On-Line Analytical Processing on Large Databases Managed by Computational Grids[*]

Bernhard Fišer, Umut Onan, Ibrahim Elsayed, Peter Brezany
Institute for Software Science, University of Vienna
Liechtensteinstrasse 22, A-1090 Vienna, Austria
E-mail: {brezany,elsayed,fiser,onan}@par.univie.ac.at

A Min Tjoa
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Favoritenstrasse 9-11/E188, A-1040 Vienna, Austria
E-mail: tjoa@ifs.tuwien.ac.at

Abstract

Management of large data repositories integrated into the Grid poses new challenges for Grid research. There already exist several successful Data Grid projects addressing processing files storing large volumes of scientific data and projects developing services for accessing remote relational and XML databases. However, so far, no effort was devoted to On-Line Analytical Processing (OLAP), an essential support for modern decision support systems. In this paper, we present our approach to the design and implementation of a Grid-enabled OLAP server, which is one functional building block of the GridMiner system, a novel infrastructure for knowledge discovery in Grid databases. We present the global architecture model of our solution, describe how the OLAP components are integrated into the GridMiner system, and present the software architecture of our first prototype. The OLAP components were implemented in Java on top of the software toolkit Globus 3.

1 Introduction

Grid computing has been identified as an important new technology by a remarkable thread of scientific and engineering fields as well as by many commercial and industrial enterprises [10]. Its goal is to virtually share and manage computer resources across enterprises, industry or workgroups independently of the operating characteristics of their computer systems. It can be used to temporarily increase computational power and storage needs on demand and, on the other hand, a system which currently is in idle state can be announced as usable for others. So far, essentially all major Grid projects have been built on protocols and services of the Globus Toolkit (http://www.globus.org/toolkit/), which is an open architecture, open source framework for Grid computing. It consists of several services and software libraries. The latest Globus version, Globus 3, is based on the Open Grid Services Architecture (OGSA) [5].

Many advanced Grid applications are data intensive, that is, significant processing is done on very large amounts of heterogeneous and geographically distributed data. Therefore, the management of data within Grids is a challenging research and development problem. In the past four years, several Data Grid projects have been started ([11]), with the aim of setting up a computational and data-intensive Grid of resources for the processing and analysis of data coming (mainly) from scientific explorations. Almost all these applications are file-based, and until recently, there has been relatively little effort applied to integrating databases into the Grid [12]. The UK core e-Science program's OGSA-Database Access and Integration (DAI) project (http://umbriel.dcs.gla.ac.uk/NeSC/general/) is currently designing and building wrappers for relational and XML repositories so that they offer Grid-enabled services conforming to the OGSA framework.

One important database architecture that has emerged in the traditional database research and development fields in the nineties is the *data warehouse*, a repository of multiple heterogeneous data sources, organized under a unified schema in order to facilitate management of decision making [8]. Data warehouse technology includes data in-

[*] This research is being carried out as part of the research project "Modern Data Analysis on Computational Grids" supported by the Austrian Research Foundation.

tegration and *On-Line Analytical Processing (OLAP)* [2]. OLAP allows interactive analysis of multi-dimensional data of variable granularities with functionalities such as summarization, consolidation, and aggregation, as well as the ability to view information from different angles. This functionality is based on a sophisticated data structure called the *datacube*.

However, so far, to our best knowledge, in the Grid community, no effort has been devoted to the data warehousing and associated technologies. Just these issues are addressed within the *GridMiner* project (www.gridminer.org) in Vienna, Austria. Our recent paper [1] reports about the Grid data integration and virtualization solution, based on Grid data mediation technology, we have developed.

