Agent Based Simulation to Evaluate Adaptive Caching in Distributed Databases

Santhilata Kuppili Venkata[1], Jeroen Keppens[1] and Katarzyna Musial[2]

[1] Department of Informatics, King's College London, London, UK
{santhilata.kuppili_venkata,jeroen.keppens}@kcl.ac.uk
[2] Faculty of Science and Technology, Bournemouth University, Poole, UK
kmusialgabrys@bournemouth.ac.uk

Abstract. Caching frequently used data is a common practice to improve query performance in database systems. But traditional algorithms used for cache management prove to be insufficient in distributed environment where groups of users require similar or related data from multiple databases. Repeated data transfers can become a bottleneck leading to long query response time and high resource utilization. Our work focuses on adaptive algorithms to decide on optimal grain of data to be cached and cache refreshment techniques to reduce data transfers. In this paper, we present agent based simulation to investigate and in consequence improve cache management in the distributed database environment. Dynamic grain size and decisions on cache refreshment are made as a result of coordination and interaction between agents. Initial results show better response time and higher data availability compared to traditional caching techniques.

Keywords: Cache management, Distributed databases, Agent based simulation

1 Introduction

Nowadays, large volumes of data are inseparably connected with scientific and commercial applications. Common query interface provides uniform access and allows a client to interact with multiple data stores seamlessly. Often user queries tend to get repeated when groups of users working on related projects, send their queries to multiple databases. Repeated queries need same data to be retrieved and processed several times causing repeated data transfers, high bandwidth utilisation and thus delayed responses.

The main focus of our research is to investigate the effectiveness of adaptive caching with sub-query fragmentation technique [1]. This work aims to reduce average query response time and reduction in data transfers. Adaptive caching in distributed cache system works on the aggregated information across groups of users related by a specified work-flow and need information at various stages of their work repeatedly from different locations. It is observed that when users are distributed, often their queries are not repeated fully but overlap only partially.

We utilise this feature to develop sub-query data caching based on the query information collected from multiple groups of users. Cache collects data needed across users and adapts itself to common data usage patterns.

Adaptive caching works by collaborating the knowledge gathered by independent cache units in the system. We have developed a query analysis tool (QA tool) to support distributed query decomposition in the distributed database environment. Each cache unit supply information about the query origination and usage of data locally. QA tool then analyzes the information to find associations between queries and finds patterns. QA tool predicts future requirements and takes a decision about the best way to cache data across multiple cache units.

It was expensive to obtain a dedicated real life distributed system to evaluate our cache techniques and their suitability. Adaptive cache system requires independent autonomous cache units to collect data about queries and forward information about their current and predicted needs. Hence the nature of this application makes an ideal model for agent based simulation. Our simulation has active agents such as users, cache units, query analyzer(s) that work together and coordinate their actions with static agents such as databases. This paper mainly is focused on two goals (i) reduction in response time, (ii) efficient cached data management as a result of cache grain modification and cache refreshment.

This paper is organised as follows: A brief context of caching in databases, adaptive caching is given in section 2. A detailed application of the multi-agent system and implementation of agents for the successful cache maintenance is explained in Section 3. Evaluation of simulation and analysis of the results are presented in Section 4.

2 Background

A general background about caching in distributed database environment and a brief introduction to sub-query fragmentation is explained in this section.

Caching in Databases: Data caching is used to improve the performance of the database management system to achieve reduced query response time and thus lessen the burden of processing resources. The effectiveness of a cache depends on three important cache management techniques: *cache granularity, cache refreshment and cache coherence.* In client-server systems cache refreshment uses time based algorithms such as LRU (Least Recently Used), MRU (Most Recently Used), frequency based algorithms e.g. LFU (Least Frequently Used) or the combination of both [2]. Cache effectiveness is measured in terms of number of cache hits.

A distributed database environment consists of data distributed over multiple data stores across the globe. Distributed caching systems also need to consider the resource utilization such as network bandwidth, processing time at data servers and the heterogeneity of the data from multiple databases. Usually user queries are fragmented into smaller segments according to the data source from where the data can be retrieved to achieve query optimization [3]. Proxy caches are installed to improve optimization of network and processor resource utilization. General practice in the distributed caching is to cache all the data that comes from a single source and reuse for the future queries [5].

The usual grain of cache is a page/table/attribute in applications that query relational databases [2]. In web applications, queries are sent to retrieve information from text documents that reside in web servers. Hence, the grain for cache is often independent data item such as a frequently visited web page, an image or a multimedia item [6]. Extending any of these caching methods to distributed databases is difficult, as applications need to integrate data from multiple databases. Large data transfers, work loads at servers, network limitations and limited cache storage size are the general issues to consider while designing a cache [4].

