

Semantic Cache Investment

Konrad G. Beiske, Jan Bjørndalen, Jon Olav Hauglid

konrad@beiske.org, jan@alumni.ntnu.no, joh@idi.ntnu.no

NTNU

Abstract

Semantic cache and distribution introduce new obstacles to how we use cache during query processing in databases. We have adapted a caching strategy called cache investment to work in a peer-to-peer database with semantic cache. Cache investment is a technique that influences the query optimizer without changing it. It suggests cache candidates based on knowledge about queries executed in the past. These queries are not only limited to the local site, but also detects locality in queries by looking at queries processed on remote sites. Our implementation of semantic cache investment for distributed databases shows a great performance improvement, especially with concurrent executions.

1 Introduction

Database management systems these days are expected to handle larger and larger volumes of data. This has renewed interest in distributed databases. The rising requirements for availability, scalability and recoverability, will eventually reach a point where a single computer is inadequate. Distributed databases is a candidate to meet this demand, but not before the challenges of the new environment have been dealt with. In a distributed environment, such as peer-to-peer (P2P), knowledge about other sites is rarely complete and data traffic is the new dominating factor in query execution. Measures must be taken to ensure that distributed queries are worthwhile.

Caching is often used in databases to increase the performance of queries. Semantic cache uses semantic descriptions to cache composite data such as intermediate results from previous queries. With proper utilization of semantic caching, data traffic can be reduced and unnecessary expensive operations avoided.

In this paper we present a solution for better cache utilization, semantic cache investment. Semantic cache investment is inspired by Kossman's cache investment [1], but applied in a P2P-based distributed system that has a semantic cache. It works in combination with the query optimizer to produce more valuable cache content. For this to be possible we introduce a new query matching technique for matching of semantic representations in a distributed environment. The next sections will describe related work, our solution, results and conclusion.

This paper was presented at the NIK-2009 conference; see <http://www.nik.no/>.

2 Related Work

Among related work we count the original cache investment [1]. Our work is based on the design described there, but differs in that our solution is for a semantic cache. In the original version the choice was between executing queries at the server or shipping the tables for execution at the client. Our version considers other peers as well, but the biggest difference lies in that a semantic cache has different notions of locality. Spatial locality in the data is replaced with semantic locality.

On query matching our work share similar ideas with solutions for query optimization with materialized views [2], and answering queries by semantic caches [3]. Common expression analysis [4] describes query tree comparison, using a representation very much like ours.

3 DASCOSA and Semantic Cache

DASCOSA is a peer-to-peer database. It is developed at the Norwegian University of Science and Technology (NTNU) to facilitate research into the field of distributed databases. Its architecture consists of autonomous independent sites connected loosely through a distributed hash table (DHT). Queries in DASCOSA are made into relational algebra trees called a query tree. In the query tree, the leaf nodes are table scans, and the inner nodes are algebra operations. Query execution plans are deployed top-down, and executed bottom-up. DASCOSA uses a cost-based algorithm to optimize the placement of these nodes. The optimizer is based on the well-known R* algorithm [5].

Semantic caching is used to cache intermediate query results in DASCOSA. Any node in the query tree could be cached and given a semantic description for the data. This semantic description allows us to substitute any node in a query tree with a cached node, if a match is found. The challenge here lies in finding the match in two very different query trees and see if a common sub-tree can be found, as given in Figure 1. For DASCOSA we have designed a representation called a Result set identifier (RSID). The RSID, while limited to PJS-queries, is a useful tool for doing quick query matching. PJS-queries are restricted to projects, joins and selections [4]. The RSID consists of sets with tables, constraints, and attributes. With query matching by RSID, we made the query optimizer able to plan for cache. Planning with cache is referred to as explicit cache. Implicit cache is a more basic technique where caches are not taking into consideration during planning, but instead only checked for match each time a site is executing a subquery.

Cache investment is an active caching technique, by taking the step further than just planning for the use of cache but also planning for what should be cached. The idea is to do analysis of previous queries and arrive at a suggestion for a profitable cache, which is given to the query optimizer. The optimizer is ultimately responsible for deciding if a cache hint will be used or not. It was originally designed for a distributed server/client architecture with data cache. Data cache is the traditional caching of data units as they are stored. When adapting cache investment to a P2P environment, we had to sacrifice the global system overview and turn to the distributed index to fill this role. The presence of semantic caching complicates the analysis of queries, because instead of just

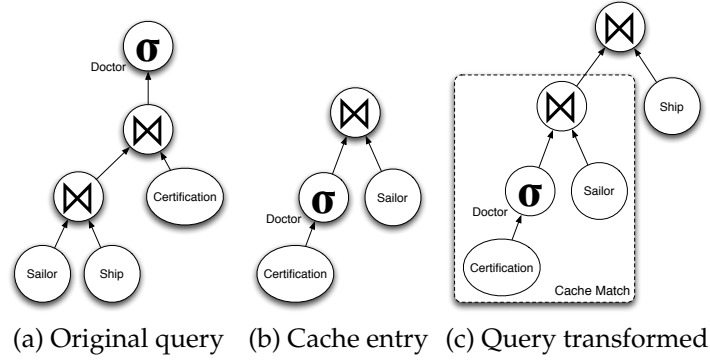


Figure 1: Example query match transformation.

tables or data blocks we now have semantically augmented data. Once again we used the RSID to identify a node in the query tree. After each executing node has completed, it is logged to the DHT. In the DHT it is routed to the indexing site of one of the tables used, which one is chosen by a hashing function.

