Saint-Petersburg State University
Mathematics and Mechanics Faculty
Software Engineering Department

# Semester work

## An Analysis Of Applicability Of Similarity-Based Semantic Caching

Performed by
student of 445 group
Anna Safonova

Scientific Adviser:
Boris Novikov
Dr. Sci. degree, professor,
Dept. of Computer Science,
St-Petersburg University

Reviewer:
Maksim Lukichev

Saint-Petersburg
2010

# Contents

# Introduction

The need to uniformly access semistructured, heterogeneous information resources triggered a huge amount of research in the area of information integration performed in last decades. Several industrial strength integration software tools are currently available. However, the query evaluation performance is still far from practically acceptable for any complex queries and is often unpredictable. Common techniques for performance improvement, such as indexing, query optimization, and caching, should be revised and re-evaluated in the distributed heterogeneous environment.

The advanced processing of semistructured data, such as RDF or OWL, requires in particular similarity based quering of these data.

The focus of this research is on advanced caching techniques for similarity queries in distributed heterogeneous systems, call semantic caching. In contrast with usual caching techniques, in addition to the usage patterns, semantic caching takes into account relationships and inter dependencies between data items to improve cache effectiveness and enable local query processing of cached data instead of distributed query processing.

Our goal is to evaluate the potential impact of different approaches to semantic caching and obtain quantitative estimations of their suitability. Based on this analysis, we are developing new caching strategies based on relationships, inter dependencies, statistics of stored data and queries in a very broad context of distributed semistructured resources.

Caching is a technique used in nearly any software or hardware system trying to achieve high performance using hierarchical storage system. The examples are CPU cache, virtual memory, file system cache, database buffer pool, application server cache, browser cache, proxy server etc.

The idea of caching is to reduce the number of repeated accesses to the data items located in relatively slow large storage. Instead, data items are temporarily copied into fast (and relatively small) memory.

In almost all implementations cache is completely transparent for the application or system that uses the cached data. That is, the presence of cache affects performance, but not the functionality of the application, and, moreover, the application cannot explicitly control the behavior of the cache.

The data items copied from the primary (large and relatively slow) memory into cache

are called cache units.

The caching is called semantic caching if the cache units are meaningful items or sets in the high-level application data model. Thus, parts of cache unit may be used to process queries other than original one, or data from several cache units may be used together to evaluate new query.

The expectation is that the above mentioned should improve reuse of the cached data. On the other hand, the extraction of cached data becomes more complex and may result in certain performance penalties. Hence, our goal is to find conditions under which the semantic caching is benefitial.

In this research we study semantic caching for similarity-based queries, precisely knn- or range queries. For these classes of queries, the objects are included into the query result based on the value of certain metric or similarity measure. We consider our caching as semantic because

- the similarity function or metric function is supposed to capture certain semantics of the object relationships, and
- the cache units are single objects, rather than complete query results.

For complex structured queries it turns out that the major restricting factor for effective use of semantic caching is the query containment problem which is proven to be decidable only for limited class of queries and has polynomial complexity for much smaller class of queries.

In contrast, caching does not cause any additional computational complexity for similarity-based queries. Moreover, an approximate algorithms can be suitable for resolving query containment problem as the algorithms used at the primary data source are often also approximate, and, due to the nature of the similarity, completely exact result does not make sense.

For the reasons above, important characteristics to be investigated for similarity caching are performance improvement, relative precision, and relative recall.

# 1 Related Work

The notion of semantic caching was introduced in numerous works including [10, 14, 12, 2, 18, 6].

Franklin et al. in [10] proposed to use the queries that have been used to obtain the results in the cache as high level descriptors. Processing a new query, they first compare it to the previously asked queries and determine whether there are useful results in the cache. In our work the same concept of semantic caching is used. Different replacement policies are discussed and compared there. The performance of semantic caching compared to traditional page caching and tuple caching. The idea of remainder query is considered. They use some semantic knowledge about the data in cache in order to efficiently manage the cache. In contrast we exploit the semantic information in processing approximate queries.