Sub-query Fragmentation: A sub-query within a query is the smallest independent thread of execution generated as a part of the overall query plan [1]. For example, a *join* between tables or a standalone nested query is a sub-query. other words, a sub-query (q_k) of a query ($\mathcal{Q}$) is defined as the atomic query segment that stores an independent data block such that $\mathcal{Q} = q_1 \cup q_2 \cup .. \cup q_n \Rightarrow \bigcup_{k=1}^{n} q_k$. Here '$\cup$' represents aggregation such as *join* between two tables.

Adaptive cache learns data access patterns and finds the longest sub-query of common interest. Sub-query caching tool aids the data localization phase in distributed query processing. We define *cache grain* as the longest sequence of sub-queries accessed frequently together. Initially, a grain may be of the size equivalent to the smallest query segment recognized by the query optimizer. But the grain size can be refined depending on the user queries. Hence it is possible to store bigger patterns (sequences of sub-queries) as a whole as a single grain in the cache. Similarly, infrequent sub-queries are removed from the grain.

3 Multi-Agent System for Adaptive Caching

Multi-Agent system (MAS) modelling is a widely used approach to solve complex learning, planning, and decision making problems in distributed systems. Autonomous processing nodes (agents) contribute, communicate and coordinate to achieve common goals. For e.g, multi-agent system models developed for distributed health, power [7–9]. To develop a multi-agent system for adaptive caching, we follow an approach consisting of a flexible and generic MAS architecture that can use decision making (with the help of machine learning) and information gathering techniques. In our system, we have identified four main types of agents as shown in Fig. 1. Each of these agents is defined with one or many attributes (A) from the tuple: Object O, States S, Communication C, Domain knowledge K, Heuristics H.

$$\text{Hence, } A = < O, S,\ C,\ K,\ H >$$

User Agents (UA) are the instigators of the querying process. Main responsibility for a user agent is to monitor the query response time.

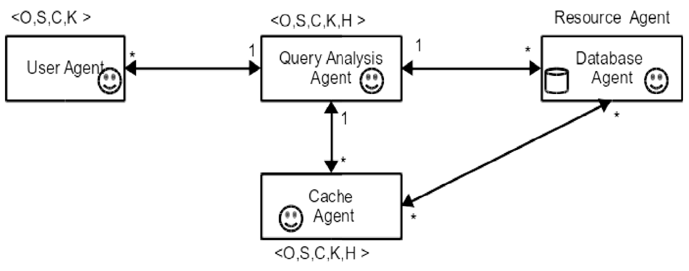


Fig. 1. Agent architecture with related agents

User Agent	A ⇒ < *O*, *S*, *C*, *K* >

O= the query,
S ={send, suspend, wait},
C= Communication with Query Analysis Agent (QAA),
K = Domain knowledge
	1. Coordinate with global clock time
	2. Set the query start time S_t
	3. t = send(Query, QAA)
	4. Set receive time (R) = t
	5. Response time = R-S_t

Cache Agents (CA) manage the query pattern store at local level. They have the responsibility to share knowledge with QAA by recommending optimal place for the cached data unit.

Cache Agent	A ⇒ < *O*, *S*, *C*, *K*, *H* >
O = set of sub-queries, S = {search, acquire, cluster, contribute }, C = {Contribute to QAA, acquire from Database agent}, *K* = Local knowledge about patterns, *H* = set of association rules to modify cache grain	1. Receive request for a sub-query 2. search & update the frequency for sub-query 3. Apply association rules 4. Contribute to QAA knowledge base 5. If data not available, acquire data from database

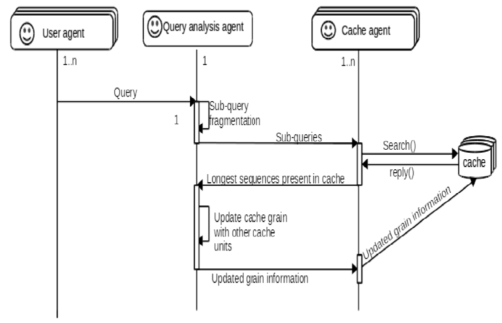
Query Analysis Agent (QAA) is a central coordinating agent at the highest level to implement the decision making layer in the system. It plans the execution module and periodically gathers queried data patterns from all cache agents and user agents. QAA then consolidates information to perform cache refreshment.
Database agents (DA) are resource (static) agents. They understand database load characteristics of the data usage and periodically submits this information to QAA. For this paper, we have not implemented any functionality for this agent.

