Applying Grid Technologies to XML Based OLAP Cube Construction

Tapio Niemi[1], Marko Niinimäki[2], Jyrki Nummenmaa[1], and Peter Thanisch[3]

[1] Department of Computer and Information Sciences,
FIN-33014 University of Tampere, Finland
{tapio, jyrki}@cs.uta.fi

[2] Helsinki Institute of Physics, CERN Offices, CH-1211 Geneva, Switzerland
marko.niinimaki@cern.ch

[3] IBM UK, Buchan House, St. Andrew Square, Edinburgh, Scotland
thanisch@uk.ibm.com

Abstract. On-Line Analytical Processing (OLAP) is a powerful method for analysing large warehouse data. Typically, the data for an OLAP database is collected from a set of data repositories such as e.g. operational databases. This data set is often huge, and it may not be known in advance what data are required and when to perform the desired data analysis tasks. Sometimes it may happen that some parts of the data are only needed occasionally. Therefore, storing all data to the OLAP database and keeping this database constantly up-to-date is not only a highly demanding task but it also may be overkill in practice.

This suggests that in some applications it would be more feasible to form the OLAP cubes only when they are actually needed. However, the OLAP cube construction can be a slow process. Here, we present a system that applies Grid technologies to distribute the computation needed in the cube construction process. As the data sources may well be heterogeneous, we propose an XML language as an interim format for collecting the data.

The user's definition for a new OLAP cube often includes selecting and aggregating the data. In our system this computation is distributed to the computers that store the original data. This reduces the network traffic and speeds up the computation that is now performed in parallel. We have implemented a prototype for the system. The implementation uses software packages called Spitfire (a data base front end) and Mobile Analyzer (a Java distributed computing platform). Both of these have their background in Grid technologies.

1 Introduction

The contents of OLAP databases are typically collected from other data repositories, such as operational databases. For a well-defined and targeted system, where the information needs are well-known, it may be straightforward to collect the right data at the right time. However, this collection process can be time consuming. Further, there is constantly more and more data generally available, and also the information needs develop. Consequently, it gets more and more difficult to anticipate the needs of OLAP users. This leads to a situation, where it is increasingly difficult to know in advance, what data are required and when for the desired data analysis tasks. Sometimes it may happen that some parts of the data set are only needed occasionally. It appears that collecting the right data on demand might be a better – or even the only – alternative for some applications. This way the data is also up-to-date, as it is collected when it is needed.

We have designed and implemented a prototype of the system that enables the user to construct an OLAP cube suitable for the data analysis at hand. We emphasize that our method is for the OLAP cube construction, not processing OLAP queries. Thus, the cube construction is not supposed to happen as "on-line" as answering OLAP queries. We believe that construction of a new cube enables the OLAP server to respond much faster to users' actual queries against the constructed OLAP cube. This is possible since a small cube is more efficient to process than a large one: for example it can contain a larger proportion of precalculated data than a large cube.

In a distributed data warehouse environment it is natural to distribute the data selection and aggregation computing, too. Aggregation functions (e.g. sum, average) usually used in OLAP are easy to distribute. The main principle in the distribution is that the data is processed as much as possible in the local node. In addition for parallelising the computing, this can also remarkably decrease the network traffic.

We have implemented the distributed computing applying Grid technologies. Foster and Kesselman describe the Grid as follows: "The Grid is a software infrastructure that enables flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and resources." [7] Thus, the Grid can be seen as an layered approach where applications access resources (like databases) and users and user groups have access rights to applications and resources. An essential part of our system is the Grid Security Infrastructure (GSI) [22], which allows secure connections to potentially all computers in the Grid. The user authentication is based on a common X.509 certificate, thus separate user IDs or passwords are not needed.

Data warehouses involved in data collection are often heterogeneous, yet their information should be integrated in the OLAP database. XML appears to be a suitable solution for this problem, since an XML sublanguage can be translated to other sublanguages using XSLT (Extensible Stylesheet Language Transformations) [3]. In a similar way, the OLAP data can be easily transformed into a form suitable for an OLAP server. This enables us to use different server products for data analysis, provided that they are able to read data in XML form.

