A Cooperative Approach to View Selection and Placement in P2P Systems

Zohra Bellahsene, Michelle Cart, Nour Kadi

A Cooperative Approach to View Selection and Placement in P2P Systems
(Short Paper)

Zohra Bellahsene, Michelle Cart, and Nour Kadi

LIRMM - Université Montpellier 2
161 rue Ada 34095 Montpellier cedex 5, France
firstname.name@lirmm.fr

Abstract. P2P systems are becoming increasingly popular as they enable users to exchange digital information by participating in complex networks. However, limited work has been done on employing materialized views in P2P data warehousing systems. We argue that this technique can be applied in P2P data sharing for (i) saving work for frequently asked queries and (ii) increasing availability in cases of failures. In this paper, we present an approach for dynamically selecting an effective set of views to be materialized and place them in key points in the P2P system so as to achieve the best combination of good query performance and low view maintenance cost, given a limited amount of storage space at each peer. Moreover, as the system is dynamic, our approach continuously monitors the incoming query and adjusts the system configuration by removing materialized views in order to replace the less beneficial views with more beneficial ones.

1 Introduction

In data warehousing and cloud computing systems, the presence of appropriate materialized views can significantly improve query performance. The goal of view selection process is to find a set of views that minimizes the expected cost of evaluating the queries of the workload. Traditionally, view selection has been carried out statically. With static view selection, a system administrator decides what kinds of queries might be carried out in the system. Most of the proposed approaches are based on query workload and choose accordingly the set of views to materialize. Obviously, static selection of views has several weaknesses: (i) the query workload is often not predictable; (ii) even if the workload can be predicted, the workload is likely to change, and the workload might change so quickly that the system administrator cannot adjust the view selection quickly enough. Therefore, dynamic view selection approaches have been developed [6,7].

The view materialization issue in P2P setting can be seen as two sub-problems. First, the view selection problem is to choose an appropriate set of views to be materialized that can improve the performance of the system. Second, the data placement problem is to decide where these views should be placed on the peers so that the whole query workload can be executed in the fastest possible way. The specific contributions of our work are as follows.

R. Meersman et al. (Eds.): OTM 2010, Part I, LNCS 6426, pp. 515–522, 2010.

- A cost model for selecting the candidate views taking into account query processing cost, view maintenance cost and network transfer cost, all at once.
- A view method deciding which candidate views to materialise. In addition, it provides a strategy to replace materialized views by more beneficial ones in the case of reaching the storage limit in a peer.
- Two policies for the data placement problem; each one provides a degree of cooperation between the peers in a cluster: (i) Isolated Policy: places the views closest to where they are more frequently accessed (ii) Voluntary Policy: approximates an ideal policy by trying to make the loss of replacing the already materialized views minimal.

The rest of the paper is organized as follows. In Section 2 presents the problem of view selection problem in P2P context and a framework for representing the view queries in order to detect common sub-expressions. Section 3 contains the cost model used in our approach. Then, the strategy for selecting the views to be materialized is described in Section 4. In Section 5, two data placement policies are proposed. An overview of related work is presented in Section 6. Finally, Section 7 contains the conclusion and future work.

2 View Selection Problem in P2P Systems

A view is a derived relation defined by a query in terms of source relations and/or other views. It is said to be materialized when its extent is computed and persistently stored (otherwise, it is said to be virtual). We consider Selection-Projection-Join (SPJ) views that may involve aggregation and a group by clause as well. The general problem of view selection is to select a set of views to be materialized that optimizes both the view maintenance [8] and query processing time given some constraints like storage space. To find the optimal solution satisfying all constraints is a NP-complete problem, therefore it is necessary to develop heuristics.

In P2P systems, the problem can be formulated as follows. Assume a model containing N peer nodes, in which each node n_j has associated storage B_j and query workload $Q = \{Q_{j1}, Q_{j2}, \ldots, Q_{jk}\}$, that changes over time, where each query Q_{ji} has an associated non-negative weight f_{ji} which describes the frequency of Q_{ji}. Every pair of nodes ns and nt is connected by an edge with a cost $Ct(ns, nt)$ per unit of data transferred. The issue is to dynamically materialize a set of views $M = \{V_1, V_2, \ldots, V_m\}$ and place each of them at a proper node in order to provide high query performance time and low view maintenance cost of the entire system taking into account the space constraint and the cost transfer between the peers.

We will use the Multi View Materialization Graph (MVMG) framework as in our previous work [1] for representing views to be materialized in order to exhibit common sub-expressions between different queries. The MVMG, which is similar to the AND-OR DAG representation of queries in multi query optimization, is a bipartite Directed Acyclic Graph (DAG) composed of two types of nodes: AND nodes and OR nodes (i.e., operation and view nodes respectively). In fact, the MVMG represents AND-OR DAGs of several queries in a single DAG. In this work, the MVMG is built incrementally because the view selection method is applied as a query arrives. We borrow the rule provided in [9] for identifying common sub-expressions. For example, equivalent nodes,

obtained after applying join associativity property, are replaced by a single equivalence node. In addition, we consider metadata like frequency at each node of the graph.

