Efficient and Semantic OLAP Aggregation Queries in a Peer to Peer Network

Yang Kehua and Agnes Manirakiza

Abstract—**This paper presents TrackerCache system - a Peer to Peer (P2P) system for efficient serving OLAP queries and aggregation of data cubes in distributed data sharing system. TrackerCache system provides a central component called tracker, which acts as a directory, keeping track of all submitted queries and a list of peers caching their results. The key idea is to enable peers to share their local caches, to form an efficient and semantic distributed cache sharing system. For this issue, a query forwarding policy is introduced. To face inconsistent information in the client side cache, we proposed a knowledge replacement policy based on Binary Decision Tree (BDT) to evict unwanted data. Then, we present experimental evaluations to demonstrate the efficiency of our proposed solutions.**

Index Terms—**Aggregate queries, distributed data cache, peer to peer network, replacement policy, trackercache.**

I. Introduction

In today's distributed data exchange, decision support systems need interactive ness and demand fast time response. Data warehouse plays an important role in supporting information exchange by collecting and consolidating current and historical data from a variety of several internal and external data sources. Data warehouse support Online Analytical Processing (OLAP) tools which has been proven to be one of the most popular tools for on-line, fast, and effective multidimensional data analysis [1].

However, most of the queries are complex and requiring the aggregation of large amounts of data. In literature, different techniques to speed up a query have been studied and implemented such as pre-computation of aggregates in the database [2], [3], indexing structures and caching in the client side [4], [5] or at middle tier.

In another area, scientific researchers are now encouraged to provide open access to their databases so that results can be widely shared, but this can cause performance problems if the data proves popular. The limitations of server scalability as the number of simultaneous accesses increase problems in file sharing. Meanwhile, that has been very successfully addressed in recent years by the introduction of P2P [6], [7] techniques that harness the power of the clients in order to reduce the load on the server. This paper presents TrackerCache system - a Peer to Peer (P2P) system for efficient serving OLAP queries and aggregation of data cubes in distributed data sharing system.

The system uses tracker server - a BitTorrent protocol component which keep records of which user has submitted which queries in the system. These records are used by new query submitters to help them find peers which can resolve their queries. BitTorrent file sharing protocol [8] has ability to decompose file into small blocks (chunks). Our approach uses chunk based caching [4], [5] that divides the data-cube into uniform semantic regions.

The contributions of this paper includes: The concept of TrackerCache which improves data location and query processing in a P2P data sharing system. Then, we support aggregations in the distributed cache by the use of tracker server. We proposed also a knowledge replacement policy to evict unwanted cached data in order to provide data consistency in the system.

This paper is organized as follows. In Section 2, existing related work is reviewed. In Section 3, an overview of the system is proposed. The techniques used to locate and retrieve chunks are described in section 4. Section 5 presents caching replacement policy which is based on BDT. The details of experiments conducted and analysis of the results are discussed in section 6. Finally, conclusion and future works are presented in section 7.

II. Related Work

BitTorrent [7], [9] is an extremely popular application for sharing large files, based on the principle of decomposing a file into multiple small brocks. In literature [4], Wigan P2P database was designed on BitTorrent approach similar to our work discussing the problem of server overload but in client server database. Active cache [8] is a technique implemented to speed up query that matches data in the cache and can also answer queries that required by aggregation of data.

CubeCache [4] combines multiple client caches into a single query processing and caching system. CubeCache provides aggregation in the cache system.

In addition, distributed OLAP systems require an integrated decision-making system that reflects the global and local schemas of business. Such cooperative interchange has been investigated in [2], [3] and their techniques can be applied to handle integration and mapping tasks. In their works, OLAP aggregate queries in P2P environment are also discussed.

Various semantic caching algorithms exist in the OLAP literature [4]-[5], [8], [10]. Client-side caching [8] is a technique that has been widely deployed to minimize the costs incurred during distributed OLAP operations. Usually, each node caches data independently. Reference [11]

Manuscript received May 8, 2012; revised June 9, 2012.

Yang Kehua is with the department of Computer Science, Hunan University, Hunan, China (e-mail: khyang@hnu.edu.cn).

Agnes Manirakiza is with the department of Computer Engineering and Information Technology, University of KIST, Rwanda.

studied the problem of caching OLAP in Web environments.

III. System Overview

In our model, TrackerCache system introduced is based on P2P network.

