The OLAP-Enabled Grid: Model and Query Processing Algorithms

Michael Lawrence Andrew Rau-Chaplin
Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada B3H 1W5
{michaell,arc}@cs.dal.ca
www.cgmLab.org

Abstract

The operation of modern distributed enterprises, be they commercial, scientific, or health related, generate massive quantities of data. Decision makers increasingly utilize On-Line Analytical Processing (OLAP) tools to glean from this rich data resource nuggets of information which can be used to better run their enterprises. A typical approach to OLAP is to construct a single centralized data repository by copying all of the raw data from the sites where it is generated to a cental location, where it is integrated, and then to route all queries to that central location. As the amount of data and number of sites and users grows this approach suffers from significant scalability problems.

In this paper, we present a model and algorithmic framework for an "OLAP-Enabled Grid" whose goal is the efficient support of OLAP operations. We show how a Grid computing infrastructure can be used to store and manage expensive to compute data aggregations and to answer OLAP queries in a fully distributed manner. Our focus is on the efficient optimization of resources for answering queries based on a distributed query algorithm which uses cached and pre-aggregated data stored over a Grid computing infrastructure.

1. Introduction

Many enterprises maintain sets of physically distributed operational databases which contain extremely large amounts of data. For example, the sales transactions of a retail chain, patient records of a group of hospitals, or data from high-energy physics experiments. Decision makers and analysts often turn to On-Line Analytical Processing (OLAP) tools to extract useful information about the underlying trends in their data. With the proliferation of networking technology supporting long distance collaboration, many of these enterprises are able to operate in a distributed fashion. The current approach to constructing a data warehouse is to copy the raw data from the various distributed operational databases and to move it to a central site where it can then be processed to form a single large centralized repository. As the amount of data and number of sites grows this "centralized" approach suffers from significant scalability problems.

At the same time, enterprises are adopting Grid computing as a way of harnessing geographically distributed computing resources. Grids, widely distributed collections of heterogeneous computers whose resources are pooled together for a specific task, are becoming an attractive platform for high performance and data intensive scientific and commercial computing due to their philosophy of efficient utilization of existing computational infrastructure. In this paper we present the OLAP-Enabled Grid, a new approach to data warehousing and OLAP which, rather than integrating the operational data into a single data warehouse, provides a virtual data warehouse to its users by transparently supporting OLAP operations through the use of resources in a Grid.

Our OLAP-Enabled Grid model arises from the scenario that the data of a single organization is distributed across a number of operational databases at remote locations. Each operational database has capabilities for answering OLAP queries, and access to a possible variety of other computational and storage resources which are located close by. The organization may be a single enterprise, as in the case of a chain of retail stores maintaining point of sale transactions at its various outlets, or the union of a number of collaborating organizations, as in the case of scientific research groups collaborating to form a data Grid. There are a number of users who are interested in doing OLAP on these databases, viewing them as a single integrated data warehouse. The users are distributed over the network, and may enter and leave the network in an ad-hoc manner. The work presented in this paper gives a precise modeling of the interaction between the entities in such a Grid, and proposes

Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)
0-7695-2582-2/06 $20.00 © 2006 IEEE

algorithms for answering queries using cached query results and materialized views.

The remainder of this paper is organized as follows: Section 2 presents the OLAP-Enabled Grid, followed by distributed query processing strategies in Section 3. A review of related work in the area of distributed OLAP and Grid computing is given in Section 4, followed by a summary and brief discussion of future work in Section 5.

2. The OLAP-Enabled Grid

As described in the introduction, the OLAP-Enabled Grid considers an organization which has a number of widely distributed sites, each site being a local network and containing some number of entities. An example is shown in Figure 1. The entities are computers having distinct roles in the OLAP process. Each site is under control of a part of the organization and maintains some number of data warehouses, for example a department of a large enterprise, research lab or retail outlet.

We group entities in the OLAP-Enabled Grid according to five different roles, however we note that there may exist entities in the Grid which do not participate in the OLAP process at all.