We use the relational model to formalise our notion of an OLAP cube, but this does not mean that the implementation needs to be relational OLAP. In our formalism, a dimension schema is a set of attributes. An OLAP cube schema $C = D_1 \cup D_2 \cup ... \cup D_n \cup M \cup I$, where $D_1 ... D_n$ are dimension schemata, M is the set of measure attributes, I is a set of measure identification attributes, and $D_1 \cup D_2 \cup ... \cup D_n$ is a superkey for C. An OLAP cube c is a relation over the OLAP cube schema $C = D_1 \cup D_2 \cup ... \cup D_n \cup M \cup I$. If D is a dimension schema, then a relation d over D is called a dimension. It is generally assumed that each dimension schema D_k is chosen in such a way that there exists a single-attribute key K_k for it, although theoretically this would not need to be so. Also, if this is not the case, then an artificial key can be formed with values concatenated from key attribute values.

We assume that the measure items can also be identified independently of the dimension information, that is, we assume that also I contains a superkey for C. This is a fairly natural assumption, as we generally think that the measurements can be somehow directly identified. That is, the measures can be identified without the classification data using e.g. date and time information about the measurement, or some artificial ID. This may be needed in technical measurements, e.g. high energy physics. In addition, the special ID attribute is useful for vertical distribution of the OLAP data. However, we allow the sets I and D to intersect. In the extreme case, the set I may be a subset of D meaning that no different measure IDs actually exist.

```
<!DOCTYPE olap_cube [
 <!ELEMENT olap_cube (fact_table,product, ...)>
 <!ATTLIST olap_cube name CDATA #IMPLIED>
 <!ELEMENT fact_table (fact_row*)>
 <!ELEMENT fact_row EMPTY>
 <!ATTLIST fact_row value CDATA #IMPLIED
    product CDATA #IMPLIED
    export_country CDATA #IMPLIED
    import_country CDATA #IMPLIED
    year CDATA #IMPLIED>
 <!ELEMENT product (product_row*)>
 <!ELEMENT product_row EMPTY>
 <!ATTLIST product_row product_name CDATA #IMPLIED
                           sub_group CDATA #IMPLIED
                           main_group CDATA #IMPLIED>
 :
]>
```

Fig. 1. A part of the example OLAP cube DTD

Although our formal model is based on the relational model, an XML language is used to represent the actual data (in order to later expand our study to databases based on some other than relational model). Because of efficiency, we partly normalise the OLAP relation and use the so the called *star schema* style in our XML formalism to represent the OLAP cube schema and to store OLAP cube data. Figure 1 shows a part of the DTD for the OLAP cube of our example data warehouse. An example on the XML document representing the OLAP cube can be seen in Figure 6.

Our prototype implementation software applies Spitfire database front end [13] and Mobile Analyzer, a distributed computing platform [15] (see Section 4). The data used in our examples and testing the prototype implementation contains about four million entries of world trade data distributed in four different databases. The idea of the system is shown in Figure 2. The method is explained in more detail in Section 3.2.

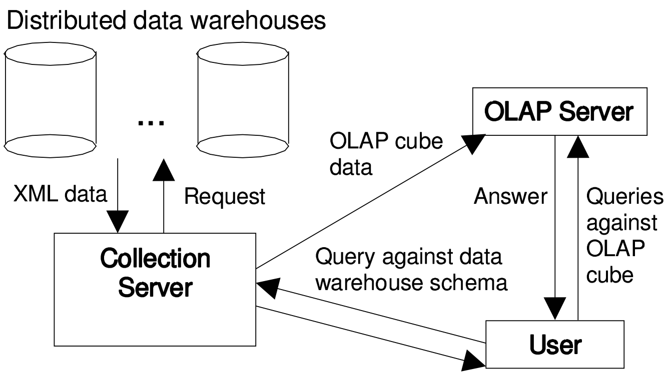

Fig. 2. The General Architecture of the System

The rest of this paper is organized as follows. In the next section, the related work is briefly studied. In Section 3, we explain how the data collection and distributed aggregation calculation can be performed. The implementation of the system is described in Section 4. Finally, the conclusions are given in Section 5.