We assume that the data warehouse is organized along with XPeer architecture [2]. The nodes are grouped into clusters; at each cluster there is a super-peer, called cluster-peer, which has extra capabilities and duties in the network. It is in charge to take the decision for materializing the appropriate set of views, which will increase the performance of the peers within this cluster. Each cluster-peer acts as a centralized server to a subset of nodes in the cluster representing clients. It can be considered as a reference for its clients as it keeps a full repository about all the materialized views at each peer in the cluster and it is the responsible for the configuration, and administration of the materialized views in its clients. Our approach consists in continuously monitoring the incoming queries at any peer of the cluster and adjusting the system configuration in order to maximize query performance at the peers of the cluster. Our method proceeds in three steps, as follows:

- Select candidate views for materialization.
- Decide which views to materialize or dematerialize.
- Place the materialized views on the appropriate peers.

3 Cost Analysis of View Selection Strategy

As mentioned earlier, a MVMG is used for modeling the workload of the entire cluster. We present the following definition for the MVMG at the cluster c.

- For any leaf node r which represents a base relation, $f_u(r)$ denotes the update frequency of r.
- For any root node q which represents a global query, $f_{p_i}(q)$ denotes the query access frequency of the query q at the peer p_i which belongs to cluster c.

The frequency of a query q, $f(q)$, at all peers within the cluster is computed as follows:

$$f(q) = \sum_{p_i \in cp} f_{p_i}(q)$$

where cp is the set of all peers in cluster c.

3.1 Cost Model

For every vertex v in the MVMG

- $C_m(v)$ denotes the view maintenance cost of v based on changes to the base relation r if v is materialized. Here, we consider the incremental maintenance, so $C_m(v)$ represents the cost of computing the differential from the data origin plus the cost of transferring it to the specified cluster.
- $C_e(v)$ denotes the cost of computing v in its data origin. It is the average cost processing time.
- $C_n(p_i \rightarrow p_j)$ denotes the network transfer cost. It is the cost per unit of data transferred for the edge between two peers p_i, p_j.

– $C_t(v, p_i \rightarrow sp)$ denotes the total cost to get v from its data origin p_i to the cluster-peer sp. It includes the processing and transfer costs. Finally, we have:

$$C_t(v, p_i \rightarrow sp) = C_e(v) + C_n(p_i \rightarrow sp) * size(v)$$

where $size(v)$ denotes estimated size of materialized view v.

3.2 Criteria of Selection

The cluster-peer assigns a value, called *goodness(v)* for each view v in MVMG. This goodness represents the cost for materializing the view at the cluster. Obviously, if the view is expensive to compute or is frequently used then it is more beneficial to keep it locally. On the other hand, if the view is frequently updated then materializing this view becomes less attractive especially if the maintenance cost is relatively high. Let $CS(v)$ be the saving if view v is materialized and let $CM(v)$ be the view maintenance cost according to the update frequency.

– $CS(v) = \sum_{q \in Q_v} f(q) * C_t(v, p_i \rightarrow sp)$
– $CM(v) = \sum_{r \in I_v} f_u(r) * C_m(v)$
where Q_v denotes the set of queries that use view v and I_v denotes the base relation instances, which are used to produce v.

Then, the following formula calculates the goodness of view v:

$$goodness(v) = CS(v) - CM(v)$$

4 View Selection Strategy

Pre-Selection. A first phase of selection on views is performed to eliminate the views that provide no benefit. A view is considered as beneficial if and only if its materialization reduces significantly the query processing cost without increasing significantly the view maintenance cost. The first step of our approach pre-selects a set of views called candidate views V_C from the set of views from the view query. These views should be the most beneficial views for the peers of the specified cluster and are computed by the following formulae:

$$goodness(V_C) = \sum_{v \in V_C} goodness(v, V_C) \text{ be maximal} \qquad (1)$$
$$\mid V_C \mid \text{ be minimal}$$

Final Selection. As mentioned earlier, the candidate views returned from the first phase are promising for materialization, but they will not be materialized by default because we have given a limited amount of storage space. Therefore, any candidate view will be materialized if it is beneficial enough to the peers of the cluster. Now, suppose that the candidate v will be materialized in the peer p_j. If p_j has enough space to store v_i, then it will be materialized at p_j immediately. However, if the storage limit at p_j is reached, then v_i will be materialized only if it proofs to be more beneficial than those which are

already materialized. In this case, the views with less benefit should be removed to free the necessary space for materializing the new candidate. Otherwise, the candidate view will be ignored.

