Improving the Performance of SQL Join Operation in the Distributed Enterprise Information System by Caching

Weiwen Yang
Columbia University, Wy49@columbia.edu

Yanzhen Qu
Colorado Technical University, yqu@coloradotech.edu

Improving the Performance of SQL Join Operation in the Distributed Enterprise Information System by Caching

Weiwen Yang
Columbia University
Wy49@columbia.edu

Yanzhen Qu
Colorado Technical University
yqu@coloradotech.edu

ABSTRACT

The enterprise information system (EIS) contains databases and other data sources in multiple data centers. Users query the EIS via clients. The client has a working space in the cloud. Caching data in client space will reduce the total execution time of the query. However, the client space has limited resources to store data. There are two options for caching data at the client space: caching the final results of query operations, or caching the source data tables. The problem is that some query operations such as “joining multiple big tables” will simply produce a result too big to store in cache in some cases. By contrast, caching source data tables may be a better choice in those situations. This paper presents an algorithm that combines active caching and passive caching to improve the cache hit, thus improving performance of the SQL join query in the cloud computing environment.

KEYWORDS

Distributed database, cloud computing, passive caching, information system, active caching, SQL join.

INTRODUCTION

SQL query transactions are frequently used in the local databases. Query transactions’ executions are normally very fast, as the source data tables are in local databases. Conventionally, database cache granularity is usually at the database level, table level and the result set level (Altinel, Bornhövd, Krishnamurthy, Mohan, Pirahesh and Reinwald, 2003). The SQL query execution may generate a result set. However, in a cloud computing environment, data may be stored in multiple remote datacenters. The distributed database system is huge and usually not in the same host as the client application. The client may be far away from the database servers in cloud computing environments. The users query the remote databases through clients. The client has a working space in the cloud, which is the intermediate space between the user and the enterprise information systems of the data centers. The cross datacenter join operation is slow for large tables. Because the data distributed in different datacenters must be brought to the client working space to execute the query and the result will be sent to the user, it will take time to transfer the data from different data centers to the client working space. Certain types of query transactions, such as various types of “join operations”, can be extremely slow due to data size and the limited network bandwidth. Caching data in a client’s working space will reduce the total execution time of the query, especially when the data involved in query transactions may be used multiple times. Still, the client space usually has limited cache memory and can store only a limited amount of data. Therefore, quite often, a decision has to be made on what should be cached: Should it be the result of a query, or the source data tables of a query? Regardless, a query transaction such as “joining multiple big tables” will simply produce a result too big to store in cache in some cases. With that being the case, it may be wiser to cache only the source data tables. In other circumstances, the opposite may be true. The SQL join operations include Left Outer Join, Right Outer Join, Full Outer Join, Inner Join, and Cartesian Product. *Join* is one of the most expensive SQL query operations. Improvement of the performance of SQL join operations is crucial.

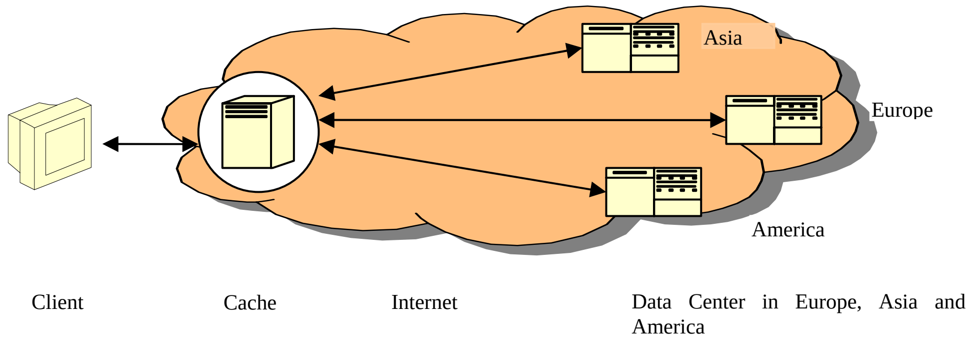

Figure 1. Caching.

Figure 1 shows the caching mechanism when querying data from the remote data center. When the user submits a query to the client, it returns the result if the data is found and the time is not expired in the cache. Otherwise, it connects to the Internet, fetches data from the data center, and then computes the result in the client working space.