According to [10] the entries in the semantic cache are organized by semantic regions. Semantic regions are defined dynamically based on answered queries. Each semantic region has a constraint formula (high-level description).

Semantic caching, a caching architecture for online data services and e-commerce applications, that caches the results of selection queries were considered in [13].

The semantic caching for XML Databases was considered in ([23],[24]). The XPath language to navigate XML documents and extract information from them were used there, XQuery language was used in ([15],[22],[19]).

The problem of partial query matching is studied in [15],[23],[22]. A semantic cache architecture is presented in [19] .

All the above is related to semantic caching of structured queries (relational or XML), while the focus of our research is on similarity caching.

RDF is the standard for encoding metadata and other knowledge on the Semantic Web, it's used in ([17], [21],[20]).

The semantic caching is based on the notion of similarity of RDF queries in ([20]), the problem of subsumption for RDF queries is discussed, a cost model is presented and a similarity measure for RDF queries is derived.

A mechanism for efficient caching of Web queries and the answers received from heterogeneous Web providers was used in ([8],[7]). Caching mechanism for conjunctive Web queries based on signature files was used in [5].

Algorithms to detect similarity between query predicates, efficient algorithms for proving containment among similar query predicates, a technique to dynamically aggregate similar queries in the cache to support efficient search are considered in [1]. Their concept of similarity differs from our based on distance function between query objects.

Caching mechanism for conjunctive queries based on signature files was used in [5] . The relation of semantic containment between a query and the corresponding cache items, the reminder query construction and replacement policies are discussed in terms of signature files. This idea is closed to exploiting distance function between feature vectors instead of objects themselves. But the idea of using signature files is very restrictive.

A cache hit is said to occur if the requested item is similar but not necessarily equal to some cached item in [16]. In our work we also consider the nearest neighbor queries and their caching. The similarity between items is defined by the special similarity function between item keys, which are similar to our feature vectors, and special utility function. Utility function is used in defining cache hits and replacement policy. Balls of similar items are constructed around one representative according to the similarity function.

The similarity caching problem in which a caching algorithm can return an element from the cache that is similar, but not necessarily identical, to the query element is introduced in [9] . They proceeded to quantify the hardness as a function of the complexity of the underlying metric space and showed that the problem becomes easier as we proceed from general metric spaces to those of bounded doubling dimension, and to Euclidean metrics. This idea is close to our decision to count distance function between feature vectors.

High-dimensional spaces -index structures for improving the performance of searching, different types of possible queries and algorithms for answering them are considered in [3]. Nearest neighbor query and range query are considered in our work and some proposed index structures should be exploited in it. We research the impact of caching and absolute performance of retrieval of cached data is not essential for our research. For this reason we do not analize indexing for cache model.

Current solutions for searching in metric spaces are discussed in [4]. Different distance functions and algorithms for several types of queries are in consideration.

Similarity search in metric spaces is considered in [11]. In order to compute the similarity between the cache content and the submitted query, their method needs to keep in cache not

only the identifiers of the objects returned by a cached query, but also the features associated with these objects. Object features are in fact needed to measure the metric distances between them and a new submitted query objects. Two different algorithms to manage the cache and answer queries in case of both exact and approximate hits are presented: QCache (Query Cache) and RCache (Result Cache), based on the counting of the maximum safe radius of query, which we can answer using cache only. In our work we implement these algorithms for different kinds of objects to conduct our experiments.

# 2 A Similarity-Based Caching

Instead of similarity measure we are using distance as it has an important property of triangle unequality.

Metric caching system exploits the metric property of the distance measure for the quality evaluation of the approximate results that the cache can guarantee.

Our system exploits the concept of distance between a query object and a collection of objects: similarity search can be seen as the process of retrieving from the collection the most relevant objects, where the ranking criterion is given in terms of the distance from the query object. The main constraint is that our caching system is considered for objects from metric spaces.