3.1 Task1: Cache Grain Modification

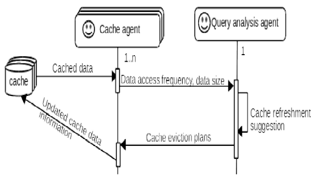
is a part of periodical cache management. Deciding an optimal grain is achieved by sequence of interactions between user agents, query analysis agent and cache

Query Analysis Agent	`A` $\Rightarrow$ $<$ *O*, *S*, *C*, *K*, *H* $>$

`O = set of sub-queries,`
`S = {send, update, maintain },`
`C = {communicate to UA, CA and Database agent},`
K `= Global knowledge (query patterns),`
H `= association rules to modify cache grain, cache data mobility`
	`1. Receive request for a query`
	`2. fragment and send query to CA`
	`3. send or receive data requests to databases`
	`4. Update knowledge base`


User agent
Query analysis agent
Cache agent
1..n
1
1..n
Query
1
Sub-query fragmentation
Sub-queries
Search()
reply()
cache
Longest sequences present in cache
Update cache grain with other cache units
Updated grain information
Updated grain information

(a) Communication between agents to update cache grain using sequence diagram


Cache agent
Query analysis agent
1..n
1
cache
Cached data
Data access frequency, data size
Cache refreshment suggestion
Updated cache data information
Cache eviction plans

(b) Communication between agents for cache refreshment using sequence diagram

Fig. 2. Communication between agents to perform adaptive caching activities

agents. From time to time, query analysis agent actively collects data access patterns across all groups of users (user agents) and then decides on the longest sequences that are queried frequently. Similarly, a grain is shrunk when a containing sub-query is accessed less frequently. Sequence diagram to achieve optimal grain is shown in Fig. 2(a).

3.2 Task2: Cache Refreshment

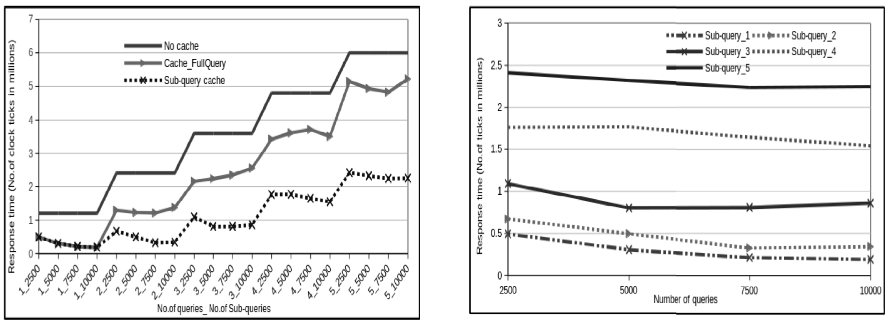
a.k.a. cache eviction. Periodically, the query analysis agent collects data access frequency and the cached data size from cache agents. Less needed queries are removed from the cache. Owing to the distributed nature of caching we propose two distributed caching algorithms: (i) Distributed Least Recently Used (DLRU), (ii) Distributed Least Frequently Used (DLFU). Both these algorithms are based on the metadata about each cached grain indexed in the query index for the current period of time and it's historical information. A decision is made for each of the cached grain to store, delete or relocate from the current location to new location dynamically by collecting information from distributed cache units. Typical interactions between query analysis agent and cache agents is shown in Fig. 2(b).

4 Evaluation

In order to evaluate the efficiency of decisions taken by agents, we have implemented a discrete time step agent simulator using Java (JDK-1.7) programming language. A centralized common time thread (global clock) was implemented to run in an infinite loop of time steps that forwards itself by one tick with every iteration of the loop. Network elements, database servers, cache servers, and users were defined with a unit of work to be completed within one clock tick.

Experimental Setup: We generated input query traces using TPC-H benchmark[3] composite queries (Query number: 5, 7, 10). In order to measure and compare, we have generated synthetic workloads using known statistical distributions. Each experiment was repeated with identical query traces for multiple number of times. We made following assumptions: (i) maximum data size for each query is fixed (since our aim is to estimate the percentage of data transfer reductions, this assumption would not hamper any observation); (ii) transmission networks are congestion free (hence the data transfer delay consists of only transmission time over the network); and (iii) all tabulations show time in terms of number of ticks elapsed by the global clock. Goals to be achieved by the tasks above was divided into two distinct cases.

Case 1: Observation of Average Query Response Time Comparison of average response time between different caching strategies is plotted in Fig. 3(a). We have compared average response time when (i) no cache is used, (ii) cache that stores only full query results as a whole is used, and (iii) cache with sub-query caching with grain modification is used.