2 Related Work

In order to achieve scalability, commercial OLAP server products have been designed to exploit distributed computing in a number of ways. For example, Microsoft's OLAP architecture copes with a large number of concurrent users by offloading aggregate processing and replicating cubes, or parts thereof, as "local" cubes on client hosts [11]. Apart from distributing the processing load, this approach can also reduce network traffic by caching results on the client for subsequent re-use.

The use of XML is starting to spread to OLAP processing through the XML for Analysis Specification [4]. At present, the use of XML is confined to describing the structure of the result of an OLAP query, as well as providing the mechanism for transmitting the query and the results over the Internet. By contrast, our approach uses XML to describe the cube structure itself. In Microsoft Analysis Services, when the user creates a new cube, it is stored in units called partitions [11]. Distributed partitioned cubes are stored on multiple servers. All of the metadata is stored on one of these servers and the partitions stored on the other servers are called remote partitions. This architecture facilitates coarse-grained parallel processing since query processing is performed on all servers containing relevant partitions.

Golfarelli et al. [9] have studied how an OLAP cube schema can be designed based on XML data. They present a semi-automated method to build the OLAP schema from XML data sources. Jensen et al. [14] study how an OLAP cube can be specified from XML data on the web. They also propose a UML (Unified Modelling Language) based multidimensional model for describing and visualising the logical structure of XML documents. Finally, they study how a multidimensional database can be designed based on XML data sources. Their method is also capable of integrating relational and XML data.

Aggregation calculation has been studied in many works. The main aim has been to perform calculations as efficiently as possible. In this spirit, distributing the calculation is studied in some works (e.g. [23, 5, 17, 8]). In some papers, the correctness of aggregations has been studied (e.g. [16, 12]). This research is focused on obtaining correct aggregations in the presence of dimension hierarchies but some of these results can be applied to distributed aggregation calculation, too.

3 Constructing an OLAP cube from distributed data warehouses

3.1 Defining Contents of New OLAP Cubes

The database / data warehouse schema is represented as an OLAP schema to the user. For simplicity, we assume that it is always possible to construct one integrated universal data warehouse schema for the whole distributed data warehouse.

In this paper we do not study how the user can deduce the contents of the OLAP cube. One possibility is to use the query based OLAP cube design method [20]. According to the method, the user defines the contents of the OLAP cube by forming MDX [18] queries against the conceptual schema of the data warehouse.

In our current prototype, the user's request is represented by a set of selection constraints and roll up operations in an XML document. An example can be seen in Figure 3. The query XML document has two parts:

1. Selection constraints: define which dimension values are taken into account. The selection constraints can be defined on any level of the hierarchy. If no selection constraints are given, then all values are taken into account.
2. Roll up operations: determine the level of detail at which data will be stored in the new OLAP cube.

```
<query_definition>
 <selection_constraints>
   <constraint name="year" value="1980, 1990"/>
   <constraint name="import_country.continent" value="Asia, Europe"/>
   <constraint name="product.main_group" value="forest"/>
 </selection_constraints>
 <roll_up_operations>
   <operation name="import_country.continent"/>
   <operation name="product.main_group"/>
 </roll_up_operations>
<query_definition>
```

Fig. 3. A sample XML query

3.2 Distributed Aggregation Calculation

Lenz and Thalheim [12] and Gray et al. [10] have studied applying aggregation functions in OLAP cubes. Gray et al. have classified aggregation functions according to their properties related to how the functions can be calculated from subsets. The groups are: 1) distributive, 2) algebraic, and 3) holistic functions. Distributive functions are relatively easy to calculate in subgroups. The most common aggregation functions in this group are sum, min, max, and count. According to Lenz and Thalheim an algebraic function can be expressed by finite algebraic expressions defined over distributive functions. An example of an algebraic function is the (arithmetic) average. It is still relative easy to compute from sub results. For holistic functions partitioning does not work, since there is no fixed size for sub results needed in computation. In this paper our work focus on distributed and algebraic aggregation functions.

We assume that the data warehouse is stored as a star schema, that is, it consists of (logically) one fact table and one dimension table for each dimension. Each of these tables can be stored in multiple sub databases. The distribution can be horizontal or vertical. In the vertical distribution we demand that the measure identifier is stored in each sub database. The vertical distribution can be useful if the data has very high dimensionality since the user analysing the data may be interested only in a small subset of the dimensions.