Our cache works for objects of different nature, for example RDF graphs and texts, pictures. The cache exploits distance function between feature vectors instead of objects themselves. The nature of the objects is transparent for cache; it is inside feature vectors and distance function between them. The construction of feature vectors and distance functions is out of the scope of this work.

## 2.1 The Cache Model

In content-based similarity search, even when the current query object was never seen in the past, the similarity among the asked query and the cache content can be exploited.

The cache can give the answer composed of the objects currently cached which are the closest to the current query.

The system is queried by to different kind of queries: range queries and k nearest neighbors queries. In a range query, a query point q, a distance r, and a metric d are specified. The result set comprises all points p from the database, which have a distance smaller or equal to r from q according to metric d. If a user wants a natural number k of closest points, he will perform a k-nearest neighbor query. The k-nearest neighbor query selects k points from the database such that no point among the remaining points in the database is closer to the query point than any of the selected points.

## 2.2 The Metric Spaces and Approximate Queries

We assume that in our similarity-based search application, objects are transformed into feature vectors of a vector space with fixed, finite dimension d. Therefore, a collection of objects is a set of points in a d-dimensional data space.

Let Data Set be a collection of objects belonging to the universe of the valid objects U and let d be a metric distance function, $d : U * U => R$, used to measure the similarity between two objects. (U,d) corresponds to a metric space, which is the basic assumption of our work.

## 2.3 Using a Cache to Answer Approximate Queries

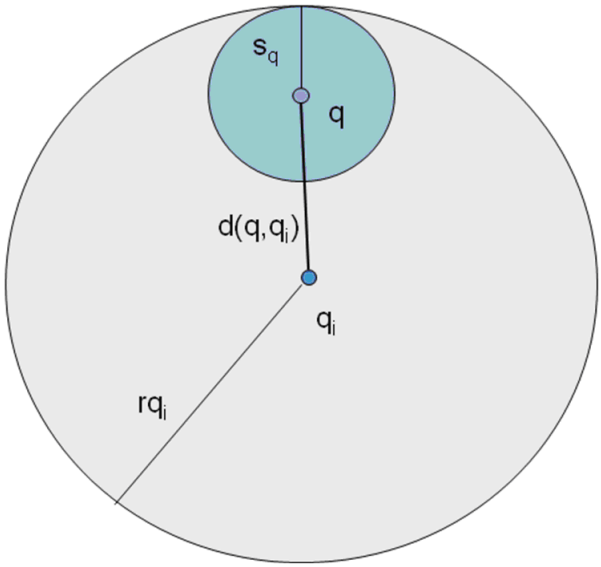


Figure 1: The safe radius

Let q be a new query. If q is in cache, then the results are known and they can be immediately returned. Conversely, it is interesting to understand whether the cached objects could be used to return some approximate results, and if so, it is important to provide some a-priori measure for the quality of such results. For the latter we use $s_q$ the safe radius of the new query q w.r.t. the cached query $q_i$

$sq(q_i) = r_{q_i} - d(q, q_i)$

We can find $k_1 <= k$ nearest objects to the new query q.

The cache give the approximate answers because the approximate hits occures only if all k nearest neighbors ($k_1 = k$) were found in cache. But it may return not the same k nearest neighbors as the primary data source.

# 3 The Experimental Environment

This section describes the target architecture and how cache can be used to improve the performance, the goals of the study and experimental environment.

## 3.1 The Architecture of System with the Cache

In our work we start from a system without cache which consists of the Client (application or system that uses the data) and the Primary Data Set. The Data Set gets queries from the Client and finds similar objects to the query.

The architecture of search system in our work consists of Client, Data Set (Primary Data Storage) and Cache. The major architectural unit considered in our research is the Cache. The Cache has the same interface as the Data Set and stores the recently submitted queries and associated results. The figure 2 shows the work of such system. The Client sends the query to the Cache. When processing a new query, the Cache compares it to the previously processed queries and determines whether there are useful results in the cache. If there is no necessary information in the Cache it redirects the query to the Data Set.