(a) Average response time comparison for different cache techniques

(b) Average response time for queries varying complexity

Fig. 3. Evaluation using average response time

To investigate the effectiveness of sub-query fragmentation for partially repeated data, we created query trace where sub-queries within the queries repeat in 40%

[3] http://www.tpc.org/tpch/

of cases. For a standard time window of cache refreshment, it is observed that sub-query caching with modified grain has considerably lower response times compared to the response time with no cache or cache that stores only full query results. Sub-query caching seems to be better as the complexity of queries increase. (The label “3_2500” on x-axis represents a query trace of 2,500 queries with each of these queries having three sub-queries). Fig. 3(b) compares average query response time with sub-query caching with growing complexity. This figure compares two caching techniques: (i) full query result caching and (ii) sub-query caching for queries with two *joins* (2 sub-queries) and three *joins* (3 sub-queries).

Case 2: Observation of Average Data Transfers

The volume of data needed for a query is proportional to the amount of resources required. Fig. 4 compares volume of data found within cache when user queries were repeated only partially.

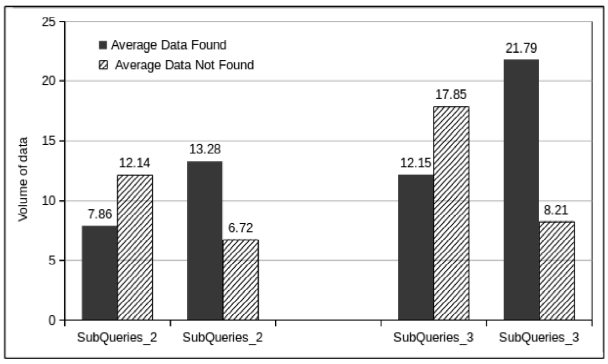


Fig. 4. Comparison of average data found in cache for full query search and sub-query search techniques

Test input consists of four traces of 10,000 queries each with varying query complexity. The first two sets of columns on the left-hand side represent the volume of data found (in black) in the cache and volume of data needed (in checkered) to be brought from remote databases. Volume of data found within the cache and volume of data to be brought for full query caching is shown on the left hand side and sub-query caching on the right. Average data size found using sub-query caching technique is found 1.6 to 1.8 times more than the full query cache, suggesting less resource utilization and hence reduction in data transfers.

5 Conclusion and Future work

This work is a part of research on mobile adaptive caching for distributed databases when query load consists of partially repeated queries from groups of users. Adaptive caching is aimed at resource optimization by coordinating needs from users in the distributed environment. We have developed sub-query caching technique to be able to maximize the benefit of cached data, using sub-queries as cache grains. In this paper, we presented the evaluation of adaptive caching in comparison with full query caching using agent based simulation. As of now, we have checked the potency of our strategy for read-only queries. We need to extend this work for concurrent read-write queries.Also we intend to develop exchange of cached data with other cache units with the help of demand assessing mobile agents using sub-queries in future.

References

1. Kuppili Venkata, S., Keppens, J., Musial, K.: Adaptive Caching Using Sub-query Fragmentation for Reduction in Data Transfers from Distributed Databases, ADASS XXV, ASP Conf, Ser., vol. TBD, pp. TBD (2016)
2. Elmasri, R., Navathe, S., Fundamentals of Database Systems (6th ed.). Addison-Wesley Publishing Company, USA (2010).
3. Ozsu, M. T., Principles of Distributed Database Systems (3rd ed.). Prentice Hall Press, Upper Saddle River, USA (2007).
4. Silberschatz, A., Korth, H.F., Sudarshan, S., Database System concepts (6th ed.). McGraw-Hill (2010)
5. Wang, X., Malik, T., Burns, R. C., Papadomanolakis, S., Ailamaki, A. : A Workload-Driven Unit of Cache Replacement for Mid-Tier Database Caching., In : DASFAA, K. Ramamohanarao, P. R. Krishna, M. K. Mohania, E. Nantajeewarawat (eds.), pp.374–385, (2007)
6. Zhu, H., Yang, T.: Class-based Cache Management for Dynamic Web Content., in: INFOCOM , IEEE, pp.1215-1224 (2001)
7. Mahmoud, S., Tyson, G., Miles, S., Taweel, A., Staa, A.,Tjeerd, V., Luck, M., and Delaney, B.: Multi-agent system for recruiting patients for clinical trials, In: proceedings of International conference on AAMAS '14, France, pp.5-9 (2014)
8. Chuan-Jun,S., and Chia-Ying, W. : JADE Implemented Mobile Multi-agent Based, Distributed Information Platform for Pervasive Health Care Monitoring., in: Appl. Soft Comput. pp.315-325 (2011)
9. Zhong, Z., McCalley, J.D., Vishwanathan, V., Honavar, V. : Multiagent system solutions for distributed computing, communications, and data integration needs in the power industry, In: Power Engineering Society General Meeting, 2004. IEEE (2004)