A. Motivation

Distributed data sharing system require network of peer systems that maintain full autonomy over its own data resources. Data exchange between peers occurs when one of the peer need data available in the other peers. A central system is assumed to coordinate the aggregate queries in such system. However, in this scenario, designing an additional centralized server clearly would not be the solution. Subsequently, clients should experience an increase in query response time due to server overload, and in the worst case the server may be unable to provide any service at all.

In file-sharing, the problem of server overloading has been addressed by the use of P2P techniques in which users (peers) supply files to each other, so sharing the load. In this paper we propose therefore to use BitTorrent protocol which has emerged as a very popular and scalable P2P file distribution mechanism [6], [7].

In addition, if there is information already cached in the system, the query will be enhanced. Such aggregated data can be a matching query or not as described in section 3.2. Therefore, distributed caching of OLAP queries in a P2P network is a solution to support query process.

B. Tracker Cache System

In TrackerCache system presented in Fig. 1, a user who is sitting at Peer 1 may need to analyze summarized data about for instance, total sales. The central component (tracker server) should fit this distributed OLAP queries. Here, a cached information in each peer meeting the needs will be retrieved and be aggregated at peer 1. The TrackerCache acts the same functionalities as Digame in [12]. However, a key difference TrackerCache doesn't need to download the hall database and it manages distributed database in autonomous way by using a tracker server.

In Bittorrent protocol each file has an extension of .torrent. Each autonomous database has a tracker and the user wishing to retrieve data has to download a .torrent file from web server. A .torrent file consists of the file name, file length, block length, hash information of each block, and the most important information "URL of the *tracker*". Note that a .torrent file can have one or more trackers it depends on the query contains and location. As mentioned above tracker server keeps track of all peers interested in a specific file (or group of files). The tracker does not distribute any actual contents, but only meta-data about it.

When a query arrives at the Tracker from a peer, it checks these adverts to see which other peers could answer the query. In TrackerCache system, it is possible for a downloader's query to match exactly with an advertisement or not. In this, it is similar to Wigan [6]. However, a key difference is that TrackerCache goes beyond this and supports answering queries that are of one or more advertisements from distributed database.

An example of how BitTorrent retrieves and distributes chunks can be found on [7] for more details.

IV. Chunk Based Cache

In BitTorrent protocol [6], [7], large files are divided into small chunk of size 256 kilobytes. Storing files in chunk allows simultaneous transfer downloading or uploading from different neighbors. The downloader learns what chunk the neighbor has and keeps periodically asking from the list of chunks ◦

A. Data Localization

In P2P system peer can join or leave any time or the tracker server can be down while the download process has started. BitTorrent protocol [8], [13] uses random selection and rarest first algorithm as a solution. Rarest first tries to download the chunk with fewest available copies i.e. the rarest chunk first. The benefit of rarest first is not only to evict starvation when some peers depart but also evict starvation across all peers wanting a file.

Localization methods related to BitTorrent, are focused on modification of a random behavior to suggest potentially attractive peers.

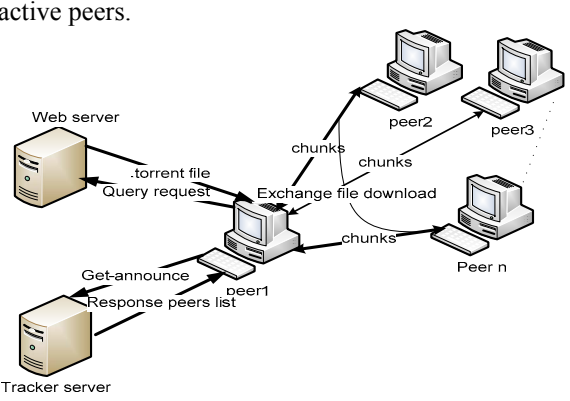


Fig. 1. Tracker cache system

B. The Proposed Query Forwarding Policy

Consider a query to find the following data from peer 1 as shown in Fig. 1.

The policy is described as follows:

Query 1:

SELECT item_sold, dmonth, sum (total_sales)
FROM Product
WHERE dmonth >"june" AND dmonth <"Nov" GROUP BY item_sold, dmonth

Let us assume that tracker server cache contains the adverts whose SQL statements queries are given bellow:

Advert1:

SELECT item_sold, dmonth, sum (total_sales)
FROM Product
WHERE dmonth <"April" AND dmonth >"Sept"
GROUP BY item_sold, dmonth

Advert2:

SELECT item_sold, dmonth, sum (total_sales)
FROM Product
WHERE dmonth < "march" AND dmonth > "July"
GROUP BY item_sold, dmonth

To answer *query 1* there are two alternatives. The first

alternative is to evaluate the queries by aggregating the results of the cached queries while the second alternative is to issue them to the relational backend.