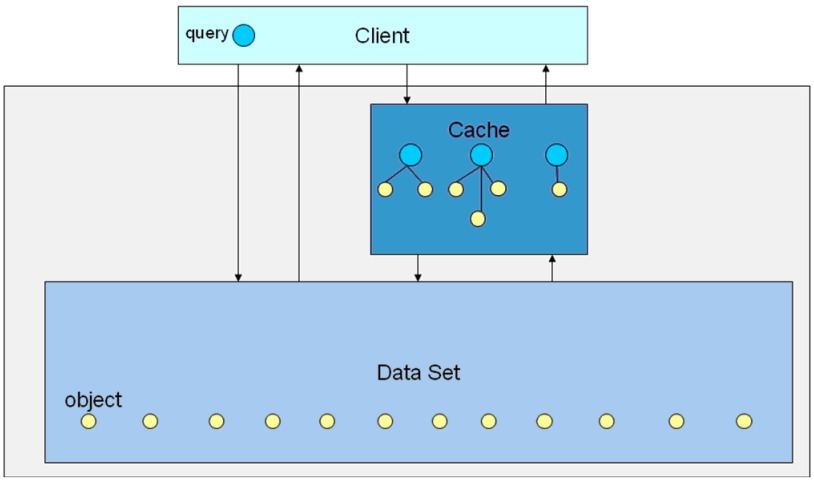


Figure 2: Cache architecture

Both the Data Set and the Cache store objects from metric space with some characteristic features associated with them which are needed to measure the metric distances. The distance function is completely transparent for the client. It is defined and used at the primary data source and must be known to the caching engine in our model.

There are some requirements to the Cache. It must be able to compute the distance functions between the feature vectors. It takes some information about the objects from the Data Set. The Cache needs to store feature vectors of the objects to be able to compute distance function between them and queries from the Client. The Cache can take objects themselves from the Data Set and store them but it does not use them for search.

## 3.2 Updates and Replacement

Processing of the updates of data located at the primary data storage is critical for effectiveness of any cache. Indeed, the cache can be completely transparent only if it contains exactly same data as the primary storage. Consequently, the updates of the primary data store must be propagated to the cache, and, if an application updated the cache, these updates must be propagated to the primary data store.

Technically the problem of cache synchronization can be splitted into two almost independent problems:

- Update awareness
- Cache replacement.

That is, the information on the updates of the primary data storage should somehow reach the caching engine. If the primary resource is using query-response model, then this information may be obtained only via polling the data storage periodically.

Alternatively, if the server is based on publish-subscribe model, the update notifications can be received and processed in the caching engine as the updated data are requested from the server when they are actually updated.

As soon as the updates are received, the obsolete data must be removed from the cache. There is no need to download new values as this will happen when these data will be requested by the application next time. In other words, lazy approach might be good here. We do not elaborate this topic futher in this report.

For semantic caching the engine must decide which cache units are affected by the update and hence must be removed. Further, data items that do not belong to the remaining cache units can be deleted from the cache. The latter is exactly same problem as incremental cache replacement. Although this problem is not computationally hard, it might require

significant amount of query evaluations. For this reason, most of proposed semantic cache implementations do not do any incremental updates of the cache. Instead, they simply clean all data from the cache when it is overflowing. We use the same strategy for this project.

## 3.3 The Goals

The primary goal of the research is to evaluate the usefulness of similarity caching in the context described above. The benefits from use of this kind of caching are expected to be limited, because the performance of similarity cache, most likely, will be affected by the 'course of dimensionality' and hence the results of experiments on real data sets are hard to predict. Hence the sub-goal is to identify usage patterns where semantic caching can be beneficial.

The main objective of work is to compare and evaluate different metric cache algorithms for search system and measure its productivity through the number of parameters. The consideration of the average number of approximate hits will show the advantages of semantic caching. The evaluation of results quality by the relative error on the sum of distances will present the usefulness of metric cache in search systems. The dependence of these parameters of the Cache performance on the kind of object, cache size and the number of objects in result set is to be analyzed.