1. *OLAP Server* - A machine which has sole control over an operational database. It may maintain some materialized views and may also act as a computational or storage resource. The OLAP servers all have the same schema, but each maintains a partition of the total data available to the users.

2. *Computational Resource* - A machine which offers cycles for performing tasks on the behalf of other entities in the Grid.

3. *Storage Resource* - A machine which offers disk space for storing data on behalf of other entities in the Grid.

4. *Resource Optimizer* - There is exactly one resource optimizer for each site. A resource optimizer has the information necessary to perform scheduling and allocation of computational and storage resources, and to carry out queries. It may also have some cache space for storing common query results for queries generated in its site.

5. *User* - Users submit ad-hoc queries to resource optimizers and may enter and leave the network at will. Each user has an amount of cache space for caching query results.

An analogous model in the peer-to-peer (P2P) setting was described in [1]. It is similar in that peers can take on the role of users, OLAP servers or storage resources, however the P2P setting is different in that there is no centralized control (where as Grids may have specialized servers managing subsets of the infrastructure), and that requests are routed through the logical connections between peers.

As shown in Figure 1, in the proposed model each site can have any number of OLAP servers which may or may not donate storage or computational resources. In Figure 1, the amount of computational resources an entity has is shown by the number of rows in its rack, while the amount of storage resources are indicated by the size of the attached disk. There is one resource optimizer at each site, whose responsibility is to keep an updated catalogue of the available resources (computational, storage, data, and cache) in the Grid, and use this information to help schedule computations, allocate storage resources, and answer queries in a distributed fashion. Having described our basic model, in the next sections we focus on the efficient optimization of resources for answering queries based on a distributed query algorithm which uses cached and pre-aggregated data.

3. Query Processing

To answer queries in the OLAP-Enabled Grid we propose a two-tiered process. The first tier is using the caches on the local site in a cooperative manner to answer as much as the query as possible, and the second tier is requesting the missing fragments from the OLAP servers. This two-tiered process is necessarily dependent on our choice of location for cached query results and materialized views. We allow user caches to store any query results which may be of interest, which in general are multi-dimensional view fragments spanning multiple OLAP data warehouses. Pre-aggregated views of an OLAP data warehouse are stored on the storage resources of the site local to that data warehouse, i.e. we do not consider pushing materialized views out to other sites. The reason for this is that by making caches the first source for query results (as done in [2]), the caches in a site will tend to be saturated with data from the most frequently requested views in that site. Hence most queries on those views will not make it to the sites which maintain those views. As a result, the most frequently requested views as seen by an OLAP server are not necessarily in high demand at any one particular site. They are more likely less oft requested views at each site, so that they do not demand significant cache space at those sites, but for which the aggregate demand over all sites is significant. Thus it does not make sense to push these materialized views out to other sites since they are not frequently requested there.

At each tier, the resource optimizer plays a central role. In the first tier, the user submits a query q to the resource optimizer. Having an index of all fragments cached in the local

Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)
0-7695-2582-2/06 $20.00 © 2006 IEEE

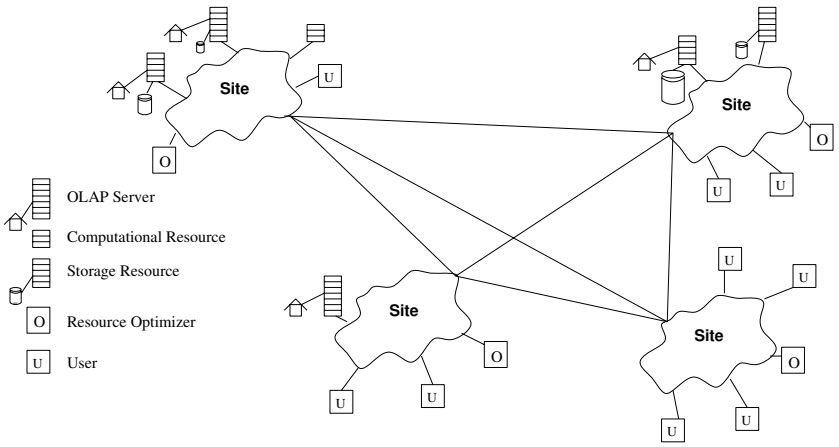

Figure 1. An example OLAP-Enabled Grid.