The horizontal distribution of the fact table enables us to distribute the aggregation computing easily. The idea is to perform the computing where the data is stored. The computing can be performed faster and the amount of data to be transmitted becomes smaller. The distribution of the fact table is simply described as predicate expressions by using XML. In Figure 4, the fact table is distributed according to years and the import country dimension according to the continent.

Figure 5 illustrates the computing methodology used in our system. We have one central component, called a collection server, which sends requests to remote databases and performs the final aggregation, if it is needed. The remote nodes can request dimensional data from each other. The needed dimension data is

```
<fact_table_distribution>
   <year="1980" database="tkt-3060-1.cs.uta.fi/trade1980">
   <year="1981" database="tkt-3060-2.cs.uta.fi/trade1981">
   :
</fact_table_distribution>
<dimension_table_distribution>
   <product_distribution>
      <product="all" database="tkt-3060-9.cs.uta.fi/products">
   <product_distribution>
:
</dimension_table_distribution>
```

Fig. 4. A distribution of a data warehouse

joined with the fact table to find out to which categories the item rolls up or evaluating selection constraint at higher levels of the hierarchy. The remote nodes send the sub results back to the collection server, though in general, the results do not arrive simultaneously. However, the collection server starts to process a sub result immediately after it has arrived. Therefore, there is no need to wait that all sub results are received and the final result will be computed shortly after the last sub result has arrived.

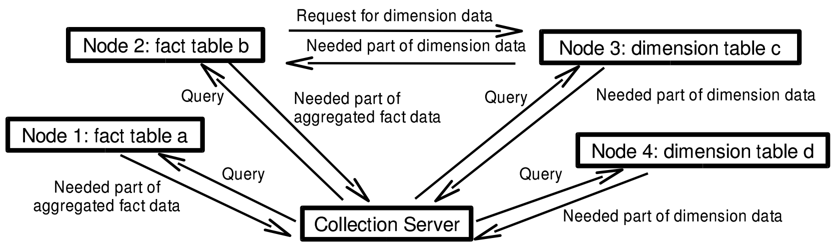

Fig. 5. The system architecture

Two main methods that can be applied to aggregation computing are sorting and hashing [10, 1]. In the sort method, the data is first sorted according to the grouping attributes and then the groups are aggregated. In the hash based methods, a hash table is used to detect which values must be aggregated. The hashing method is usually faster because no sorting is needed but, on the other hand, it may need lots of temporary storage space.

A query or request to distributed OLAP databases contains selection operations and/or roll up operations. A query containing only selections is easier to evaluate since the remote nodes always return complete data, meaning that no further aggregation computing is required in the collection server node. In remote nodes each selection constraints can be evaluated by joining one fact table and one dimension table. However, a query can contain several selections related to different dimensions, so several joins may be needed. To optimise the process, the smaller table should be transmitted to the node of the larger one. To know which one is larger, the information on the numbers of rows in tables can be stored but, in general, it is natural to assume that a fact table is larger than a dimension table. The semi-join is a commonly used method in distributed joins (see e.g. [21], [2]). In our system, we do not need a general semi-join but a simplified method can be used when joining fact and dimension tables located in different nodes. In our simplified version, we only transmit the request of the needed column names and the given selection constraints to the remote node. The selected values of the given column are sent back to perform the final join.

Evaluating roll up operations is slightly more difficult since the data must usually be summarized. If we see the OLAP cube as a relation, after an roll up operation the relation would contain tuples whose key attributes are the same. This implies that these rows must be aggregated. Using hash techniques or merging method to sorted data, summarizing data can be done in a single pass. If we have n rows in the cube relation, the time needed using only one central computer is n. If we have k computers and data is distributed equally to all nodes, we first need n/k operations in each node to perform all remote aggregations, and then the number of returned rows, n', to do the final aggregation in the central node. In the worst case $n' = n$ meaning that the distributed method is worse than a non-distributed one. However, in practice n' is significantly smaller than n. Thus, the distributed method can be much faster.

Algorithms 1 and 2 shown in the Appendix describe the distributed cube construction process in more detailed. If hash techniques are used in aggregation computing, the complexity of the algorithms is linear related to the number of fact table rows in the largest sub cube before any aggregation computing.

