A Hybrid Cache Investment Strategy for Distributed Database Queries

Sanju Gupta
The IIS University
Jaipur, India

Swati V. Chande, PhD
IIIM Jaipur, India

ABSTRACT

Caching is one of the most popular techniques for last decades for improve the performance of distributed database queries. There are different techniques to implement the caching. This paper will present one of the caching methods, i.e. Cache Investment Technique. This technique is use to select best caching candidates item .These candidates item is not only useful for current query but also useful for subsequent queries. This paper will review initially existing cache investment policies and their comparative analysis in distributed environment. Then a new feasible policy proposed which is hybrid of existing policies. This proposed policy is based on existing a history based policies .This new policy gives us better candidate item, this improves the hit ratio. Improvement in hit ratio ensures the reusability of stored data efficiently. Due to this efficient reuse of stored data, lesser amount of data is required to be retrieved from remote location. Thus it improves the performance of queries in distributed database system. This paper will present the architecture of this proposed policy and give the detail explanation of each module of proposed policy.

Keywords

Distributed database, Caching, Cache investment .Investment Cost, Return on Investment (ROI)

1. INTRODUCTION

In the distributed system, many activities occur simultaneously. Usually, common resources like Web/application servers, cache servers, database servers and underlying network are shared by many clients. Computing load distribution is done by distributed systems. Sharing and allocation of resources is a major challenge in designing distributed architecture. Strategies like Caching, clustering, load-balancing, pooling and time-sharing improves the system performance and availability [3]. This paper focus on caching and one of the caching method that implementing caching i.e. Cache Investment in the distributed environment.

Emerging distributed query processing system support flexible execution strategies. In which each query can be run using data shipping or query shipping or combination of both. Dynamic data caching can provide excellent availability and performance benefit in any distributed environment. A Circular Dependency arises when dynamic caching and flexible execution are combined: query operator placement decisions are based on (cached) data location but Caching occurs as a by-product of query operator placement. Due to this dependency the query optimization decision/optimal Plan for individual query can actually produce sub optimal performance for all queries in long run. If the circular dependency is not addressed, a distributed query processing system will suffer from suboptimal performance and poor utilization of caching resources. To address this problem, in 1997 Mr.Kossman and Mr.Fraklin have developed Cache Investment. Cache investment is a technique for combining data placement and query optimization in a manner that does not change the query optimizer directly. The cache investment based on some cache investment policies decided that some part of a query would be a good idea to cache. It affect data placement by influencing the optimizer to make suboptimal choice regarding operator site selection. These suboptimal choices will be based on policies producing cached data placement beneficial for subsequent queries (8).

The paper is organized as follows: Section 2 explains Cache investment and its policy in distributed database. Section 3 describes the comparative analysis of existing cache investment policies. Here it was found that there are some areas where improvement in the performance of queries can be done by changing or modifying existing policies Section 4 gives the explanation in detail of proposed technique for distributed environment. Section 5 presents the conclusions of this paper.

2. CACHE INVESTMENT AND ITS POLICIES IN DISTRIBUTED ENVIRONMENT

Cache Investment, is a technique for combining data placement and query optimization. Cache investment keeps statistics and create copies of data at clients only if these copies are beneficial. The foremost objective of cache investment is alike to that of any caching approach namely, to place copies of data closest to where they will most likely be accessed. Thus, the basic policies for cache investment rely on insights similar to those used in other caching schemes. But some are the aspects of cache investment that differentiate it from these other approaches of caching. These are: [9]

- Its method of integration and utilization of the query optimizer,
- Its applicability to complex relational queries; and
- Its consideration of both base data and index data as investment candidates' item.

Cache investment is implemented in existing system as a module that sits outside of the query optimizer. For individual queries this module sometime influence the optimizer to make suboptimal operator site selection in order to effect a data placement which will be beneficial for subsequent queries. Alternately can say that, it influence the optimizer for investing the resource during the execution of one query in order to get benefit in later queries. Because this module only influences the optimizer, it is always up to the optimizer to determine if a suboptimal choice is advisable. Here are two basic thoughts behind cache investment. The first is to carry out what-if analyses in order to decide whether it is significance caching parts of a table or index. More precisely, what-if analyses are used in order to (1) calculate the cost (i.e., investment cost) of loading a client's cache with parts of a table and (2) to calculate the benefits of caching parts of a table. The second is to pull out the optimizer, if these queries involve data that should be cached at the clients than optimizer has to perform queries at client's .This way, data copies are faulted in at these clients and cache can be used in subsequent queries. Queries that involve data that should not be cached be supposed to be executed preferably at servers without additional cost for faulting in data [10].