The final decision to materialize view v is based on the notion of benefit.

$$Benefit(v) = goodness(v) - goodness(V^-)$$

where V^- is the set of already materialized views to be removed in order to free the necessary storage space.

The replacement strategy will be applied at any peer to calculate the benefit of the candidate v_{new} over the views that are already materialized at that peer. At the first, it will select all materialized views at certain peer with lower goodness value than v_{new}, and adds these views to the set of candidate (for removal) views V_c^-. Finding the set of views to be removed is formed by solving an instance of the KNAPSACK problem. Given the set of objects in V_c^-, the size of the storage space s_{max}, knapsack will find the views with highest goodness to remain materialized. The rest of views V^- will be removed from peer p in order to free enough space to materialize v_{new} according to the following formulas:

$$size(V^-) + freespace > size(v_{new})$$

$$goodness(V^-) < goodness(v_{new})$$

5 View Placement Policy

In this section, we present two policies for where to materialize each candidate view; each policy defines a degree of cooperation between the cluster peers. However, both have the same goal, which is getting better performance with lower traffic between the cluster peers.

5.1 Isolated Policy

The principle of the Isolated Policy is that the cluster-peer tries to solve the problem of each peer in the cluster separately without using the capabilities of the other peers within the cluster. In the case where there is not enough space in this peer, it will not use the free resources on the other peers but instead the replacement policy will be used to free enough space in that peer in order to store the new candidate view if it is more beneficial. For each candidate view, the cluster-peer will find the peer that uses this view more frequently. The candidate view will be materialized at that peer if it has enough space. However, if this peer reaches the storage limit, then the candidate view will be materialized only if it is more beneficial than the already materialized views in that peer. The Isolated Policy strategy is divided into two phases:

Phase 1: While there is enough storage space at a peer, the candidate views will be materialized.

Phase 2: If the storage limit in is reached, a candidate view will be materialized only if its benefit is positive. In this case, the views in V^- will be removed to free enough space to store v_{new}.

5.2 Voluntary Policy

In contrast to Isolated Policy, Voluntary Policy attempts to exploit under-utilized resources (e.g., storage space) that may exist in some neighbor peers and at the same time avoid wasting any view that has been materialized. Assume that we want to materialize the candidate v_{new}. The cluster-peer will find the peer that has the maximum frequency in using v_{new} and at the same time has enough space to materialize it. However, in the case where all the peers in the cluster reach the storage limit, then the cluster-peer will select the peer that provides the highest benefit with the lowest transfer cost for materializing v_{new}.

The strategy of Voluntary Policy is divided into two phases:

Phase 1: Try to store the new candidate v_{new} without loosing any already materialized view by finding the peers at the cluster that have enough free space to store v_{new} then select the one that has the highest frequency in using v_{new}.

Phase 2: In the case where not enough space is found at any peer in the cluster, therefore it is necessary to remove some already materialized views. In order to reduce the loss of removing already materialized views in the cluster, the set of views to be removed which are the least beneficial views to all the peers in the cluster, should be computed. For that, the cluster-peer chooses the peer associated with the maximum value of $BenPeer$. If this value is positive then the candidate v_{new} will be stored in that peer after removing the less beneficial view V^- specified by the replacement strategy. The criteria for selecting the peer is based on the value of the following expression:

$$BenPeer_{p_i} = \frac{b_i}{C_n(sp \rightarrow p_j)}$$

The first term of this formula b_i represents the benefit of materializing the candidate view v_{new} versus the loss of removing the already materialized views at the peer p_i. The second term $C_n(sp \rightarrow p_j)$ represents the cost of transferring v_{new} from the cluster-peer to p_i.

6 Related Work

6.1 Dynamic View Selection in Data Warehousing

DynaMat [7] is a system aimed to unify the view selection and the view maintenance problems. The principle of this system is monitoring constantly the incoming queries and materializing their query results given the space constraint. During the update only the most beneficial subset of materialized views is refreshed within a given maintenance window. However, Dynamat approach does not use any framework to detect common views between queries for reuse purpose.

The dynamic data warehouse design is modeled as search space problem in [11]. Rules for pruning the search space are proposed. The first rule relies on favoring the query rewriting that uses views already materialized. The second one modifies the previous rule to favor common sub-expression. However, the proposed view selection algorithm is still in exponential time. Besides, neither implementation nor evaluation of the method have been performed.

Another dynamic approach presented in [10] provides a solution for materializing the indexes which can be seen as a special case of the materialized views. The authors introduce a novel self-tuning framework that continuously monitors the incoming queries and adjusts the system configuration in order to maximize query performance. This approach doesn't react to temporary variations of the query distribution but focus on real changes of the workload. Comparing with our design, we have achieved this feature simply by taking into account the frequency of the view to compute its goodness when taking the decision of materializing it.