site, the resource optimizer locates all relevant fragments which can be used to compute part of q, and formulates a set of sub-queries $Q = \{q_1, q_2 \dots q_n\}$ for the remaining fragments. Each of these queries q_i is sent on behalf of the user to the resource optimizer at each site having an OLAP server whose fact table intersects with q_i. In tier two query answering, a resource optimizer receives a query q to be answered using a data warehouse on its site. It locates all OLAP servers and storage resources at this site having materialized views which can answer q and chooses the one which can answer it the quickest based on a cost modeling. The results are sent back to the user who requested them. This process is outlined in Algorithm 1.

Algorithm 1 Two-Tiered Query Answering Overview

1: User U sends query q to its local resource optimizer R
2: R uses its index on the locally cached query fragments to locate which ones can be used to partially answer q, and formulates a set of sub-queries Q for the missing portions.
3: The local users send their relevant cached fragments to U.
4: **for all** $q_i \in Q$ **do**
5: **for all** OLAP Servers S potentially containing results for q_i **do**
6: Send q_i to S's local resource optimizer R_S.
7: R_S answers q_i by sending it to S or the local computational resource which can answer it the quickest.
8: The answer to q_i from S is sent back to U.
9: **end for**
10: **end for**
11: Once U has received all of the results it needs, it combines them and performs any final aggregation necessary to get the final answer to q.

One obvious question is: why search the local caches before the local OLAP servers? Since they are likely much better equipped to answer queries than the user machines doing the caching. The reason is that a well designed caching strategy will not cache data from local OLAP servers, since it is much easier to get and as a result much less valuable than data from remote OLAP servers. Hence by searching the caches first we are not necessarily favouring cached data over local OLAP servers.

3.1. Tier 1 - Cooperatively Using Local Caches

We index caches by using an R-Tree for each view in the data cube lattice which indexes the fragments of that view which are stored in the caches. Each fragment is also annotated with the number of rows it contains, as well as the address of the machine (user or gateway cache) which contains the fragment. When a resource optimizer O is given a query q it searches the R-Tree at the view v corresponding to q's level of aggregation, and identifies all intersecting view fragments. Based on the intersecting regions of the identified fragments, it computes a set of sub-queries Q which request the missing fragments. It then searches up the data cube lattice beginning with the ancestors of v, and at each view w re-writes the queries in Q as queries on w and attempts to answer them recursively, which in turn makes other recursive calls. Any of the queries in Q which cannot be completely answered by the recursive procedure is sent to all OLAP servers whose bounding boxes overlap with the query. All of the results, both cached and from the sub-queries, are returned to the user which combines them into the result set for q.

The algorithm which computes the set of missing view fragments (queries) Q given a query q and a set of intersecting view fragments Q^I is described as follows. Q is

Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)
0-7695-2582-2/06 $20.00 © 2006 IEEE

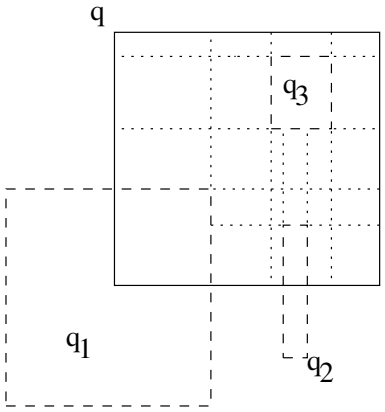

Figure 2. A multi-dimensional range query q (solid lines) given to a resource optimizer, which finds the intersecting queries q_1, q_2, and q_3 (dashed lines) in the cache. The set of fragments Q which remain to be answered is initialized to those regions shown which have at least one dotted line side, and the remaining sides dotted, dashed and solid lines.