2.1 Types of Cache Investment Policies

The cache investment based on some cache investment policies decided that some part of a query would be a good idea to cache. Cache Investment Policies decide that when and for which tables the investment required initiating caching. These policies are used for each query that is executed at a client and can influence the operator site selection for that query. Types of cache investment policies are explained in following figure-

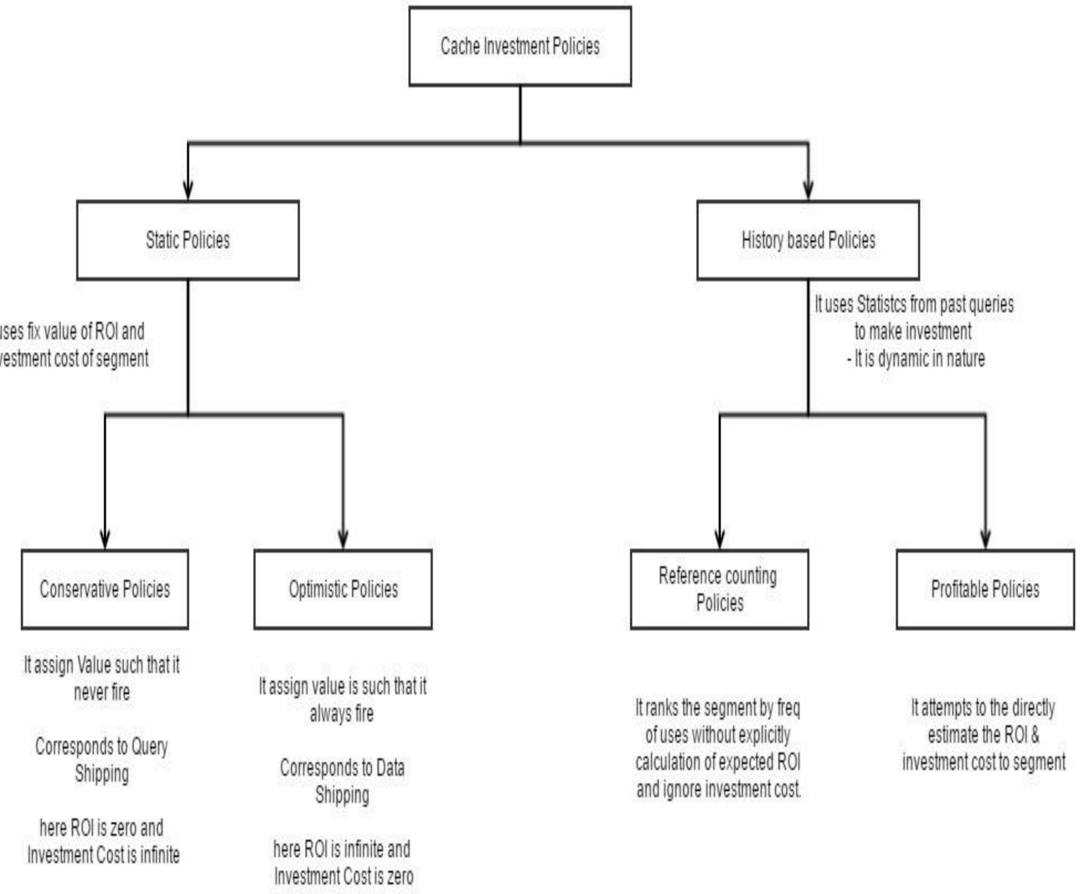
Cache Investment Policies
Static Policies
History based Policies
It uses fix value of ROI and investment cost of segment
It uses Statistcs from past queries to make investment
- It is dynamic in nature
Conservative Policies
Optimistic Policies
Reference counting Policies
Profitable Policies
It assign Value such that it never fire
Corresponds to Query Shipping
here ROI is zero and Investment Cost is infinite
It assign value is such that it always fire
Corresponds to Data Shipping
here ROI is infinite and Investment Cost is zero
It ranks the segment by freq of uses without explicitly calculation of expected ROI and ignore investment cost.
It attempts to the directly estimate the ROI & investment cost to segment

Figure 1: Types of Cache Investment Policies

2.1.1 Static Policies

It assigns fixed values for investment and Return on Investment (ROI) to fragments independent of any history. There are two types of static policy-

A) Optimistic Policy-

It assign value in such manner that it never fires. It sets the ROI of all fragments to be ∞ and the investment to be 0. The behaviour of the Optimistic policy corresponds to that of data shipping.