In this policy, we forward messages to web server looking a torrent file. And tracker server locates peers caching results which return a random list of them. Since the messages are forwarded to fewer peers, the time response is high. However, since the numbers of search messages are higher the query forwarding is best suited for the system. Assume that a user issues a query q_i at $peer_i$. The query forwarding works as follows:

- Initialize query at local $peer_i$
- Send request to the web server
- $peer_i$ get .torrent file from web server// each .torrent file define a new overlay network, called swam
- Tracker get announce and send back a random group of the swam peers to $peer_i$ // a list of peer caching results
- $peer_i$ connect and share information on which chunks they have.
- Let Ch_{miss} be the set of chunks that are not available from other peers. Go back to step iv
- Exchange related chunks in Ch_{miss} with $peer_i$
- $peer_i$ aggregates data and send back a query path to the TrackerCache for future reuse

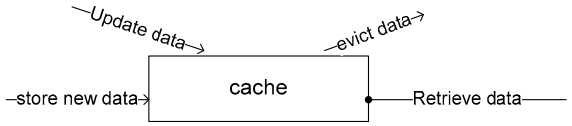
V. Caching Policy

Caching process in OLAP application domain provides not only increase in query performance but also data localization and availability.

Unfortunately, replacement schemes are required because old chunks have to be replaced to make room free for new chunks in the cache.

They are four types of operations which TrackerCache system can handle as shown in Fig. 2. Firstly, the new demand should be cached. Then, cached data should be updated to enhance data consistent. When user needs data, this information should be retrieved from the cache if it is available. Finally, old and unused data must be evicted from the cache to make room free for new data. In this paper emphasis is on replacement issue.

Remind that we have two different caches management in the system. The first is at tracker server and the second at client side. All should be maintained accordingly ◦



Fi g. 2. Cache operations

A. Tracker Server Maintenance

In practice, the data items at tracker server are not evicted from a cache unless the corresponding peer forwarding state is torn down.

To synchronize caching and forwarding state, we have slightly altered scribe's leave procedure. When, a caching peer wishes to leave the network, it has to send a leave message to the tracker server and this clear all cached adverts. When a caching peer issues a scribe notice update message, it is forwarded by a refresh message and propagate the hall network based on the corresponding advert cached at tracker server. The corresponding will remove the cached data and refresh the cached data by new ones. The peer joining the system should send a scribe join notice, and advertise his adverts to the tracker server.

In the next section we describe a knowledge replacement policies used at client side cache.

B. Knowledge Replacement Policy

Below we describe the proposed knowledge replacement policy. The functional attributes used by knowledge replacement policy are based on: *execution time,* which is the time spent to execute the query; *lifetime,* which is the time duration a chunk stayed in the cache; and *access frequency*; which is the number of times a chunk is accessed ◦

To model the victim knowledge replacement policy uses the high *execution time*, the Least Recent Used (LRU) and Most Recent Used (MFU).

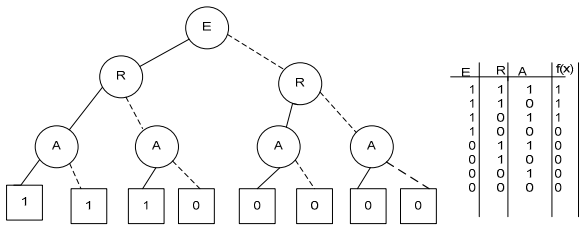
In this paper we combined the above three attributes to build a replacement policy, which is based on Binary Decision Tree. The access number is combined with lifetime and the execution time (measured by the CLOCK) at the same time.

Here, considering time taken to execute the query (to retrieve the chunk) is a key idea to maintain rarest chunks because they are sometimes retrieved from the back end storage or are highly aggregated.

We have combined the CLOCK scheme with the notion of access number. When a chunk is accessed, its access number gets increased. As long as the chunk is accessed this number keeps increasing.