The quantitative metrics to be experimentally measured are the following:

**Cache hits percentage** shows how many of incoming queries we answered from cache.

**Relative error** shows the average relative error of the answer received from the cache.

In this work we will first of all consider the approximate cache hits, but our caching system also find the exact hits. The reason is that exact caching systems are considered in other works and the average number of exact hits depends first of all on queries series and only then the caching algorithm.

All metrics listed above depend on the primary data storage size and cache size.

## 3.4 Experiments

In all our experiments the similarity metrics were calculated for texts or textual fragments extracted from semistructured data.

The environment includes two query processing engines. These engines are used to Primary data storage, as a cache storage.

There are two different algorithms to manage the cache and answer queries in case of both exact and approximate hits: QCache (Query Cache) and RCache (Result Cache), based on the counting of the maximum safe radius of query, which we can answer using the Cache only.

RCache (Result Cache) The cache is looked-up for the k objects that are the most similar to a given query object q. A hash table is used to store and retrieve efficiently queries and their results lists. RCache is also adopts a metric index that is used to perform the k nearest neighbors searches over the cached objects in order to return approximate answers when possible.

QCache (Query Cache) The idea is to search a set of kc suitable queries among the cached ones, and use their neighbors to produce an approximate answer. The hash table from the RCache is used. The implementation uses the metric index, which contains only the recent past queries to find the $k_c$ cached queries that are the closest to q [11].

## 3.5 The Data Sets

The similarity cache was used to run experiments on data for intelligent analytical applications based on publicly available financial reports represented in XBRL format.

There is actually also some syntetic data because in some reports we got text information from tables.

The distance function is derived from well-known cosine similarity measure, widely used in the information retrieval. The distance function constructed from this similarity measure does not meet the condition of the triangle unequality. This will bring some error to the approximate answer.

For each word in text element we count the ratio of the appearance in text element to the appaerance in all elements from the primary data storage. For each text element in the primary data storage we construct feature vector from this values. It is important to mention that our feature vectors do not reflect and contain information about the structure of financial reports.

Updates of the data are not monitored by the cache.

We restrict our experiments on the knn queries.

# 4 Experimental Results

For each combination of the primary data storage size and cache size a randomized series of the queries were run several times, and the average values were calculated.

We conduct all experiments for randomized number of the nearest neighbors less than 25 in knn queries and with cache size 0,25 of the primary data storage.

The figure 3 shows the average number of approximate hits for different size of the primary data storage.

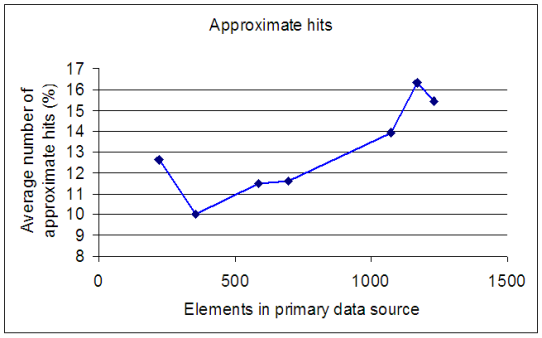


Figure 3: Approximate hits

Some picks on the graphic in our opinion depend on the distribution of the feature vectors in space. The growth of the average number of approximate hits can be seen.

The figure 4 shows the dependence of the dimension of the feature vector on the primary data storage size. The growth of dimensionality slows down as the number of documents grows because of the dictionary saturation. This fact affects the performance of caching algorithms.

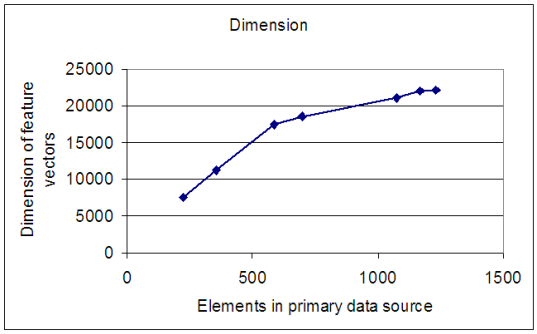


Figure 4: The dimension

As we use approximate cache in the figure 5 the relative error is considered. The error is small and the idea is that we can make our algorithm less precise.