B) Conservative Policy-

It sets the ROI of all fragments to be 0 and the investment cost to be ∞. So that it never considers any fragments to be candidates' .Thus, Conservative policy never fires. The behaviour of the Conservative policy corresponds to that of query- shipping.

2.1.2 History- based Policies

History based policies are mainly two types- Reference counting and profitable. Both policies try to adapt the workload at each client based on the past history of the queries at that client. They differ in that the profitable policy attempts to directly estimate the investment and ROI for the table. While the reference counting policy is simpler, it ranks fragment by their frequency of use, without explicitly calculating expected ROI and ignore investment cost.

1. Reference Counting Policy

The reference counting policy considers only the most frequently used tables to be candidates and ignores the cost of investment.

The value of a table for reference counting is a count of the number of queries in which a table is used, possibly weighted by the regency of those accesses as determined by α parameter.

This policy does not compute estimated ROI for the table and it ignores the cost of investment. Instead, it decides on which table should be candidates based on the value it maintains for each table, the size of the table, and the size of the client cache.

This policy is simple and easy to implement. But this policy is not suitable for systems that are designed for process many updates transactions.

2. Profitable Policy-

The profitable policy calculates an expected ROI for each fragment and chooses as candidates those fragment who have the highest ROI and where investment cost is less than their expected ROI. Although the profitable policy is good enough as it gives the emphasis on investment cost and ROI. This policy attempts to more closely approach the ideal algorithm of cache investment by directly estimating the ROI of fragment and taking into account the cost of investment. This policy also has some shortcoming such as – This policy is less accurate than it could be because the estimation is performed for each table individually rather than combinations of tables. Another shortcoming of this policy is that in the ROI calculation the size of the cache is not considered.

3. Comparative Analysis of Existing Cache Investment Policies

Table 1: Analysis of Existing Cache Investment Policies

Type of Policy	Name of Policy	Description	Advantage	Disadvantage	Suitable for
Static Policy	Optimistic Policy	It set the ROI of all fragments to be ∞ and the investment to be ϴ	It place all scans at the client in order to exploit client caching	Server resources are not efficiently use.	Data shipping system
Static Policy	Conservative Policy	It assigns the ROI of every fragment to be ϴ and the investment cost to be ∞. So it never considers any fragment to be candidate.	Efficient uses of server recourses because the optimizer places all scans at servers.	It client caches are always empty.	Query shipping system
History Based Policy	Profitable Policy	It calculate an expected ROI and investment for each fragment and choose as candidates those fragment who have the highest ROI and where Investment cost is less than their expected ROI	The Policy is good enough because it gives the emphasis on investment and ROI. This Policy attempts to more closely approach the ideal algorithm of cache investment by directing estimating the ROI of fragment and taking into account the cost of investment	In the ROI calculation the size of the cache is not considered, it is not as much accurate as it could be because the estimation is performed for each table individually rather than combinations of tables.	Hybrid Shipping system
History Based Policy	Reference Counting Policy	This policy considers only the most frequently used tables to be candidates and ignores the cost of investment.	This Policy is simple and easy to implement	This policy is not suitable for system that are designed for process many updated transaction.	Hybrid shipping system

4. PROPOSED POLICY-

This proposed policy is a hybrid of existing history based policies. From the detail analysis of the literature it was found that there are some areas where it can improve the shortcoming of reference counting and profitable policies. For this a new feasible policy is proposed which uses the advantage of these existing policies. This proposed policy is based on Kossmann and Franklin work and gives us better candidate item, this improves the hit ratio. Improvement in hit ratio ensures the reusability of stored data efficiently. Due to efficient reuse of stored data, lesser amount of data is required to be retrieved from remote location. Thus it improves the performance of queries.