To check the victim we used a binary decision tree algorithm as shown in Fig. 3. Table I shows sample elements used in the experiments. The observation shows that the knowledge replacement policy can be found by the following function shown in equation (1).

$$f(x) = (E \wedge R) \vee (E \wedge A) \quad (1)$$



Fi g. 3. Binary decision tree combined with true table of f(x) -vertex represents decision; follow dashed line for value 0, solid line for value 1 and function value determined by leaf value.

TABLE I : Sample Attribute-based Representation Considered to Make Decision

Example/ chunks	Execution time	Accessed times	Recent	decision
Chunk1	low	many	no	evict
Chunk2	low	less	yes	evict
Chunk3	high	many	yes	stay
Chunk5	high	many	yes	stay
Chunk6	low	many	yes	evict

where E=Execution time, R=Recent data and A= Accessed number

Our knowledge replacement policy is represented in a Binary Decision Tree (BDT) as shown in Fig. 3 and a true table. All victims are represented by 0 otherwise stay by 1.

The observation shows that frequent accessed chunks with high execution time and recent released with high execution time have high probability to stay in the cache otherwise they are evicted from the cache.

VI. Experimental Evaluations

In this section, we describe the results of performance evaluation. In the experiments, we compared the performance of the TrackerCache query forward method with of the CubeCache method. We also analyze the efficiency of knowledge replacement policy. Our experiments are performed on an Intel pentium IV 2.0G MHZ with 2 GB RAM, running on windows. This implementation is written in Java and Oracle 11 g database storing the data.

A. Query Response Time

In this experiment we used ten different queries repeated 50 times on twenty nodes.

On the first stage, CubeCache outperforms TrackerCache. By the time, those peers starting later submit their queries; those starting earlier have received the query results and have advertised these through the Tracker, thus offering an even greater choice of uploaders. By the time TrackerCache has another uploader available, in addition to the seed, which can provide the query results initially; it offers an improvement in performance for two reasons. Firstly, this uploading peer does not have to perform any complex aggregation because it has already the (joined) query results. Secondly, the query result set sizes are, in some cases, considerably smaller than the database tables and thus the uploader does not have to search through a large dataset or filter out any rows. Fig. 4 compares these results and illustrates the average response times for those queries which can be answered by TrackerCache and those which can only be answered by the CubeCache.

B. Effects of Tracker Server Maintenance

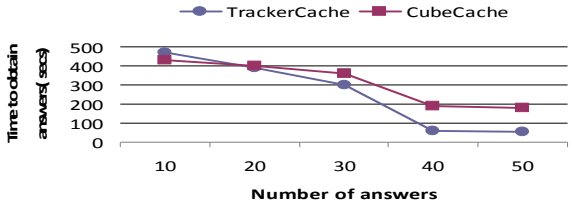
We evaluate the impact of node failure in our system. In this the hit-count (number of chunks located in the cache) measured in the presence of node failures were compared with CubeCache.

We simulated nodes failure to occur with probability of 4 % which is a reasonable estimate with a P2P system with node departure.

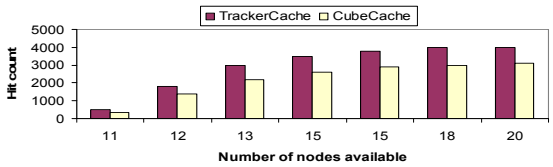
From Fig. 5, we can clearly see that as the experiments progresses, and more nodes fail, TrackerCache perform better than CubeCache. Thus, is because TrackerCache is more resilient to node failure due to its caching policy.

C. Effects of Knowledge Replacement Policy

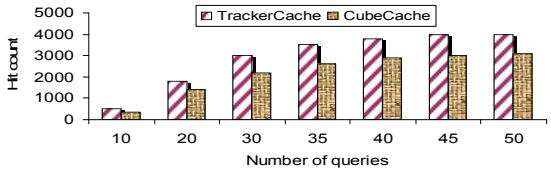
We are now interested on the effects of varying system sizes on the performance of cache replacement policy. Fig. 6 presents the available hit count and number of queries answered while varying the number of queries from 10 to 50.This demonstrates that as the number of queries increase the availability of chunks also increase. Even if, the number is variant (nodes join and leave), the observation shows that TrackerCache with knowledge replacement policy (based on BDT) performs better than CubeCache (based on LRU-CLOCK).



Fi g. 4. Effect of cache management and query processing



Fi g. 5. Tracker cache performance with node failure results



Fi g. 6.Tracker cache performance with respect to number of queries issued.

