAD-XML: semantic cache architecture for XML data files using XPath with cases and rules. *IEEE International Multitopic Conference, INMIC 2008*, 23 – 24 December, Karachi, Pakistan, pp. 361 – 367.
DOI: https://doi.org/10.1109/INMIC.2008.4777764
31. Sumalatha M.R., Vaidehi V., Kannen A., Rajasekar M. & Karthigaiselven M. (2007a). Hash mapping strategy for improving retrieval effectiveness in semantic cache system. *IEEE International Conference on Signal Processing, Communications and Networking*, 22 – 24 February, Chennai, India, pp. 233 – 237.
DOI: https://doi.org/10.1109/icscn.2007.350737
32. Sumalatha M.R., Vaidehi V., Kannen A., Rajasekar M. & Karthigaiselven M. (2007b) Dynamic rule set mapping strategy for the design of effective semantic cache. *IEEE 9th International Conference on Advanced Communications Technology*, 07 May, Gangwon-Do, Korea, pp. 1952 – 1957.
DOI: https://doi.org/10.1109/icact.2007.358753
33. Sumalatha M.R., Vaidehi V., Kannen A., Rajasekar M. & Karthigaiselven M. (2007c). Xml query processing – semantic cache system. *International Journal of Computer Science and Network Security* **7**(4): 164 – 169.

34. Sun X., Kamel N.N. & Ni L.M. (1989). Processing implication on queries. *IEEE Transactions on Software Engineering* **15**(10): 1168 – 1175.
DOI: https://doi.org/10.1109/TSE.1989.559764

35. Tariq M., Qadir M.A. & Ahmad M. (2010). Description logic for semantic cache query processing. *International Conference on Information and Emerging Technologies*, 14 – 16 June, Karachi, Pakistan.