The rest of this paper first presents a short literature review on the subject in discussion. Next is the problem statement and the methodology to address the problem. Following that is the simulation experiment result. The last section presents the conclusion and brief thoughts on future works.

RELATED WORK

Caching in databases has been widely studied in the past. The types of cache include CPU, disk, web, database, search engine, and distributed system cache. Memory close to the central processing unit works faster than the main memory. There may be several levels of cache to the CPU; data are brought into memory from disk when an application executes. The data in memory is stored in blocks or pages. The memory can be replaced by some policies such as the least recently used algorithm (LRU). The disk controller may need the cache to speed up processing data.

Web servers and web browsers use caches to process the requests faster. Luo, Naughton and Xue (2008) conducted research on form-based proxy caching algorithm for improving the web queries. Rodriguez, Spanner and Biersack (2001) analyzed the web-caching architecture, evaluated the hybrid scheme, and determined the optimal number caches in each caching level to reduce latency. Doulkeridis, Zafeiris and Vazirgiannis (2005) presented a method for caching and evaluated the benefits of context-aware service discovery. One researcher studied the filter effects in web caching hierarchies of multiple-level caches. Requests can be forwarded to the next level caching only if the first-level caching missed (Williamson, 2002). The research of Fagni, Perego and Silverstri (2006) involved improving the performance of web search engines via caching and pre-fetching query results from historical usage data. A middleware system was implemented to add dynamic content to the back end servers (Bouchenak, Cox, Dropsho, Mittal and Zwaenepoel, 2006), and redundant format style and layout for web pages can be removed for web browsers (Zhang, Wang, Pan, and Zhu, 2010).

In distributed systems, data can be cached in different places, clients, or servers to process the queries faster; a search engine stores the web page links in cache and improves performance in answering search queries. Baeza-Yates, Gionis, Junqueira, Murdock, Plachouras and Silvestri (2008) researched the trade-offs in designing efficient caching systems for web search engines, and studied the problem of finding the optimal way to split the static cache between answers and posting lists. An algorithm was developed for weighted result caching for web search engines (Gan and Suel, 2009), and the static caching of posting lists for search engines was proposed (Baeza-Yates, Gionis and Junqueira, 2007).

Database servers and clients use the cache to respond to queries quickly. Caches can greatly improve the overall performance of storage systems. They are widely used in file systems (Nelson, Welch and Ousterhout, 1998) and relational database systems (Chou, and Dewitt, 1985). Other researchers proposed the query cost into the cache policy (Ozcan, Altingovde and Ulusoy, 2011). Value-based cache was proposed for low-bandwidth clients (caching substring rather than whole files) (Irmak and Suel, 2005). The method of computing the result from the old input and the difference was proposed (Liu, Stoller and Teitelbaum, 1995). *Similarity caching* was proposed to improve the advertising system (Pandey, Broder, Josifovski, Kumar, Chierichetti and Vassilvitskii, 2009). Chierichetti, Kumar and Vassilvitskii (2009) introduced a new algorithm for retrieving similar data in the cache. Houle, Oria and Qasim (2010) presented the active caching method for deriving a query result from the existing cache data. Careful integration of file caching, pre-fetching, and disk scheduling can improve file system performance (Cao and Felten, 1996). Improving hash-join operations by two techniques (group pre-fetching and software-pipelined pre-fetching) was studied (Chen, Ailamaki, Gibbons and Mowry, 2007). A schema of the data cache management system was studied to improve the data-intensive operation in the grid for structured queries. A knowledge base was used to determine what data to store locally (Ahmed, Zaheer and Qadir, 2005). Computer researchers also investigated the performance result of simple hash-join operations in the main memory for multi-core systems. That technique is especially effective for skewed distributed data (Blanas, Li and Patel, 2011). Multi-thread hash-join operations with all data in the main memory were studied, noting shared critical structure at the cache level and evenly distributed load among threads. Simultaneous multithreading will significantly speed up the hash-join operations compared to the single-threaded operations (Rashid, Hassanein and Hammad, 2008). A framework was proposed to exploit the known statistics of input streams to make cache replacement decisions based on maximizing the expected number of result records. The framework partitions a single relation into multiple relations to improve performance (Xie, Yang and Chen, 2005). The PAX storage model significantly reduced query execution time for sequential file scans because the local data availability was improved greatly. Morphing uses a page structure and provides a more general storage model to decompose the records into groups of attributes (Hankins and Patel, 2003). The multi-tier enterprise architecture may include a database server, application server, web server, and load balance server. The cache table was defined and populated in query execution time. The cache tables, query execution plans, cache constraints, and asynchronous cache population methods were developed to improve query performance (Altinel et al., 2003).