This paper deals with the OLAP functionality of the GridMiner system and is organized as follows. Section 2 provides a brief introduction into OLAP and an overview on the global architecture models we consider and shows how our system fits into the Grid model. In Section 3 we discuss our current OLAP engine, which is based on the sequential model, and in Section 4 we show how this engine is integrated into the Grid. In Section 5, give an overview of our planned future work, which is focused on a more advanced solution based on the parallel and distributed OLAP model. Finally, we provide a brief conclusion in Section 6

2 Background and Architecture Models

On-Line Analytical Processing (OLAP) has become a fundamental component of contemporary decision support systems. In 1995, Gray et al. [7] introduced the *datacube*, a relational operator/model used to compute summary views of data that can, in turn, significantly enhance the response time of core OLAP operations.

There are two standard datacube representations: ROLAP (set of relational tables) and MOLAP (multi-dimensional array). The ROLAP's summary records are stored directly in standard relational tables without any need for data conversion. However, a complex analytical query is cumbersome to express in SQL and it might not be efficient to execute. The array-based model, MOLAP (Multi-dimensional OLAP), has the advantage that native arrays provide an immediate form of indexing for cube queries. Data is stored in a multidimensional structure which is a more natural way to express the multi-dimensionality of the enterprise or scientific data and is more suited for analysis.

OLAP applies aggregation functions to multi-dimensional organized data. Typical aggregation functions are *accumulation* of values, *counting* of the existence of values and finding the *minima* or *maxima* of value sets. The multi-dimensional organized data structure usually is referred to as the *datacube*, the *hypercube* or simply just the *cube*. Each *dimension* of the cube is associated with a specific attribute, this could be the selling date of some product, the color of a product or similar attributes. A specific attribute value within a dimension, for example the product color "green", is called a *position* or an *item*. The value which is functionally dependend on the positions is referred to as the *measure*. This, for example, could be the number of products sold.

Aggregation functions are applied by either reducing the number of dimensions or by reducing the depth of information *hierarchy*, which is the structure within one dimension, which, for example, could be $year \rightarrow quarter \rightarrow month \rightarrow day$. In literature, OLAP operations are usually referred to as *slice-and-dice*, *roll-up*, *drill-down*, etc. In fact, these are subsequent compositions of inter-dimensional and hierarchical aggregation.

We are following the MOLAP approach by virtually building a multi-dimensional cube through sophisticated encoding the positions of the dimensions (index encoding) and using this as a linear address of the measures (because conventional computer memory has one-dimensional organization).

The Grid is an architectural framework to join computer systems of different capabilities and operating systems to merge functionality, computational power and storage availabilities. There are well defined workflows and protocols to ensure communication and there is a framework for database integration, the OGSA-DAI, which is Java-based.

To provide a new functional block for the Grid community, especially for the GridMiner project, which is based on OGSA-DAI and, hence, also Java-based, to reduce interoperability efforts, we build our OLAP service within this framework in Java too.

Performance and usability of the system mostly depends on the global architecture, hence, it has to be chosen thoroughly. Issues influencing the structure are performance and memory considerations as well as communication costs and complexity. Basically we distinguish between several possible solutions.

1. *Centralized*: (a) sequential (We already have a first prototype.); (b) parallel (Implementation is based on message-passing mechanisms, e.g., MPI.).

2. *Geographically distributed*: (a) sequential (This case is similar to 1(a), however, communication costs are much higher.); (b) parallel (A challenging task, which is a part of our research agenda.).

The first approach is to build a centralized sequential engine which, of course, is the most common method, which is usually the most easiest way of implementation compared to the other ones, but it can be used very well to gather information about deeply buried problems, which occur during implementation and to study different algorithms for cube construction and storage, query processing and indexing in

Grid environment. Additionally, it more quickly leads to a result because, as mentioned earlier, the Grid interoperability is also of a high relevance. That's why we first chose to design and implement this type of architecture. We have already a well developed prototype ready and we present the applied design and implementation concepts in following Sections 3 and 4 whose purpose is to introduce the reader into this complex field of data aggregation and querying.

The second kind of architecture is a centralized parallel solution. This is done in a similar way than the first one, but the query processor will partition the query sequence and compute partial aggregates on different hosts. The host intercommunication of this architecture could be based on the Message Passing Interface standard (MPI) (http://www-unix.mcs.anl.gov/mpi/). Some efforts have been done in this area and are discussed in [6]. We'll rely on these methods and conclusions for our further developments.

