OLAP Query Processing in a Database Cluster*

Alexandre A.B. Lima[1], Marta Mattoso[1], and Patrick Valduriez[2]

[1]Computer Science Department, COPPE, Federal University of Rio de Janeiro, Brazil
{assis,marta}@cos.ufrj.br
[2]Atlas Group, INRIA and LINA, University of Nantes, France
Patrick.Valduriez@inria.fr

Abstract. The efficient execution of OLAP queries, which are typically read-only and heavy-weight, is a hard problem which has been traditionally solved using tightly-coupled multiprocessors. Considering a database cluster as a cost-effective alternative, we propose an efficient, yet simple, solution, called fined-grained virtual partitioning to OLAP parallel query processing. We designed this solution for a shared-nothing database cluster architecture that can scale up to very large configurations and support black-box DBMS using non intrusive, simple techniques. To validate our solution, we implemented a Java prototype on a 16 node cluster system and ran experiments with typical queries of the TPC-H benchmark. The results show that our solution yields linear, and sometimes super-linear, speedup. With 16 nodes, it outperforms traditional virtual partitioning by a factor of 6.

1 Introduction

Decision support applications require efficient support for On-Line Analytical Processing (OLAP) on larger and larger databases. OLAP queries are typically read-only and heavy-weight. In the TPC-H benchmark [12], specific to decision support systems, twenty-two database queries are complex, heavy-weight and read-only and only two have updates. Furthermore, OLAP queries have an ad-hoc nature. As users get more experienced about OLAP system features, they demand more efficient ad-hoc query support [5].

The efficient execution of OLAP queries, where “efficiency” means “as fast as possible”, is still an open problem. High-performance of database management has been traditionally achieved with parallel database systems [13], implemented on tightly-coupled multiprocessors. Parallel data processing is then obtained by partitioning and replicating the data across the multiprocessor nodes in order to divide processing. Although quite effective, this solution requires the database system to have full control over the data and is expensive in terms of software and hardware. Clusters of PC servers provide a cost-effective alternative to tightly-coupled multiprocessors. Recently, the *database cluster* approach, i.e. clusters with off-the-shelf (black-box) DBMS nodes, has gained much interest for various database applications [1, 4, 9].

In this paper, we propose a solution to efficient OLAP query processing in a database cluster using simple parallel processing techniques. The basic technique we

* Work partially funded by CNPq, CAPES, INRIA and COFECUB.

employ is virtual partitioning [1] which gives more flexibility than physical (static) data partitioning [7] for parallel query processing. In its simplest form, it consists in fully replicating the database among the cluster nodes. To distribute the workload, predicates are added to queries to force each DBMS to process a different subset, called a *virtual partition*, of data items. Each DBMS processes exactly one sub-query. The problem is these sub-queries can take almost as long as the original query to be executed. Depending on the estimated amount of data to be processed, DBMS optimizers can opt for fully scanning the virtually partitioned table, reducing (or even eliminating) benefits obtained from virtual partitioning. Temporary disk-based structures demanded by sub-queries that deal with huge amounts of data can also limit virtual partitioning performance. In this paper, we propose a major improvement called *fine-grained virtual partitioning* (FGVP) which addresses these problems. As proposed in [1], virtual partitioning assigns each cluster node one sub-query, what can lead to problems aforementioned. Our approach is to work with a larger number of virtual partitions, much greater than the number of nodes. It is an attempt to keep sub-queries as simple as possible, avoid full table scans and expensive temporary disk-based structures. Our experimental results, based on our implementation on a 16-node cluster running PostgreSQL, show that linear, and sometimes super-linear, speedup is obtained for typical OLAP queries. In the worst cases, almost linear speedup is achieved. FGVP outperformed the traditional virtual partitioning for all queries when using more than two nodes. We think FGVP also provides a good basis for dynamic load balancing as it makes it possible to perform sub-query reallocation. This article is organized as follows. Section 2 presents our database cluster architecture. Section 3 describes our fine-grained virtual partitioning technique. Section 4 describes our prototype implementation as well as experimental results. Section 5 concludes.