All the previous research works regarding the caching mentioned above seldom touch upon the subject of how to efficiently execute the SQL join operations of the query transaction in the cloud computing environment.

PROBLEM STATEMENT

In a cloud-based enterprise information system, the data from different datacenters need to be brought to the client space to perform SQL join operations. Each client has a working space in the cloud. The network bandwidth and the table size can greatly affect query performance. It is critical to improve the SQL join operations on the client side.

METHDOLOGY

To solve the problem as stated in the problem statement, the authors herein present an algorithm which combines both the active caching and passive caching. The existing caching includes passive and active caching. The passive caching stores the previously used data. The active caching retrieves source data into client space based on predicted usage behavior. Each type of caching has its own advantages and disadvantages. The passive caching will increase the cache hit ratio if the previous queries are repeated; otherwise, the passive caching will not increase the cache hit ratio. The active caching will increase the hit ratio if the pre-fetched data matches the query; if not, the active caching will not affect the cache hit ratio. The details are as follows.

Decision-making Formula

The passive caching stores the previously used data with *time to live period*. That means the data exist only within a limited period in the passive caching layer. The active caching layer pre-fetches the source data into the client space based on historic usage once in a while. When the client receives the query, it first returns the valid result if it is in the cache and not expired, or computes the result if the valid source tables are in the cache. The client will bring data from the remote data center to compute the result if the source tables are not in the cache or expired. Combining the above passive and active caching can improve performance. Experiments demonstrating this approach improved the caching hit rate and thus improved query performance.

- Determine the *n* most frequently used tables of the last *x* days. Find the related tables based on the usage and the nearest neighbors in the Entity Relationship Diagrams (ERD) graph. If table A has a foreign key that refers to table B, there is an edge from A to B. Tables A and B are considered vertices. The weight of an edge is 1. Given a distance *d* and the name of the table, compute all related tables within distance *d*. All the related tables are pre-fetched into the cache once in a while to maintain its freshness, a process called *active caching*. The *n* most frequently used table in the last *x* days cannot exceed the expected disk limit, which is $\sum_{i=1}^{n} T_i < Limit$, where T is *i-th* table size.
- The query operations of the last *m* hours are cached based on the result and the size of the source tables. If the joined result size is at least K times that of the source table size, then the source tables are cached. Otherwise, the result is cached. The condition to cache the source table is $R \div (\sum_{i=1}^{n} t_i) \geq K$, where R is the joined result size, i is an integer, *n* is the number of source tables, t_i is the *i*-th table size, and K is the predefined integer. This process is *passive caching*.
- Combining the above active caching and passive caching will improve the performance of SQL join operations.

Building the Table Graph

The nodes are labeled from 1 to *n*, where *n* is the number of nodes. Each item of the list is an object containing all its neighbors. For example, node A has neighbors B and C, and the edges are AB and AC. Each element of the list corresponds to a node in the graph. If table A has a column reference to table B, then there is an edge from A to B. The weight of the edge can be considered as 1. The adjacency list representation of the graph can be used to represent a directed and undirected graph. The space complexity of the graph is O(|V|+|E|), where |V| is the number of nodes and |E| is the number of edges.

Computing the Related Tables

The top *k* most frequently used tables will be used to calculate the related tables in the cache of the last *m* days. For any outer join and inner join operation, one table must have reference to the other table on the join columns. The following algorithm computes the related tables with a distance of less than *d*.

The input is a graph, a source node, and the distance *d*. First the distance of each node is initialized to an infinite value. Each neighbor node of the source node is added to queue Q. When Q is not empty, pick the last node *u* from Q. If the distance of *u* is less than *d*, then add to the output list. If the distance of *u* is less than *d*, explore each neighbor *v* of *u*. If the distance[u] + path(u,v) < distance[v], then set distance[v] = distance[u] + path(u,v). Exit the while loop if Q is empty. Return the output at the end of the function.

```
function computeNeighborNodes(graph, sourceNode, d):
    for each node v in graph:
        distance[v] = infinite;
    end for;
    for each neighbor of sourceNode:
        enqueue Q;
    end for;
    List output;
    output.add(sourceNode);
    while  Q is not empty:
        u = dequene(Q);
        if distance[u] <= d
            output.add(u);
        if  distance[u] < d
            for each neighbor  v of u:
                enquene(Q, v);
                if  distance[u] + path(u,v) < distance[v]:
                    distance[v] = distance[u] + path(u,v);
                end if;
            end for;
        end if;
    end while;
    return output;
end function.
```

Figure 2. Computation of the related tables of a single node.