Another parallel solution could be a geographically distributed architecture, but this is very similar to the centralized parallel solution with the great disadvantage of very high communication costs because of presumably low network bandwidth and high latency. Hence, we'll not follow this approach.

The really most challenging but probably also most reliable architecture is a distributed parallel solution. It consists of at least two geographically distributed sites which within could consist of parallel clustered nodes. Each site appears to be one host and also is addressed by one IP address. Thus communication can be done only between both hosts, but not between a node from one location with a node from the other location. Hence, this is the architecture we plan to implement in the future. We outline our first design concepts for this infrastructure in Section 5.

3 The Sequential OLAP Engine

From a global point of view, our sequential OLAP server currently consists of the data cube structure, an index database and function blocks for cube construction, querying and connection handling (see Figure 1).

The data cube structure consists of an increasing number of chunks, which again consist of a fixed maximum number of measures. A measure is the smallest unit of the cube, one atomic element, and it actually contains just an integer value. The chunk is a part of the whole cube. It has the same dimension like the cube, which means it contains measures associated with a number of positions of each dimension. Because the amount of memory used by the whole cube usually will be much higher than a system may provide, each chunk offers methods for storing and loading its data onto and from the disk storage. Thus, always only a limited number of chunks is kept within the memory at the same time. Storing and loading targeted chunks is called chunk swapping and is a subsystem of the datacube structure implementation. This is similar to paging in modern operating systems with the distinction that our chunks may grow up to a specific size, hence, the memory resident chunk location table, which is a list of chunks which are currently resident in memory, varies its size. This is because aggregation results are also stored within the same datacube.

The index database contains the literal positions, the meta-information, of each dimension and maps unique integer values, the position indexes, within each dimension to them. Furthermore, it provides methods for *linearizing* the multi-dimensional position indexes used for addressing a specific measure. As already mentioned earlier, this is necessary because of the one-dimensional (linear) organization of conventional computers memory. Several methods are available, e.g. hashing, *bit encoded sparse structure* (BESS) [6], binary trees or others. Because we want to deal with a huge number of tuples, which actually means billions or more, we need an algorithm which is fast on one hand and, on the other hand, is not limited to some upper boundary. This is necessary to avoid multiple scans of the source data and allow insertion of measure aggregations after cube construction. We developed a method called *dynamic bit encoding* (DBE) [4] which is based on BESS with the difference that the bits used for each dimension are kept within bit masks which are extensible and mutually exclusive against each other one from a binary point of view. The position indexes are processed by the *or* operation using these masks, which results in a linear measure address called the *global index*.

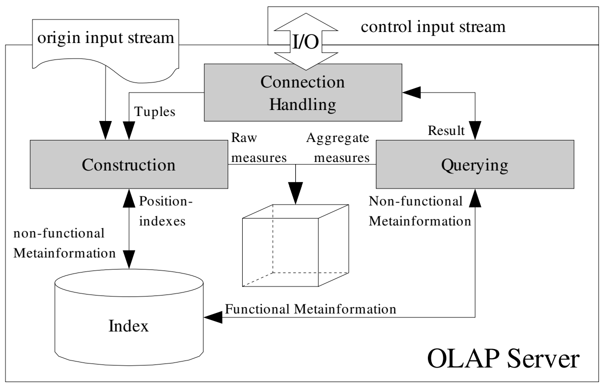

Figure 1. The system architecture of the sequential prototype.

The cube *Construction* functional block reads one tuple after the other, passes over the items to the index database, retrieves its global index and then passes the (raw) measure and its associated global index to the data cube structure.