2 Database Cluster Architecture

A database cluster [1] is a set of PC servers interconnected by a dedicated high-speed network, each one having its own processor(s) and hard disk(s), and running an off-the-shelf DBMS. Similar to multiprocessors, various cluster system architectures are possible: shared-disk, shared-cache and shared-nothing [13]. Shared-disk and shared-cache require a special interconnect that provides a shared space to all nodes with provision for cache coherence using either hardware or software. Shared-nothing (or distributed memory) is the only architecture that does not incur the additional cost of a special interconnect. Furthermore, shared-nothing can scale up to very large configurations. Thus, we strive to exploit a shared-nothing architecture as in PowerDB [11] and Leg@Net [4]. Each cluster node can simply run an inexpensive (non parallel) DBMS. In our case, we use the PostgreSQL [8] DBMS, which is freeware. Furthermore, the DBMS is used as a "black-box" component [4]. In other words, its source code is not available and cannot be changed or extended to be "cluster-aware". Therefore, extra functionality like parallel query processing capabilities must be implemented via middleware.

We use data replication to improve performance. As in [1, 4], we assume full database replication for simplicity: each database is replicated at each node. To maintain copy consistency, we can assume a preventive replication protocol [2] which scales up well in cluster systems. But since database updates are rare in OLAP applications, copy consistency is not an issue. The only potential problem with full database replication is database size, which can be huge. A good solution is to have a mix of data replication and data partitioning as in [9]. Our technique can be employed in such configurations.

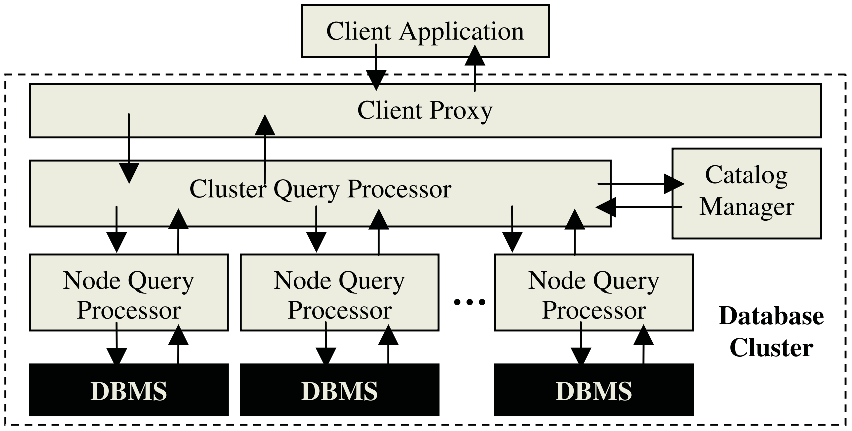

Fig. 1. Database Cluster Architecture

In our system architecture, query processing is done by independent DBMSs orchestrated by distributed middleware. Our middleware main components are Client Proxy, Cluster Query Processor, Catalog Manager and Node Query Processor (see Figure 1). The Client Proxy is the system entry point. It is used by client applications for query submission. Upon reception, the Client Proxy passes the query on to the Cluster Query Processor (CQP) which is responsible for elaboration and execution of Query Execution Plans (QEP). Elaborating a QEP means determining the virtual partitioning attributes, the number of virtual partitions and the participating nodes. For QEP elaboration, metadata information like schema definition and database statistics is obtained from the Catalog Manager. When the QEP is ready, the CQP starts the threads responsible for distributed query execution management and final result composition. This process involves interaction with the Node Query Processors. By definition, each node in a database cluster runs a DBMS instance [1]. In our architecture, each node has also a Node Query Processor (NQP) which is responsible for node-level query execution and result composition. After receiving its workload from the CQP, the NQP interacts with the local DBMS, submitting SQL queries and collecting their corresponding results. These results are locally computed and sent to the CQP which does global result composition. When all NQPs are finished, the CQP sends the final results to the Client Proxy for delivery to the client application.

3 Dynamic Virtual Partitioning

In this section, we introduce the principle of virtual partitioning and describe our proposal of fine-grained virtual partitioning.

Assuming a fully replicated database, virtual partitioning adds predicates to queries to force each DBMS to process a different subset, called a *virtual partition*, of data items. For example, let us consider query Q1 which accesses the Lineitem relation, the largest one from TPC-H:

```
Q1:    SELECT l_returnflag, l_linestatus, SUM(l_quantity), COUNT(*)
       FROM    lineitem
       WHERE  l_shipdate <= date '1998-12-01' - interval '90 day'
       GROUP BY l_returnflag, l_linestatus;
```

Q1 is a typical OLAP query. It has an aggregate and accesses a huge table (in TPC-H's smallest configuration, Lineitem has 6,001,215 tuples). The select predicate is not very selective since there are 5,916,519 tuples that satisfy it. For simplicity, it contains no joins. Using virtual partitioning, this query would be rewritten by adding the predicate "and l_orderkey >= :v1 and l_orderkey < :v2" to the "where" clause of Q1. The rewritten query can then be submitted to the nodes that participate in Q1 processing. Lineitem's primary key is formed by the attributes l_orderkey and l_linenumber. Since there is a clustered index on the primary key and l_orderkey has a large range of values, l_orderkey has been chosen as the virtual partitioning attribute. Each node receives the same query, but with different values for *v1* and *v2*, so that the whole range of l_orderkey is scanned. This technique allows great flexibility for node allocation during query processing: any set of nodes in the cluster can be chosen for executing any query.