The actual cache investment algorithm takes place at each indexing site receiving log entries for executed operations. On arrival, log entries are processed by adding them to the candidate set and increasing the value of cache candidates by the benefit they would have provided to the query, as given in Equation 1. Finally, all candidate values are decreased by multiplication with an aging factor between 0 and 1. Candidates with value less than a given threshold are removed in order to prevent the history from growing unbounded. In Equation 2, AvgReduction is the estimate for the reduction factor for queries executed at the site which is applied before data is shipped. The distance between sites are given by synthetic network coordinates¹.

$$Benefit = Max(PreviousCostAtSite - UseCost, 0) \quad (1)$$

$$UseCost = TupleCount * AvgReduction * Distance(Site, CandidateSite) \quad (2)$$

4 Results

We have evaluated our solution by measuring execution times for queries from the TPC-H [6] benchmark. We used three different workloads. Each workload consisted of a hot set and a cold set. Workloads 1-3 correspondingly had hot sets {1, 6}, {1, 13} and {11, 12} and cold sets {2, 3, 10, 11, 12, 13, 14, 15, 16, 17}, {2, 3, 6, 10, 11, 12, 14, 15, 16, 17} and {1, 2, 3, 6, 10, 13, 14, 15, 16, 17}.

In order to simulate different network topologies we used IPv6 tunnels from Hurricane Electric² that allowed us to select a remote proxy for each site.

All the results given in Figure 2a where obtained with a single site initiating the queries. There is a clear tendency that cache investment pays off and that the reward is greater with greater network distance. In order to test if cache investment is capable of exploiting locality among queries from different sites we also did a test with three sites concurrently executing a workload each. All the sites ran all the workloads, but no site had the same workload within the same

¹DASCOSA uses Pyxida for this service. <http://pyxida.sourceforge.net/>

²<http://www.tunnelbroker.net>

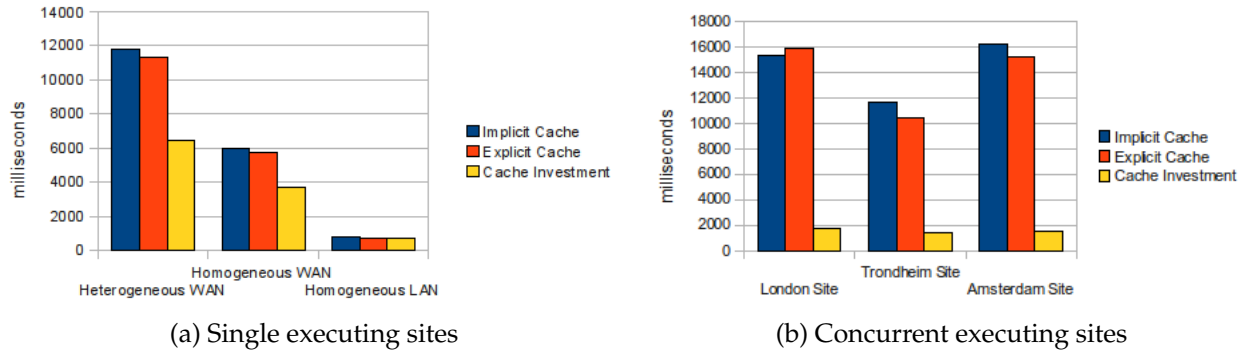


Figure 2: Execution times for different network configurations.

time frame. The results of this test are given in Figure 2b. Figure 2b shows that the concurrency increases the average execution time for all sites substantially, except for cache investment.

5 Conclusion

In this project we have designed an active caching strategy, called semantic cache investment, and implemented it for DASCOSA.

Evaluation has shown improvement in performance when using semantic cache investment. We have explored how network topology and concurrency affects caching in distributed databases. Results have shown that cache investment is effective. Cache investment is most effective in Wide-Area-Network systems with concurrent query execution. This shows that cache investment takes into account locality between both sites and queries.

References

- [1] Donald Kossmann, Michael J. Franklin, Gerhard Drasch, and Wig Ag. Cache investment: integrating query optimization and distributed data placement. *ACM Transactions on Database Systems*, 2000.
- [2] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of ICDE'1995*. 1995.
- [3] 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*. 1999.
- [4] Sheldon Finkelstein. Common expression analysis in database applications. In *SIGMOD '82: Proceedings of the 1982 ACM SIGMOD international conference on Management of data*. 1982.
- [5] Guy M. Lohman, C. Mohan, Laura M. Haas, Dean Daniels, Bruce G. Lindsay, Patricia G. Selinger, and Paul F. Wilms. Query processing in R*. In *Query Processing in Database Systems*. 1985.
- [6] Transaction Processing Performance Council (TPC). *TPC Benchmark H (Decision Support) Standard Specification*, 2.8.0 edition, 2009.