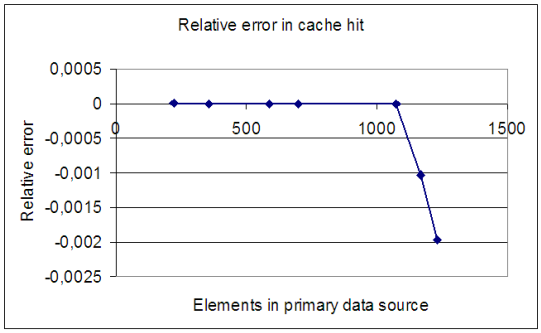


Figure 5: The cache hit error

The figure 6 shows the relative error in case of cache miss, to show that the relative error in the case of cache hit is small.

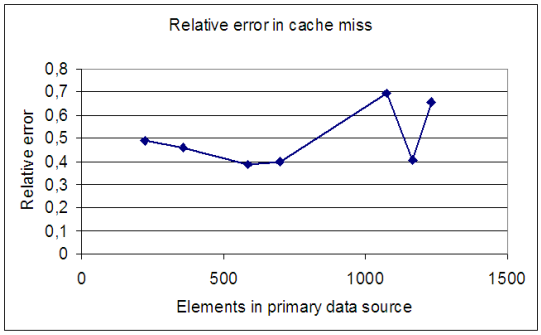


Figure 6: The cache miss error

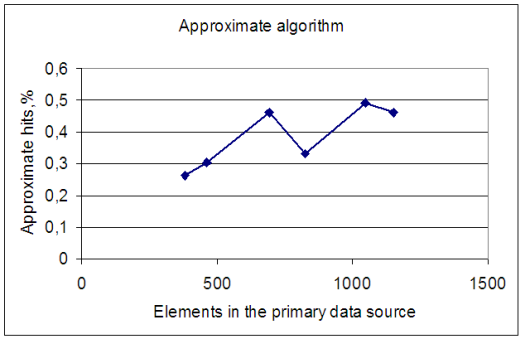


Figure 7: Approximate algorithm

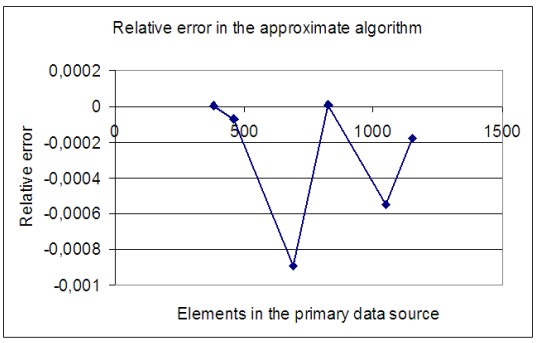


Figure 8: The approximate algorithm mistake

The difference between the average number of the approximate hits in case of exact algorithm and the approximate algorithm is shown in figure 7. The benefits are not huge, but this metric is to be considered in connection with the next.

On the figure 8 the difference between the relative errors in this cases is considered. The less decrease of precision gives the increase of the average number of approximate hits about 0,3 percent.

# Conclusions

In this research we presented an analysis of approach to similarity caching in metric spaces.

Starting from the knowledge gained from previous research, we developed a model of similarity caching engine and experimentally evaluated its performance. We implemented an efficient approximate algorithm for query containment problem.

Although quantitative metrics measured on our model meet, in general, our expectations. Of course, the results will heavily depend on the distribution of the objects in metric spaces, feature vectors and distance function.

We show that the variable approximate models of our algorithm may be considered with acceptable precision.

For future work we plan investigate the combined cache supproting both semistructured and structured queries.