One basic goal of virtual partitioning is to reduce the amount of data read from disk by each DBMS. Clustered indexes play an important rule as, by using them, each DBMS can work on a different subset of disk pages. However, the existence of such indexes based on attributes used for virtual partitioning does not guarantee their use during query processing. DBMS query optimizers decide using them or not according to estimations on the amount of data to be retrieved, which depends on the attribute value range specified in each sub-query. Incidentally, the optimizer can even opt for performing a full scan on the virtually partitioned table. We experienced this when performing experiments with virtual partitioning and PostgreSQL.

To overcome this problem, we propose an optimization technique called "fine-grained virtual partitioning" (FGVP). Instead of assigning one sub-query per node (as in [1]), our approach is to produce an initial number of virtual partitions greater than the number of participating cluster nodes. For example, if four cluster nodes are chosen to participate in a query processing we could produce sixty-four virtual partitions, generating sixty-four sub-queries. Then, sixteen sub-queries would be submitted to each node. The number of initial partitions should be much greater than the number of participating nodes, each partition corresponding to a small range of data items. We believe small sub-queries can bring many improvements to traditional virtual partitioning. They make it possible to avoid full scans on the virtually partitioned table. Besides, some queries demand temporary structures to store data while being processed. According to the amount of data, disk resources have to be employed. Small queries can exclusively use main memory structures.

In [10], small physical database fragments are also used for OLAP query processing with good results. A data fragmentation technique to be applied on fact tables is proposed, called multi-dimensional hierarchical fragmentation (MDHF). Its success solely depends on a good fragmentation. FGVP has the advantage of not requiring physical data fragmentation, facilitating the migration of applications from sequential environments, as in [4].

FGVP can provide a good basis for the introduction of dynamic load balancing in virtual partitioning. As proposed in [1], traditional virtual partitioning assigns each node exactly one sub-query. Normally, when a DBMS starts executing a query, it is not possible to externally stop it, modify the query and resume execution. So, there is no room for workload redistribution. Working with more sub-queries than cluster nodes, as FGVP proposes, could make sub-query reallocation possible, easing load balancing. The more partitions we have, the more opportunities to perform dynamic optimization.

Obviously, if there are too many virtual partitions, performance can degrade as more inter-process communication would become necessary. The problem is thus to determine the number of virtual partitions. By now, we use a static approach based on database statistics and DBMS-specific information, like the threshold after which the DBMS starts performing full table scans instead of using clustered indexes. As our goal is just to investigate the general behavior of FGVP, we employ this simple approach by now in spite of not thinking it is appropriate for cluster databases with black-box DBMSs. Alternatively, as database clusters and multi-database systems [7] are similar in many aspects, one can reuse techniques of predicting query execution costs in such environments [3, 6, 14].

4 Experimental Validation

To validate our solution, we implemented a prototype on a cluster system and ran experiments with the TPC-H benchmark. The cluster system has 16 nodes, each with 2 Pentium III 1 GHz processors, 512 Mb main memory, and a disk capacity of 40 Gb. Cluster nodes are interconnected through a 1 Gb/s Myrinet network. We use the PostgreSQL 7.3.4 DBMS running on Linux Mandrake 8.0. We generated the TPC-H database as specified in [12] with a database size of 1.96 Gb. The fact tables (orders and lineitem) have clustered indexes on their primary keys. We also built indexes for all foreign keys. As our goal is to deal with *ad-hoc* queries, no other optimization was performed. The database is replicated at each cluster node.

Our prototype is implemented in Java and some components like the Cluster Query Processor (CQP) and parts of the Node Query Processor (NQP) are implemented as Java RMI objects. Our implementation exploits multi-threading. Each query is processed by a different thread of the CQP and NQP. Result composition is done in parallel at each NQP. Only the final global result composition is done by CQP in one of the participating nodes. To maximize system throughput and avoid bottlenecks, sub-query submission and result composition are processed by separate threads.