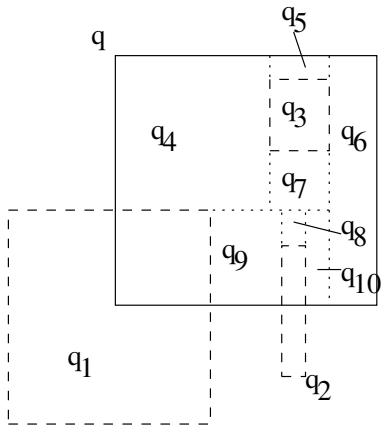

Figure 3. A multi-dimensional range query q (solid lines) given to a resource optimizer, which finds the intersecting queries q_1, q_2, and q_3 (dashed lines) in the cache. The final set of fragments Q which will be either answered by cached fragments of views further up in the data cube lattice or sent to the OLAP servers is $\{q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}$.

initialized by drawing a grid of unequally sized cells on q by extending the boundaries of each query in Q^I through q. For example, in Figure 2, $Q^I = \{q_1, q_2, q_3\}$ is drawn with dashed lines, and the grid imposed on q (the solid-line box) is shown with dotted lines. The queries in Q are initialized to those boxes which have at least one side as a dotted line, and all other sides dotted, dashed, or solid lines (i.e. all regions not covered by q or queries in Q^I). The algorithm then merges adjacent pairs of fragments in Q which have the same dimensions on the adjacent hyperplane, until no more such merges are possible. As a heuristic the pair to be merged at each step is chosen to maximize the area of the resulting merged fragment. For example, in Figure 2, the first pair merged is the two fragments directly above q_1, as this results in the largest merged fragment. The final set of fragments in this example is $Q = \{q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}$, shown in Figure 3.

Our basic lattice search procedure, as outlined in Algorithm 2, can be thought of as a view fragment adaptation of Deshpande's Exhaustive Search Method (ESM) [3]. It begins with a query q on a view v, and searches for cached fragments of this view and from aggregating higher level cached fragments to answer as much of q as possible. Its output is a set of fragments Q^I which can be computed from the cache as well as a set of remaining fragments Q which must be sent to the OLAP servers to be answered.

We call our strategy the query fragment bubble-up strategy and Deshpande's [3] the query chunk bubble-up strategy, since requests for sub-fragments or chunks are bubbled-up to ancestor views in the data cube lattice. Our algorithm, like Deshpande's ESM, is naive to the possible overlap between queries at higher level views, a more clever implementation would proceed breadth-first, using a global queue for fragment requests at each view like Deshpande's Virtual Count Method (VCM).

Our procedure will only be beneficial over an all-or-none strategy, where either q is entirely answered from the caches or not at all, if the OLAP servers can efficiently the group of number of small queries faster than a large query which contains all of them. This is certainly reasonable, since reading the records to answer a query is a disk bound activity where the amount to be read (with an appropriate index) is proportional to the size of the query. There are a number of remaining questions about the algorithms described here, for example the worst-case number of fragments in Q found by Find Missing Fragments as a function of the number of fragments in Q^I, as well as the time complexities of the algorithms.

3.2. Tier 2 - OLAP Server Site Query Answering

The set of fragments Q of a query q which cannot be answered from the caches on the local site must be sent to the OLAP servers to be answered. For each query q' in Q, we form a d-dimensional bounding box B by adding global selection ranges to those dimensions not grouped on

Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)
0-7695-2582-2/06 $20.00 © 2006 IEEE

Algorithm 2 Cache Search

Input: Query q on view v

Output: Set of fragments Q^I of q which are answered from the cache and a set of fragments Q which need to be answered by the OLAP servers.

```
1: Q^I ← all cached fragments intersecting q as found by searching the R-Tree of view v.
2: Q ← FindMissingFragments(q, Q^I).
3: for all q' ∈ Q do
4:    for all Parent views w of v in the lattice do
5:       Re-write q' as a query over w by taking a global selection range on all dimensions in w which are aggregated in v.
6:       Q^I', Q' ← CacheSearch(q', w).
7:       if Q' = ∅ then
8:          Compute q' by aggregating the results in Q^I'.
9:          Remove q' from Q and add it to Q^I'.
10:         Break from inner loop.
11:      end if
12:   end for
13: end for
```

by q. Then to determine which OLAP servers may contain records necessary for answering q', we then search the R-Tree containing the OLAP servers' fact table bounding boxes for intersection with B, and send q' to each matching server S.