VII. Conclusion

In this paper we presented the TrackerCache system - a distributed caching system for OLAP results.

The architecture is derived from the popular BitTorrent file-sharing protocol. As shown in the experimental evaluation, TrackerCache achieves significant performance gains with respect to CubeCache.

This is accomplished by the query forwarding algorithm to aggregate distributed data cache, the data localization techniques for efficient locating data in the network and a semantic knowledge replacement policy to evict unwanted chunks the system.

The further investigation would include a real implementation to evaluate the efficiency of the system. Researchers should be interested to investigate a technique to control a mapping schema in the cache storage in this proposed system. Another direction would be techniques to manage cached storage due to node failure and departure.

References

[1] S. Chaudhuri and U. Dayal, “An overview of data warehousing and OLAP technology,” in SIGNOD Rec. New York, NY, USA: ACM, vol. 26, no. 1, pp. 65-74, 1997.

[2] M. Minuto and A. Vaisman, “Aggregate queries in peer-to-peer OLAP,” In *DOLAP, Washington,* DC, USA, pp. 102–111, 2004.

[3] A. Vaisman, M. M. Espi, and M. Paradela, "P2P OLAP: Data model, implementation and case study," *Information Systems, Elsevier Science Ltd*, vol. 34, no. 2, pp. 231-257, 2009.
[4] S. Sangeetha, F. Brian, Liu, and Ling, "CubeCache: Efficient and Scalable Processing of OLAP Aggregation Queries in a Peer-to-Peer Network," *Technical Report GIT-CERCS-07-12,* Georgia Institute of Technology, 2005.
[5] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K. L. Tan, "An adaptive peer-to-peer network for distributed caching of olap results," in *SIGMOD Conference*, Madison, Wisconsin, USA *:ACM*, pp. 25–36, 2002.
[6] J. Colquhoun and P. Watson, "Query Matching in a BitTorrent-Based P2P Database System," *Technical Report Series,* no. CS-TR-1184, *Computing Science, Newcastle Univ*. Tyne, 2010.
[7] BitTorrent protocol report (2009) Time out: SridharBR [Online]. Available: http://www.slideshare.net/SridharBR/bit-torrent-protocol-report.
[8] P. Deshpande and J. F Naughton, "Aggregate aware caching for multi-dimensional queries," in *EDBT*, pp. 167-182, 2000.
[9] A. Legout, G. U. Keller and P. Michiardi, "Rarest First and Choke Algorithms Are Enough," *In: Internet Measurement Conference*, 2006, pp. 203 – 216.
[10] T. Loukopoulos, P. Kalnis, I. Ahmad, and D. Papadias, "Active Caching of On-Line-Analytical-Processing Queries in WWW Proxies," In *Proc. 30th Int'l Conf. Parallel Processing (ICPP '01),* Sept. 2001, pp. 419-426.
[11] Bittorrent protocol (2010), Time out: bittorrent [Online]. Available: http://www.bittorrent.com.
[12] C. P. D. Laborda, C. Popfinger, and S. Conrad, "Digame: A Vision of an Active Multi database with Push–based Schema and Data Propagation," In: *GI-/GMDS-Workshop on Enterprise Application Integration (EAI'04), Oldenburg*, 2004.
[13] P. Deshpande, K. Ramasamy, A. Shukla, and J. F.Naughton, "Caching multidimensional queries using chunks," in *SIGMOD Conference ACM Press*, pp 259-270, 1998.

Yang Kehua was born in China, 1979. He received B.Sc. degree in Computer Science from Xiang Tan University, Hunan, China, in 1999, M.Sc. degree at Southeast University, Jiangsu , China, in 2005.
He is currently a Teacher in the department of Computer Science, Hunan University, Hunan, China, since 2005. His research interests include Database and Embedded software development. He worked as Computer Technician at Xinhua Water Supply company, Hunan, China from 1999/07~2000/08.

Agnes Manirakiza was born in Rwanda, 1978. She received B.Sc. Degree in Computer Engineering and Information Technology from Kigali Institute of Science and Technology (KIST), Kigali-Rwanda, in 2005, M.Eng degree in Computer Engineering from Hunan University, China in 2011. She is currently with the department of Computer Engineering and Information Technology, University of KIST, Rwanda where she lead a research group that investigates object-oriented Databases issues since 2011. She has worked as Timetable and Logistic Officer at KIST, Kigali, Rwanda from 2006/02~2011/07