EXPERIMENTS AND RESULTS

The experiment was performed on the Hardo MapReduce cloud computing environment. The cluster consists of 10 Linux machines. About 500 tables were used in the experiment. Two hundred tables have a foreign key that references other tables, while others do not. Each machine has about 10 tables. Each of the 250 tables is about 1GB. The largest table is about 5GB. Assume the client has a space of 50GB for caching. The size of all cached tables cannot be more than 50GB. If each table size is 1GB, then a total of 50/1 = 50 tables can be cached. The last 24 hours of data is cached if it fits in the above algorithm.

Query	Number of Queries
Select single table	**10**
Joint of 2 tables	**10**
Joint of 4 tables	**10**
Joint of 6 tables	**10**
Joint of 10 tables	**10**
Joint of 12 tables	**10**
Joint of 20 tables	**10**
Joint of 30 tables	**10**

Table 1. Types of Queries

Those tables form a graph. The node is the name of the table. There is an edge if a table has a column reference to another table. The **first step** is to generate 100 different queries to test the cache hit ratio. The equal joints and left outer joints are tested. The **next** is to compare the cache hit ratio of three cases**:**

1) Cache data based on the most frequently used tables' active cache.
2) Cache the most recent data - passive cache.
3) Cache the most frequently used data and cache the most recent data, that is, combining the active cache and passive cache.

Case	Number of Tests	Cache Hit Ratio
Active caching	10	29.8%
Passive Caching	10	37.5%
Combining Active and Passive Caching	10	66.2%

Table 2. Results of Experiments

The cache hit ratio was increased to 66.2% when combined with the active caching and passive caching. The cache hit ratio of passive caching is higher than the cache hit ratio of the active caching in the experiments conducted by the authors of this research.

Choosing *n* previous tables such that the total size of *n* tables cannot exceed the expected disk space limit, the experiment shows that the cache hit ratio increases as *n* increases. The hit ratio reaches its highest when *n* is equal to the total number of previously used tables. This implies that the bigger *n* is, the higher the hit ratio is. The query operation result of the last *m* hours is cached. The experiment also shows that the cache hit ratio increases as the size of *m* increases.

CONCLUSION

In this paper, *active caching* refers to caching the most frequently used data, and *passive caching* refers to caching the most recent data. A formula was used to decide whether to cache the source table or query result based on the size ratio of the request and the source table. An algorithm was developed to find related tables most likely to be used in the join operation. The tables are modeled as a graph. There is an edge if a table has a column reference to the column of another table. The experimental result demonstrated that combining the active caching and passive caching improved the cache hit, thus improving the query performance. The proposed algorithm is effective and applicable for join operations of the transactional databases in the cloud. The passive cache will not increase the cache hit ratio if the previous queries are not repeated. The active caching will not improve the caching if the pre-fetched source data does not match the query.

Future work may influence compression of cache data so that more content can be cached. Additionally, a survey with the users may be useful for finding out what tables are frequently used, as that may reflect current or future usage situations.