An OLAP server S, upon receiving a query q appeals to its local resource optimizer O for determining a cheap way in which to compute q. This depends on materialized views, the power and load on the various computational resources at the site, as well as the load-dependent bandwidth between sites. The query may be answered directly by S itself, or may be answered by another computational and storage resource containing materialized views, or by another computational resource which can receive the necessary data from a storage resource which contains it. Each option is explored, and the cheapest one taken. O figures out how quickly each machine M which is both a storage and computational resource (including S) and contains appropriate materialized views can answer q locally. O also determines the smallest view/index combination stored on a storage resource D (D = "disk"), and computes how quickly q can be computed if this view/index is transferred to the best computational resource at the site. The best computational resource will often be the most powerful, but depending on this machine's load, another less powerful but more available machine may be deemed best.

In determining the cost of answering a query using materialized views, many studies, for example [4, 5, 6, 7, 8, 9, 10, 11], employ a linear cost model where the cost of answering a query q using a materialized view v is equal to the number of records in v. This is irrespective of the size of q. Under this model, it would be pointless to try and partially answer queries from local caches, i.e. if the query cannot be entirely answered from cache, then it should be sent to the OLAP servers. However in practice when index structures are build on the views, the time to answer a query is proportional to the number of records which must be read as well as the cost of aggregating these records to the level that the original query has been posed on, if necessary. In the context of our previous work on the cgmCUBE system [12, 13, 14] the data is indexed using R-Trees so that each leaf points to a disk block and records the bounding box of all rows stored in that disk block. In this case the lookup time using the R-Tree is negligible and the time to answer a query is proportional to the number of records which must be read from disk. This in turn depends on the density of the view which the query is actually answered on, and the size of the re-written query over that view. We express the time to answer a query as a function with two components:

1. A reading time t_r for reading the records from disk
2. And an aggregation time t_a to aggregate the records in memory (which may be 0 in the case that q is answered directly on its view).

The reading time depends on the disk speed and load of the machine it is done on, and the aggregation time depends on the processor speed. Since aggregation is not computationally intensive except for the amounts of data involved, transferring records from one machine to be aggregated on another is more costly than the actual aggregation, and so the reading and aggregation must both be done on the same machine. We abstract the reading time as a function depending on the query and view, as

$$t_r = \ell_d \frac{f_{read}(q, v)}{s_d}$$

where s_d is the disk speed of the machine performing the reading ℓ_d is the load on its disk, q is the query to be answered, v is the view it is answered on, and $f_{read}(q, v)$ is a function giving the number of disk I/Os for reading the records in v which are necessary to compute q. Similarly, we define

$$t_a = \ell_p \frac{f_{aggregation}(q, v)}{s_p}$$

as the aggregation time, where s_p is the processor speed of the machine performing the aggregation, ℓ_p is the load on its processor, and $f_{aggregation}(q, v)$ gives the number of operations (if any) for aggregating the records read from v in order to answer q. Our total time to answer a query is hence

$$t_q = t_r + t_a, \quad (1)$$

Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)
0-7695-2582-2/06 $20.00 © 2006 IEEE

where v is chosen to minimize t_q. Note that this formulation for the sake of simplicity is naive to the fact that these two operations may be pipelined and performed in parallel. In order to make scheduling decisions, the resource optimizer should have knowledge of the processor and disk speeds of all of the computational and storage resources at its site, and can poll the machines to get their respective loads.