In our experiments, each query was run several times. To ease result presentation (Figures 2(b), 3(a) and 3(b)), we normalized their mean response time, by dividing each mean response time by the greatest response time of its associated query. Response time was measured by the client application from the moment it submitted the query till the moment it received the final result. We use TPC-H queries Q1, Q12 and Q14, each corresponding to a different OLAP query type according to the classification in [2]. Q1 accesses only the largest fact table and performs many aggregate operations. Q12 accesses both fact tables with a join between them. Q14 performs a join between the largest fact table and one dimension table. We concentrate on those three queries because they are most representative of OLAP applications.

The partitioning is as follows. For Q1, it is based on l_orderkey since it is the first primary key attribute of lineitem table and has few tuples for each value. For Q12, we use primary virtual fragmentation for the orders table based on primary key o_orderkey and derived virtual fragmentation for the lineitem table based on foreign key l_orderkey. For Q14 which accesses only one fact table (lineitem), we employ the same strategy as for Q1.

Figure 2 (a) shows the response time improvement of fine-grained virtual partitioning (FGVP) over traditional virtual partitioning (VP) varying the number of nodes. With one node, VP performs like sequential execution because there is only one partition to deal with. In this case, Q12 and Q14 yield better performance with VP than with FGVP. By analyzing the query plans generated by PostgreSQL, we observed that lineitem table is fully scanned in VP while an index is used in FGVP. Each sub-query generated by FGVP accesses only a small number of tuples. So, PostgreSQL decides to execute each one using an index as it is not aware that each sub-query is part of a larger query. For both strategies, all lineitem tuples need to be processed since there is no index on the predicate attributes of the initial query. With a single node, a full scan is more efficient in this case. Q1 performs better with FGVP than with VP even when only one node is used. This is because it consumes more CPU resources than Q12 and Q14 as it has many aggregate operations. In our prototype, we implemented modules for collecting intermediate results (from the intermediate queries) and performing the required aggregations. With two CPUs per node, query processing (performed by DBMS) and partial result aggregation (performed by our middleware) can be parallelized. By producing a large number of sub-queries, FGVP takes full advantage of this characteristic. On the other hand, as VP produces only one sub-query per node, it does not benefit too much from it. Then, FGVP performs better. With two nodes, only Q12 still yields better performance with VP. However, Q12 and Q14 start outperforming with FGVP. From 4 to 16 nodes, FGVP outperforms VP significantly, yielding an improvement factor of 6 for Q12 and Q14 with 16 nodes. This is due to different query plans generated for sub-queries produced by each technique. For instance, let us analyze the 16-node case. For Q1, VP produces a full scan of the fact table which is processed by all nodes, thus making difficult to benefit from the parallel execution. With FGVP, the table is accessed through an index and small intermediate results are generated. For Q12, VP produces a merge join algorithm [7] while FGVP produces a fast main-memory nested loop join (thus avoiding I/O operations) because the partitions are small. For Q14, VP produces full scans of the lineitem table for each (large size) virtual partition while FGVP uses an index to access the lineitem table, thus reducing response time.

Figure 2 (b) shows how FGVP scales up with the number of nodes. Q1 and Q12 have almost linear speedup. Q14 has slightly smaller speedup. In Q1 and Q12, all tables are accessed through clustered indexes while in Q14, the part table (200,000 tuples) is accessed through a non-clustered index which is less efficient.

Figure 3 shows the performance of FGVP with different numbers of partitions with 8 and 16 nodes. We observe that the best number (among those that are being showed) varies from query to query. Incidentally in these experiments, this number is the same for both configurations when we consider the same queries. Other experiments not described here show variations for the same query according to the number of nodes employed. This shows the importance of a good partition size estimation and dynamic adjustment.