# References


[1] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. Scalable template-based query containment checking for web semantic caches. In *In Proceedings of the IEEE International Conference on Data Engineering (ICDE*, pages 493–504. IEEE Computer Society, 2003.

[2] Naveen Ashish, Craig A. Knoblock, and Cyrus Shahabi. Intelligent caching for information mediators: A kr based approach. In *KRDB*, pages 3.1–3.7, 1998.

[3] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[4] Edgar Chávez, José L. Marroquín, and Gonzalo Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools Appl.*, 14(2):113–135, 2001.

[5] Boris Chidlovskii and Uwe M. Borghoff. Signature file methods for semantic query caching. In *In: Proc. of the 2nd European Conf. on Digital Libraries, LNCS 1513*, pages 479–498, 1998.

[6] Boris Chidlovskii and Uwe M. Borghoff. Semantic caching of web queries. *The VLDB Journal*, 9(1):2–17, 2000.

[7] M. Chidlovskii and C. Roncancio. Semantic cache mechanism for heterogeneous web querying, 1999.

[8] Boris Chidlovskiiy, Claudia Roncancioz, and Marie luise Schneidery. Optimizing web queries through semantic caching, 1999.

[9] Flavio Chierichetti, Ravi Kumar, and Sergei Vassilvitskii. Similarity caching. In Jan Paredaens and Jianwen Su, editors, *PODS*, pages 127–136. ACM, 2009.

[10] S. Dar, M. Franklin, B. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *VLDB '96: Proceedings of the 22th International Conference on Very*

*Large Data Bases*, pages 330–341, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[11] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. Caching algorithms for similarity search. In Valeria De Antonellis, Silvana Castano, Barbara Catania, and Giovanna Guerrini, editors, *SEBD*, pages 145–152. Edizioni Seneca, 2009.

[12] Parke Godfrey and Jarek Gryz. Answering queries by semantic caches. In *DEXA '99: Proceedings of the 10th International Conference on Database and Expert Systems Applications*, pages 485–498, London, UK, 1999. Springer-Verlag.

[13] Bjorn THor Jonsson, Maria Arinbjarnar, Bjarnsteinn THorsson, Michael J. Franklin, and Divesh Srivastava. Performance and overhead of semantic cache management. *ACM Trans. Interet Technol.*, 6(3):302–331, 2006.

[14] Arthur M. Keller and Julie Basu. A predicate-based caching scheme for client-server database architectures. *The VLDB Journal*, 5:35–47, 1996.

[15] V.Vaidehi M.R.Sumalatha and A.Kannan. Xml query processing semantic cache system, 2007.

[16] Sandeep Pandey, Andrei Z. Broder, Flavio Chierichetti, Vanja Josifovski, Ravi Kumar, and Sergei Vassilvitskii. Nearest-neighbor caching for content-match applications. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *WWW*, pages 441–450. ACM, 2009.

[17] Chris Mueller Preetha Lakshmi. Literature review, 2007.

[18] Kumar V Qun Ren, Dunham M.H. Semantic caching and query processing. *Knowledge and Data Engineering, IEEE Transactions*, 15:192–210, 2003.

[19] Heng Tao Shen, Jinbao Li, Minglu Li, Jun Ni, and Wei Wang, editors. *Caching Frequent XML Query Patterns*, volume 3842 of *Lecture Notes in Computer Science*. Springer, 2006.

[20] Heiner Stuckenschmidt. Similarity-based query caching. In *In Proceedings of the 6 th International Conference on Flexible Query Answering Systems*, 2004.

[21] Joshua Tauberer. What is rdf and what is it good for, 2005.

[22] M.Petropoulos V.Hristidis. Semantic caching of xml databases, 2002.

[23] Wanhong Xu. The framework of an xml semantic caching system. In *WebDB*, pages 127–132, 2005.

[24] Wanhong Xu and Z. Meral Ozsoyoglu. Rewriting xpath queries using materialized views. In *In VLDB*, pages 121–132, 2005.