Our basic scheduling approach is similar to that of other data-intensive computational Grid schedulers [15, 16, 17] in that it models execution time on the various available resources and chooses the cheapest option. The difference is in the details of the cost modeling as described in the above equations, which are specific to OLAP queries which can be evaluated in many possible ways using different views. In order to answer a query, our scheduler computes the value of Equation 1 for each machine capable of answering the query (storing an appropriate materialized view), and picks the one minimizing it.

Here we have assumed that each materialized view is over a single OLAP server's data only. However, we may potentially make improvements by combining materialized views from two or more OLAP servers if queries on those views frequently range over both servers. A further and easy means for improvement would be to aggregate partial results from each OLAP server before sending them back to the local site. By combining the results for N different OLAP servers at the remote site rather than sending them back independently to be aggregated at the requesting site we could reduce the amount of WAN transmission by $1/N$. We are currently exploring these extensions.

4. Related Work

There has been surprisingly little previous work on OLAP in a Grid environment. A number of researchers have taken a software component oriented approach to the problem, by describing the services and middleware interactions which should exist in order to support OLAP [16, 18, 19]. In [20, 21], Niemi et. al. describe how to answer queries in a distributed manner using a collection server which breaks each query into sub-queries and sends them to the relevant nodes containing the actual data. Although certainly useful, these approaches largely ignore the issues of resource optimization and the technicalities of data and query specification which are necessary for efficiently supporting a virtual data warehouse. Their concern is with the software which makes OLAP in a Grid possible, but not on how to manage the resources in the entire Grid to do it in the most efficient possible manner.

Other studies focus on cost modeling in computational and data Grids but don't directly address OLAP. In [15], incoming data mining tasks are scheduled on single machines so that the task is completed as quickly as possible. In [16], two general formulas are given for modeling the response time of a data-intensive application, one for when the application is executed in the local site, and one at the remote site where the data resides. Our scheduling approach, described in Section 3 is most similar to [17] who consider data-intensive applications executed on one of a number of widely distributed clusters. They provide five formulas for modeling execution time: local data/execution, local data/remote execution, remote data/local execution, remote data and same remote execution, and remote data and different remote execution. Their Chameleon scheduler computes the total time it takes for an application to complete and the output to be transferred to the user for each possibility in the five scenarios, and picks the one which is cheapest.

A major distinction with previous computational and data Grid execution cost modeling and the work presented here is that we consider the data dependency relationship between different OLAP queries. Previous data Grid work is such that each data intensive task requires one and only one data set, where as an OLAP query can be answered from many different sources of pre-aggregated data at different costs. Hence the cost modeling, scheduling and resource allocation techniques must be aware of this. Many queries may require massive amounts of data aggregation, and so the results of these queries are very valuable in terms of re-usability.

The issues of pre-aggregated data have previously been considered in Peer-to-Peer OLAP systems [1, 2]. In [1], the data placement problem for databases in Peer-to-Peer (P2P) networks is considered. In [2] a P2P OLAP system called PeerOLAP is presented which takes a completely decentralized approach by propagating queries to peers through a logical network. Each peer caches query results, and issues queries to its peers hoping to make use of their cached information.

5. Summary and Future Work

In this paper, we have presented a model and algorithmic framework for an "OLAP-Enabled Grid" and shown how a Grid computing infrastructure can be used to store and manage expensive to compute data aggregations and to answer OLAP queries in a fully distributed manner. Our focus has been on the efficient optimization of resources for answering queries based on a distributed query algorithm which uses cached and pre-aggregated data stored over a widely physically distributed Grid computing infrastructure. We are currently constructing an experimental evaluation of the system built on the cgmCUBE code base [13, 14] and investigating a variety of caching and pre-aggregation strategies.

Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)
0-7695-2582-2/06 $20.00 © 2006 IEEE

References

[1] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu, "What can databases do for peer-to-peer?," in *WebDB Workshop*, pp. 31–36, 2001.

[2] 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 '02*, pp. 25–36, 2002.

[3] P. Deshpande and J. F. Naughton, "Aggregate aware caching for multi-dimensional queries," in *EDBT '00: Proceedings of the 7th International Conference on Extending Database Technology*, (London, UK), pp. 167–182, Springer-Verlag, 2000.