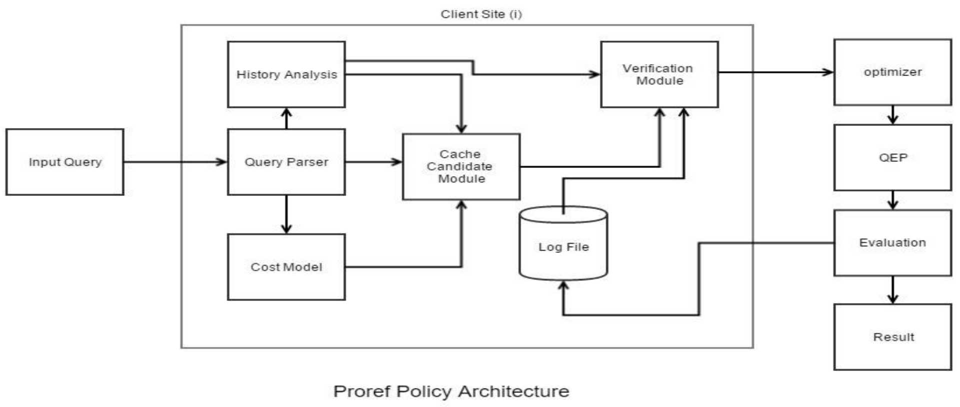

Figure 6: ProRef Policy Architecture

4.1 Formula for the Proref Policy:

To identify the cache candidate Following Formula is used for Table (q) at client (i)

$$K^i_t(q)= V^i_t(q) + R^i_t(q) \qquad \text{-----} \qquad (I)$$

Where

$K^i_t(q)$ = Proref Policy Value for table (q) after the execution of query (t)

$V^i_t(q)$ = value of table (q) assigned after the execution of query (t)

$R^i_t(q)$=Improvement in Response time after the execution of query (t)

For Value of table $V^i_t(q)$ calculation following formula used

$$V^i_t(q) =C^i_t(q) + \alpha * V^i_{t-1}(q) \qquad \text{-----} \qquad (II)$$

Where:

$C^i_t(q)$ =Component of value for reference count

$C^i_t(q)$ =0 if table is not used in query

$C^i_t(q)$ = 1 if table is used in query

α = Aging Factor ($0\leq \alpha \leq 1$). α=1 is used to give equal weightage

To calculate Response time Rit(q) following formula used

$$R^i_t(q) = O^i_t(q) - L^i_t(q) \qquad \text{-----} \qquad (III)$$

Where:

$O^i_t(q)$ = Response time to execute query with present cache condition using optimal plan for table (q)

$L^i_t(q)$= Response time to execute query with caching the table (q)

4.2 Modules for Proref policy

4.2.1 Query parser:

This Component is used to scan the name of table from the Input query.

Input From	Query
Output to	• History Analysis • Cost Module • Cache Candidate Module

4.2.2 History Analysis:

This Module is used to maintain history information about tables at client site. The information stored for a table is number that represents a value for the table based on the history of queries at that client. The values of table at a client are updated after the execution of every query at that client. The value of every table is initially set to 0. To maintain the history information a table to be created at each client site.

History_table (S.No, table_name, table_size, value, value/size_ratio).

Formula (II) is used for calculating the value. Site Information of tables and their update information can get from data dictionary. If the table got updated put the value of that table $V^i_t(q)$ =0

Input From	Query Parser, Data Dictionary
Output to	1. Verification Module 2. Cache Candidate Module

4.2.3 Cost Module:

This Module is used to calculate improvement in response time for the table used in present query using following method:

- Calculate Response time $O^i_t(q)$) using optimal plan (using existing cache condition)
- Calculate Response time $L^i_t(q)$ by caching table q (Considering in addition to optimal plan table q is fully cached)
- Difference between Response time $O^i_t(q)$) and $L^i_t(q)$=

Input From	Query Parser
Output to	Cache Candidate Module

4.2.4 Cache Candidate Module:

This Module used to identify the cache candidate. It can calculate by using following way. To get Proref policy Value $K^i_t(q)$ using formula (I). Find the table of highest Proref policy value $K^i_t(q)$ and this is our suggested cache candidate.

Input From	History Analysis Cost Module Query Parser
Output to	verification Module

4.2.5 Verification Module:

This Module is used to check whether suggested cache candidate is feasible to use or not. For this following verification check is used:

- Whether suggested cache table is already cached or not.
- Size of suggested cache table is ≤ size of available cache memory.
- Value/size ratio of suggested cache candidate table > Value/size ratio of existing cache candidate

Input From	History Analysis Cache candidate Module Log File
Output to	Optimizer

4.2.6 Log File:

This file is used to gather information of already cached table. For this it is creating a log file table.