6.2 Caching in Distributed Database Systems

An approach, called cache investment is proposed in [5] for integrating query optimization and data placement that looks beyond the performance of a single query. The goal of this study is to place copies of data closest to where they will most likely be accessed. The authors demonstrate that there is circular dependency between caching and query plan optimization which has significant performance implications for advanced distributed database systems. The main difference is that in our approach it is not necessary to materialize the view in the same site where the query has been posed, because we take into account the cooperation between the sites to avoid wasting already materialized as much as possible. However, this point is ignored in this paper [5] as they assume the client-server architecture without any cooperation between the clients in order to invest their cache together.

6.3 Caching and Data Placement in P2P Context

The caching system presented in [4] addresses the problem of PeerOLAP architecture where a large number of peers access sporadically a number of separate data warehouses for posing On-Line Analytical Processing queries. When any query arrives at a peer p, first it is decomposed into chunks. Then, peer p tries to find some chunks locally and will broadcast a demand of the missing chunks to its neighbors. We can see from this scenario that a lot of messages would pass through the network in order to find the answer of a query. In our design we avoid this problem by using the materialized information stored as an entire view and employing the cluster-peer at each cluster which keeps a repository of the entire materialized view at the peers of the cluster.

In Piazza [3], each peer can have any of the following four roles: data origin which provides the original content, storage provider which stores materialized views, query evaluator which uses its CPU resources to evaluate a query and query initiator which poses new queries to the system. Piazza deals primarily with the data placement problem which is to distribute data and work so the full query workload is answered with lowest cost under the existing resource and bandwidth constraints. In Piazza, the data placement problem is solved by separating logically the system into smaller spheres of cooperation and advertising the set of materialized views to all the nodes of a sphere. All peer nodes that belong to the same spheres pool their resources and make cooperative decisions. In our design we avoid this advertisement by employing the cluster-peer at the cluster "sphere" so each peer has to send its catalogue only to its cluster-peer.

7 Conclusion and Future Work

In this paper, we propose a method, which decides dynamically for a given query which views, called candidates, are worthwhile to be materialized taking into account query processing cost, maintenance cost and network transfer cost. We solved the data placement problem by designing two policies: the Isolated and Voluntary policies. Each policy provides a dynamic approach for placing each candidate view. The Isolated policy was motivated by replacement policies for storage management at a peer that records the highest frequency usage of a candidate view. The Voluntary policy was designed to approximate an ideal policy by trying to make the loss of replacing the already materialized views to be minimal. This policy applies the replacement policy for all the peers within a cluster then chooses to materialize the candidate views at the peer that provides the highest benefit.

References

1. Baril, X., Bellahsene, Z.: Selection of materialized views: a cost-based approach. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681. Springer, Heidelberg (2003)
2. Bellahsene, Z., Roantree, M.: Querying distributed data in a super-peer based architecture. In: Galindo, F., Takizawa, M., Traunmüller, R. (eds.) DEXA 2004. LNCS, vol. 3180, pp. 296–305. Springer, Heidelberg (2004)
3. Gribble, S., Halevy, A., Ives, Z., Rodrig, M., Suciu, D.: What Can Databases do for Peer-to-Peer. In: WebDB Workshop on Databases and the Web (June 2001)
4. Kalnis, P., Ng, W.S., Ooi, B.C., Papadias, D., Tan, K.: An Adaptive Peer-to-Peer Network for Distributed Caching of OLAP Results. In: Proceeding of the International Conference on Management of Data, SIGMOD, Madison, WI, pp. 25–36 (2002)
5. Kossmann, D.: The State of the art in distributed query processingt. Journal of ACM Computing Surveys 32(4), 422–469 (2000)
6. Kotidis, Y., Roussopoulos, N.: DynaMat: A Dynamic View Management System for Data Warehouses. In: Proceeding of the International Conference on Management of Data, SIGMOD, Philadelphia, USA (1999)
7. Kotidis, Y., Roussopoulos, N.: A case for dynamic view management. Journal of ACM Trans. Database Syst 26(4), 388–423 (2001)
8. Roussopoulos, N.: Materialized Views and Data Warehouses. ACM SIGMOD Record 27(1) (March 1998)
9. Roy, P., Seshadri, S., Sudarshan, S., Siddhesh, B.: Efficient and Extensible Algorithms for Multiquery Optimization. In: Proceeding of the International Conference on Management of Data, SIGMOD, San Diego, USA (2000)
10. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: Colt -continuous online database tuning. In: Proceeding of the International Conference on Management of Data, SIGMOD (2006)
11. Theodoratos, D., Sellis, T.: Incremental Design. Journal of Intelligent Information Systems 15, 7–27 (2000)