[4] H. Gupta and I. S. Mumick, "Selection of views to materialize in a data warehouse," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 24–43, January 2005.

[5] P. Kalnis, N. Mamoulis, and D. Papadias, "View selection using randomized search," *Data Knowl. Eng.*, vol. 42, no. 1, pp. 89–111, 2002.

[6] W. Liang, H. Wang, and M. E. Orlowska, "Materialized view selection under the maintenance time constraint," *Data Knowl. Eng.*, vol. 37, no. 2, pp. 203–216, 2001.

[7] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing data cubes efficiently," in *SIGMOD '96*, pp. 205–216, 1996.

[8] C. Zhang, X. Yao, and J. Yang, "An evolutionary approach to materialized views selection in a datawarehouse environment," *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol. 31, pp. 282–294, August 2001.

[9] H. Gupta and I. S. Mumick, "Selection of views to materialize under a maintenance cost constraint," in *ICDT '99*, pp. 453–470, 1999.

[10] H. Uchiyama, K. Runapongsa, and T. J. Teorey, "A progressive view materialization algorithm," in *Proceedings of the 2nd ACM International Workshop on Data Warehousing and OLAP*, pp. 36–41, 1999.

[11] A. Shukla, P. Deshpande, and J. F. Naughton, "Materialized view selection for multidimensional datasets," in *24rd VLDB Conference*, pp. 488–499, 1998.

[12] F. Dehne, T. Eavis, and A. Rau-Chaplin, "Parallel multi-dimensional ROLAP indexing," in *Proc. IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGrid)*, pp. 86–93, IEEE Comp. Soc. Dig. Library, 2003.

[13] Y. Chen, F. Dehne, T. Eavis, and A. Rau-Chaplin, "Parallel ROLAP data cube construction on shared-nothing multiprocessors," *Distributed and Parallel Databases*, vol. 15, pp. 219–236, 2004.

[14] F. Dehne, T. Eavis, and A. Rau-Chaplin, "The cgmCUBE project: Optimizing parallel data cube generation for ROLAP," *Distributed and Parallel Databases*, to appear.

[15] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "Scheduling high performance data mining tasks on a data grid environment," in *8th International Euro-Par Conference on Parallel Processing*, (London, UK), pp. 375–384, Springer-Verlag, 2002.

[16] H. Stockinger, K. Stockinger, E. Schikuta, and I. Willers, "Towards a cost model for distributed and replicated data stores," in *Proceedings. Ninth Euromicro Workshop on Parallel and Distributed Processing*, (Washington, DC, USA), pp. 461–467, IEEE Computer Society Press, 2001.

[17] S.-M. Park and J.-H. Kim, "Chameleon: a resource scheduler in a data grid environment," in *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid.*, IEEE, May 2003.

[18] B. Fiser, U. Onan, I. Elsayed, P. Brezany, and A. M. Tjoa, "On-line analytical processing on large databases managed by computational grids," in *DEXA '04: International Workshop on Database and Expert Systems Applications*, (Washington, DC, USA), pp. 556–560, IEEE Computer Society, 2004.

[19] W. Dubitzky, D. McCourt, M. Galushka, M. Romberg, and B. Schuller, "Grid-enabled data warehousing for molecular engineering," *Parallel Comput.*, vol. 30, no. 9-10, pp. 1019–1035, 2004.

[20] T. Niemi, M. Niinimki, J. Nummenmaa, and P. Thanisch, "Constructing an olap cube from distributed xml data," in *5th ACM international workshop on Data Warehousing and OLAP*, pp. 22–27, 2002.

[21] T. Niemi, M. Niinimki, J. Nummenmaa, and P. Thanisch, "Applying grid technologies to xml based olap cube construction.," in *Proceedins of the 5th International Workshop in Design and Management of Data Warehouses*, 2003.

Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment (HPCS'06)
0-7695-2582-2/06 $20.00 © 2006 IEEE