REFERENCES

1. Ahmed, M. U., Zaheer, R. A. and Qadir, M. A. (2005) Intelligent cache management for data grid, *The Australian Workshop on Grid Computing and e-Research (AusGrid 2005)*, vol. 44, Newcastle, Australia.
2. Altinel, M., Bornhövd, C., Krishnamurthy, S., Mohan, C., Pirahesh, H. and Reinwald, B. (2003) Cache tables: Paving the way for an adaptive database cache, *Proceedings of the 29th VLDB Conference*, Berlin, Germany.
3. Baeza-Yates, A., Gionis, A. and Junqueira, F. (2007) The impact of caching on search engines, *ACM SIGIR'07*, July 23–27, Amsterdam, The Netherlands.
4. Baeza-Yates, R., Gionis, A., Junqueira, F. P., Murdock, V., Plachouras, V. and Silvestri, F. (2008) Design trade-offs for search engine caching, *ACM Trans*. Web, 2, (4), Article 20.
5. Blanas, S., Li, Y. M. and Patel, J. M. (2011) Design and evaluation of main memory hash join algorithms for multi-core CPUs, *ACM SIGMOD*.
6. Bouchenak, S., Cox, A., Dropsho, S., Mittal, S. and Zwaenepoel, W. (2006) Caching dynamic web content: Designing and analysing an aspect-oriented solution, *Middleware, LNCS* 4290, 1–21, IFIP International Federation for Information Processing.
7. Cao, P. and Felten, E. W. (1996) Implementation and performance of integrated application-controlled file caching, prefetching, and disk scheduling, *ACM Transactions on Computer Systems*, 14, (4), 311–343.
8. Chen, S., Ailamaki, A., Gibbons, P. K. and Mowry, T. C. (2007) Improving hash join performance through prefetching, *ACM Transactions on Database Systems*, vol. 32, No. 3, Article 17.
9. Chierichetti, F., Kumar, R. and Vassilvitskii, S. (2009) Similarity caching, *ACM PODS'09*, June 29–July 2, Providence, Rhode Island, USA.
10. Chou, H. T. and Dewitt, D. J. (1985) An evaluation of buffer management strategies for relational database systems, in *Proceedings of the VLDB Conference*, 1985.
11. Doulkeridis, C., Zafeiris, V. and Vazirgiannis, M. (2005) Role of caching and context-awareness in P2P Service Discovery, *ACM MDM*, Cyprus.

12. Fagni, T., Perego, R. and Silverstri, F. (2006) Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data, *ACM Transactions on Information Systems*, 24, (1), 51–78.
13. Gan, Q. and Suel, T. (2009) Improved techniques for result caching in web search engines, *ACM WWW 2009*, April 20–24, Madrid, Spain.
14. Hankins, R. A. and Patel, J. M. (2003) Data morphing: An adaptive, cache-conscious storage technique, *Proceedings of the 29th VLDB Conference*, Berlin, Germany.
15. Houle, M. E., Oria, V. and Qasim, U. (2010) Active caching for similarity queries based on shared-neighbor information, *ACM CIKM'10*, 26–30, Toronto, Ontario, Canada.
16. Irmak, U. and Suel, T. (2005) Hierarchical substring caching for efficient content distribution to low-bandwidth clients, *International World Wide Web Conference Committee (IW3C2), ACM WWW 2005*, 10-14.
17. Liu, Y. A., Stoller, S. and Teitelbaum, T. (1995) Static caching for incremental computation, *ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, La Jolla, California.
18. Luo, Q., Naughton, J. F. and Xue, W. (2008) Form-based proxy caching for database-backed web sites: Keywords and functions, *The VLDB Journal* 17:489–513.
19. Nelson, M. N., Welch, B. B. and Ousterhout, J. K. (1998) Caching in the Sprite network file system. *ACM Trans. Comput. Syst.* 6, (1), 134–154.
20. Ozcan, R., Altingovde, I. S. and Ulusoy, O. (2011) Cost-aware strategies for query result caching in web search engines. *ACM Trans.* Web 5, (2), Article 9.
21. Pandey, S., Broder, A., Josifovski, V., Kumar, R., Chierichetti, F. and Vassilvitskii, S. (2009) Nearest-neighbor caching for content-match applications, *ACM WWW 2009*, April 20–24, Madrid, Spain.
22. Rashid, L., Hassanein, W. M. and Hammad, M. A. (2008) Exploiting multithreaded architectures to improve the hash join operation, *ACM MEDEA* 2008.
23. Rodriguez, P., Spanner, C. and Biersack, E. W. (2001) Analysis of web caching architectures: Hierarchical and distributed caching, *IEEE/ACM Transactions on Networking,* 9, (4).
24. Williamson, C. (2002) On filter effects in web caching hierarchies, *ACM Transactions on Internet Technology*, 2, (1), 47–77.
25. Xie, J., Yang, J. and Chen, Y. (2005) On joining and caching stochastic streams, *ACM SIGMOD 2005*.
26. Zhang, K., Wang, L., Pan, A. and Zhu, B. B. (2010) Smart caching for web browsers, *ACM WWW 2010*, Raleigh, North Carolina, USA.