Log_File (S.No, table_name, cached)

Input From	Query Evolution
Output to	Verification Module

5. CONCLUSION

Query performance can be improved by using of cache technique by providing required data closer to the entity required it than the source where it stored. This paper discussed about a caching method i.e. Cache investment . It is a method or technique to identify the cache candidate and provide that to the existing optimizer to improve the performance of the database queries in distributed system. To find the best cache candidate, cache investment uses their policies. There are mainly two types of cache investment policies these are – Static and History based policies. These policies help optimizers to select best candidate for caching. Here this paper done the comparative analysis of existing cache investment policies and it found the inference that there are some areas of improvement and the performance of queries can be improved by changing or modifying existing cache investment policies. To extend the work of Kossmann and Franklin a new feasible policy is proposed which uses the advantages of both history based policies. In this paper it has explained the proposed policy architecture in detail. The proposed policy helps us to get the better cache candidate item, which in turn improves the hit ratio. This ensures efficient reuse of stored data. Lesser amount of data is required to be retrieved from remote location due to efficient reuse of stored data. Thus it improves the performance of queries in distributed database system.

6. REFERENCES

[1] Donald Kossmann , Michael J. Franklin and Bjorn Thor Jonsson, "Performance Tradeoffs for Client-Server Query Processing" , ACM – SIGMOD Conference on Management of Data , New York,1996.

[2] Donald Kossmann, " The State of the Art in Distributed Query Processing", ACM Computational Surveys, vol. 32, Dec. 2000.

[3] Abhijit Gadkari, " Caching in Distribute Environment", The Architecture Journal,2009.

[4] Shaina,Anshu Kamboj, " High Performance E-Business using Application Level Caching", International Journal of Advanced Research in Communication Engineering, vol3,issue sep.2014.

[5] Mantu Kumar,Neera Batra and Hemant Aggarwalo, "Cache Based Query Optimization Approach in Distributed Database",IJCSI,Vol.9, Nov.2012.

[6] Konard G.Beiske,Jan Bjorndalen,Jon Olav Hauglid, "Semantic Cache Investment" , NIK-2009 conference.

[7] Norvald H. Ryeng, Jon Olav Hauglid, and Kjetil Norvag , "Site-Autonomous Distribted Semantic Cachig", SAC,2011 .

[8] Donald Kossmann , Michael J. Franklin, "Cache Investment Strategies", Univ.of MD Technical CS-TR-3803 and UMIACS-TR -97-50,May 1997**.**

[9] Ideh Azari ," Efficient Execution of Query in Distributed Database Systems", 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE).

[10] Donald Kossmann , Michael J. Franklin,Gehard Drasch, "Cache Investment : Integrating Query Optimization and Distributed Data Placement," ACM Transaction on Database System (TODS), Dec. 2000.

[11] Michael J. Carey,Franklin J. Carey, Miron Livny ,"Local Disk Caching for Client-Server Database Systems *", Computer Science Department University of Wisconsin-Madison,1994.

[12] Doshi P. and Raisinghani V., "Review of Dynamic Optimization Strategies in Distributed Database", Electronics Computer Technology (ICECT), 3rd International Conference, April 2011.

[13] Yan T,IacobesnM,Garcia-Mo Lina H,"Introduction of Query optimization of distributed database", WAM Press, I 999.

[14] Alaa Aljanaby, Emad Abuelrub, and Mohammed Odeh,"A Survey of Distributed Query Optimization", The International Arab Journal of Information Technology, Vol. 2, January 2005.

[15] Elmasri R. and Navathe S. B., " *Fundamentals of Database Systems*, Reading", MA, Addison-Wesley, 2000.

[16] Donald Kossmann , Michael J.Franklin,Gehard Drach,"Cache Investment for Indexes",VLDB Conference,Feb,1998.

[17] Hua-Ming Liao, Guo-Shun Pei, "Cache-Based Aggregate Query Shipping: An Efficient Scheme of Distributed OLAP Query Processing", JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 23(6): 905{915 Nov. 2008

[18] Ruby Bhati ,Nitika Bansal, S K Jha," Distributed Database System:The Current Features And Problems?", International Journal of Computer Science and Management Research, Vol 2 , March 2013