The *Querying* functional block is some kind of highly sophisticated, recursively nested loops for aggregation of measures. Because the number of computational operations of nested aggregation depends on the size of the dimensions and thereby on the order in which dimensions are aggregated, the engine uses a kind of query plan optimization to select dimensions in a ”good” way. The procedure of dimension selection is done by traversing a tree which in literature usually is called the query lattice [6]. The task of aggregation is realized sequentially by loading one chunk after the other and aggregating them one by one. To avoid repeated computation of same aggregates, they are also stored into the cube structure as if they were raw measures and also get index entries within the index database within the appropriate dimensions.

Connection handling is the network interface which allows user interaction with the system. A typical workflow of system usage is like follows. After startup the index tables and the base cube are constructed. This is done upon loading and parsing a structured or semi-structured text file representing database tuples. It is called the *origin input stream* (see Figure 1). To each position a unique index value is assigned and this assignment is kept within the index database. Then all index values from a tuple are merged together using the DBE algorithm. The encoded global index is used to uniquely locate and store the measure within the cube. After the step of tuples import, the server opens a listening TCP socket and accepts client connections. A simple command language was defined [4] for communication between server and client. This is called the *control input(/output) stream*. A client now is able to submit queries. The server supports concurrent sessions which allows multiple users to login concurrently.

Complete system documentation, source code and related documents are available on the projects Internet site (www.gridminer.org).

4 Integration into the Grid

Our OLAP server provides a simple TCP/IP interface for client interaction (Section 3), hence, it’s open and adaptable to near any higher layer architecture like the Grid by wrapping software adapters around it. The GridMiner requires service creation upon direct Java API calls and making the new service available in the service factory of GridMiner. To realize this, we provided an interface using the Java RMI technology, describing all the available operations implemented by the engine. These operations are saved in the GridMiner Knowledge Base [9] and are thus accessible for other services through the *Dynamic Service Control Engine* (DSCE) [9], which itself is also a service in the GridMiner, describing the state, and the results of all activities.

The kind of presentation of the results of our operations is very important to provide a consistent workflow management in computational grids and thus are potential input candidates for other services even either for the OLAP service itself or for graphical output processing. The operations and the results provided by the Grid integration of our OLAP server can be grouped into two categories.

The first category includes standard OLAP operations such as roll-up, drill-down, point query and slice-and-dice. However, so far to our best knowledge, there is no standard to represent the results of OLAP operations. That’s why we designed the *OLAP Modeling Markup Language* (OMML) [3] to provide consistent OLAP results. It serves as a generic data format to carry multi-dimensional data.

The second category of operations consists of data mining methods because the engine has a built-in association rule mining functionality[1], which is used to filter meaningful patterns, the so-called strong association rules, from data cubes. Furthermore, for the representation of the results of our embedded association rule mining engine we decided to use the *Predictive Model Markup Language* (PMML), a standard defined by the Data Mining Group (www.dmg.org).

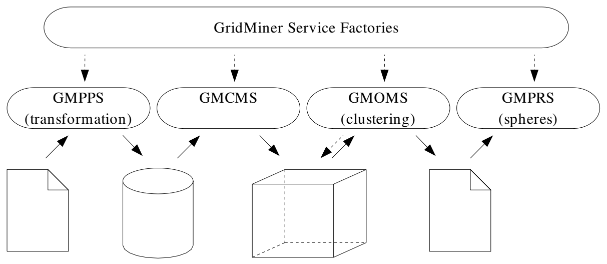

Figure 2. A typical workflow example of creation of a data cube and the application of association rule mining on it [9].

The service provided by our OLAP engine is called *GridMiner Cube Management Service* (GMCMS) and the embedded association rule mining functionality the *GridMiner Online Analytical Mining Management Service* (GMOMS). Figure 2 shows a typical example of a workflow of creation of a data cube and application of association rule mining on it. Because the cubes always contain preprocessed data and they are constructed for a distinct task, it is necessary to start preprocessing services (GMPPS) before the cube is constructed. In the next step the cube management service (GMCMS) creates an OLAP cube and after the cube is constructed a data mining algorithm can be started, which is the task of the OLAP mining within GMOMS. The results of this step are pure PMML files as explained above and they are serving as input for a presentation service which prepares the results for output presentation to the user.