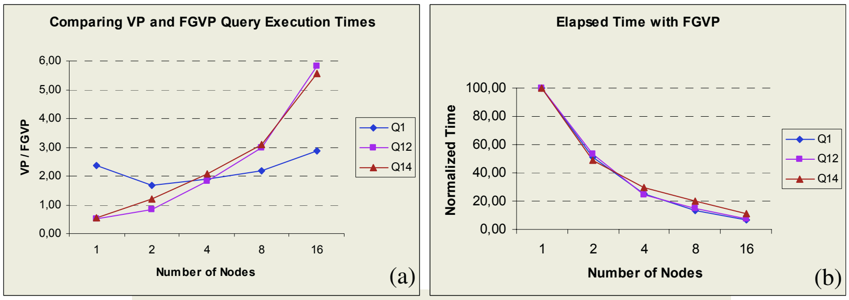

Fig. 2. Performance of VP versus FGVP

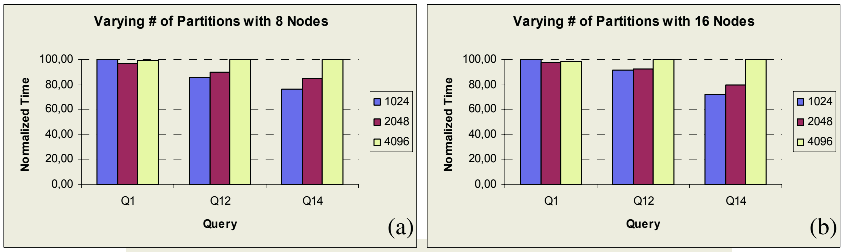

Fig. 3. FGVP with varying numbers of partitions

5 Conclusion and Future Work

In this paper, we proposed an efficient solution, called fined-grained virtual partitioning (FGVP), to OLAP parallel query processing in a database cluster. The idea behind FGVP is conceptually simple. Assuming replication of the database among the cluster nodes, queries are rewritten to deal with virtual partitions of the database. Unlike traditional virtual partitioning (VP), FGVP produces a number of sub-queries much larger than the number of nodes employed for query processing. Consequently, more light-weight sub-queries are generated, avoiding full table scans and expensive temporary disk-based structures. FGVP is a significant improvement over static virtual partitioning (VP). FGVP can also work with partially replicated databases.

Our database cluster has a shared-nothing architecture to provide for scale up to very large configurations. It supports black-box DBMS using non intrusive, simple techniques implemented by Java middleware. Thus, it can support any kind of relational DBMS. To validate our solution, we implemented a Java prototype on a 16-node cluster system and ran experiments with typical queries of the TPC-H benchmark. The results show that FGVP yields linear, and sometimes super-linear, speedup. With 16 nodes, FGVP outperforms VP by a factor of 6 which is excellent.

As a next step, we intend to introduce dynamic load balancing in FGVP. We also intend to investigate a DBMS-independent approach to calculate the number of sub-queries produced by FGVP.

Acknowledgements

Alexandre A.B. Lima would like to thank CAPES and Universidade do Grande Rio - Professor José de Souza Herdy for their support while visiting the University of Nantes during his doctoral studies.

References

1. Akal F., Böhm K., Schek H.-J., OLAP Query Evaluation in a Database Cluster: a Performance Study on Intra-Query Parallelism, *East-European Conf. on Advances in Databases and Information Systems (ADBIS),* Bratislava, Slovakia, 2002.
2. Coulon C., Pacitti E., Valduriez P., Scaling up the Preventive Replication of Autonomous Databases in Cluster Systems, *VecPar Int. Conf.*, Valencia, 2004 (to appear).
3. Du W., Krishnamurthy R., Shan M.C., Query Optimisation in a Heterogeneous DBMS, *Int. Conf. on Very Large Data Bases*, 1992.
4. Gançarski, S., Naacke, H., Pacitti, E., Valduriez, P., Parallel Processing with Autonomous Databases in a Cluster System, *Int. Conf. on Cooperative Information Systems*, 2002.
5. Gorla N., Features to Consider in a Data Warehousing System, *CACM* 46(11), 2003.
6. Lu H., Ooi B., Goh C., On Global Multidatabase Query Optimization, *ACM SIGMOD Rec*. 21(4): 6-11, 1992.
7. Özsu T., Valduriez P., *Principles of Distributed Database Systems*, Prentice Hall, 1999.
8. PostgreSQL, http://www.postgres.org
9. Röhm U., Böhm K., Schek H.-J., OLAP Query Routing and Physical Design in a Database Cluster, *Int. Conf. on Extending Database Technology (EDBT),* 2000.
10. Stöhr, T., Märtens, H., Rahm, E., Multi-Dimensional Database Allocation for Parallel Data Warehouses, *Proc. 26th Int. Conf. on Very Large Databases*, Cairo, Egypt, 2000.
11. The Project PowerDB, http://www.dbs.ethz.ch/~powerdb
12. TPC, TPC Benchmark™ H (Decision Support)), http://www.tpc.org/tpch
13. Valduriez P., Parallel Database Systems: open problems and new issues. *Int. Journal on Distributed and Parallel Databases*, 1(2), 1993.
14. Zhu Q., Larson P.A., A Query Sampling Method of Estimating Local Cost Parameters in a Multidatabase System, *Int. Conf. on Data Engineering (ICDE),* Houston, Texas, 1994.