[1]Decision tree based classification and clustering will be also implemented.

In this phase of implementation, the OMML and PMML outputs of our engine are stored onto the local file system. It is one of the future works to accomplish performance tests on how efficient these XML based results can be used as input for various services throughout the GridMiner system and bypassing the local storage because of its security impacts.

5 Toward Distributed and Parallel Grid OLAP Services

In the next phase, we are going to build a datacube on top of a collection of geographically distributed Grid datasets. This data cube can be distributed across a set of Grid hosts. However, the system has to virtualize all these resources. Therefore, for the end-user and other potential applications, we consider this datacube as one large virtual cube, which is distributed across a set of Grid hosts, which manage the creation, updates and querying of the associated cube portions. To develop appropriate scheduling mechanisms for these management tasks, we consider that the virtual cube is split into several smaller parts, called cube segments. But a cube segment could furthermore also be split into smaller segments and so on, till we achieve the level of chunks. Thinking object-oriented, we use the term cube element for all these cubes, cube segments and chunks, because they only differ in storage size but they have all the same number of dimensions and they all contain the same type elements which again are cube elements. The cube elements can be considered as program objects of the same class or as instances of one generic service. They can then be assigned to Grid hosts, having sequential or parallel computing power, which are responsible for their management.

The challenge which comes up with this object oriented approach is that there could no longer exist one global index for addressing measures so we will need methods for splitting indexes into global and local sets of indexes which are passed through the cube element hierarchy.

We will keep our method of storing aggregation results within the same structure with the difference that then every cube element will contain its own aggregates which are again aggregated in hierarchically higher level cube elements and so on.

This solution will allow both to create new autonomous Grid OLAP applications and federate existing geographically distributed OLAP applications as well.

6 Conclusions

OLAP is a kernel part of each modern decision support system. So far, no research effort was devoted to the development of Grid-enabled OLAP technology for analysing data repositories integrated into the Grid. In this paper, we described a sequential OLAP engine which was implemented as a Grid service on top of the Globus Toolkit 3 in Java. For this implementation, the new data indexing, data materialization and querying techniques were developed, which reflected the needs for processing and analysing large data volumes. We also briefly introduced several original design concepts developed for distributed and parallel OLAP services, which we are planning to implement in the near future.

References

[1] P. Brezany et al. Mediators in the architecture of grid information systems. Proceedings of the Conference PPAM'03, Czestochowa, Poland, September 2003.

[2] E. F. Codd. Providing OLAP (on-line analytical processing) to user-analysts: An IT-mandate. Technical report. E.F. Codd and Associates, 1993.

[3] I. Elsayed and U. Onan. Specification omml- olap modelling markup language, March 2004.

[4] B. Fišer and P. Brezany. Approaches to the development of olap engines. Techn. rep., Institute of Software Science, University of Vienna, February 2004.

[5] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, January 2002.

[6] S. Goil. *High Performance On-Line Analytical Processing and Data Mining on Parallel Computers*. PhD thesis, Northwestern University Evanston, 1999.

[7] J. Gray, et al. *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total*, S. Y. W. Su (ed.), Proceedings of the 12th Intern. Conf. on Data Engineering, pp. 152-159.

[8] J. Han. *Data Mining. Concepts and Techniques*. Morgan Kaufmann, 2000.

[9] G. Kickinger and P. Brezany. The Grid Knowledge Discovery Process and Corresponding Control Structures, Techn. Rep., Univ. of Vienna, March 2004.

[10] P. Messina. Foreward. In F. Berman, A. J. G. Hey, and G. Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[11] R. Oldfield. Summary of existing and developing data grids. White paper for the Remote Data Access group of the Global Grid Forum 1, Amsterdam, March 2001.

[12] N. W. Paton, et al. *Database Access and Integration Services on the Grid*, Technical Rep., February 2002.

IEEE COMPUTER SOCIETY