Example Let us assume that we have 180 countries which roll up to 6 continents. We perform a roll up operation to the continent level. In centralised model, we must do 180 operations to perform the needed aggregations. If the countries are distributed according to the continents (assuming that each continent has the same number of countries), we have 30 fact table rows in each remote node. This set of 30 possible countries rolls up to only one possible continent, so we must do 30 operations to aggregate the country data in each node. Consequently, each remote node returns only one row to the central node, that is, 6 rows in total. Thus, the total number of sequential operations is 30+6. This is 80% less than in the centralised model. Moreover, if we apply the fact that data is distributed according to the continents, we know that no similar rows can be returned and therefore no aggregations are needed to perform in the central node.

4 Prototype Implementation

Due to space limitations we are only able to give a brief review of the prototype implementation. The more detailed description can be found in [19].

The implementation of the system relies heavily on the use of XML and Grid technologies. The system is being implemented using Java and C languages (DB2 OLAP Server does not have a Java API) and Spitfire [13] software. As an OLAP server, IBM's DB2 OLAP Server is used but the system is not OLAP product dependent, so long as the input format to an OLAP server is uniform.

We use world trade data to illustrate the system. The data contains pairwise import/export figures for eight years of more than one hundred countries classified according to product groups. The XML representation of the data is shown in Figure 6.

```
<olap_cube name="trade">
   <fact_table>
      <fact_row value="200" product="fine paper"
        export_country="Finland" import_country="UK" year="1988"/>
      <fact_row value="256" product="stainless steel"
        export_country="Finland" import_country="USA" year="1989"/>
   </fact_table>
   <product>
      <product_row product_name="fine paper"
        sub_group="paper" main_group="forest"/>
      <product_row product_name="stainless steel"
        sub_group="steel" main_group="metal"/>
   </product>
   :
</olap_cube>
```

Fig. 6. A part of the example OLAP cube in XML

In our example system, the data are distributed according to years in such a way that each year is stored in a different relation. Each dimension table is stored in a single relation. A part of the XML file representing the distribution schema is shown in Figure 4.

We use a database front end, Spitfire, to access remote databases. Spitfire is developed in association with European Data Grid Project [6], provides HTTP/HTTPS-based services for accessing relational databases. Upon receiving a request that contains an SQL query from a web client, Spitfire returns a response in XML format. In addition to relational databases, we could also use XML database systems or plain XML documents on the web. For security, Spitfire contains a certificate based user authorisation system [13]. The computing facilities of the system are implemented using the Mobile Analyzer technology [15]. The basic idea of Mobile Analyzer is that the user provides Java classes that are executed remotely. The system has facilities to retrieve results and status information back from the computing servers. Like Spitfire, Mobile Analyzer uses a certificate based user authorisation system.

Each database node has a local collection server process running as a Mobile Analyzer agent taking care of the local data processing. The database requests are sent to both local and remote databases as normal SQL queries via HTTP using Spitfire. The database servers process the query and format the answers to our XML presentation using XSL stylesheets. Then the local collection server performs needed selections and aggregations and finally returns the data to the global collection server. The global collection server starts processing the data when first two sub results have arrived. If the sub results do not arrive at the same time, the collection server aggregates the first sub results while the last remote servers still process the local data. In this way, performing the global aggregations does not necessarily need much extra time.

We first tested the implementation using four computers (Pentium 1000 MHz) processing locally stored data in Tampere, Finland and one computer (Pentium 800 MHz) collecting the sub results in Geneva, Switzerland. Depending on the request, using five computers was 2-3.5 times faster than using only one computer. The distributed computing became relatively faster when the amount of locally processed data increased and the request contained many roll up operations and selections. We also tested the prototype using two computers (Pentium 1000 Mhz) in Tampere and two computers (Pentium 400 MHz) in Geneva, and then using one computer in Tampere and one in Geneva. The collection server ran in the both cases in Tampere and dimension data was read over the network. In the used query the data was aggregated to the level of the continents and the main product groups. Using five computers was now about 30 % faster than using three computers.

5 Conclusions and Future Work

We have presented a method to help in analysing large distributed heterogeneous data warehouses. The method helps the user to construct an OLAP cube from distributed data for his/her analysis requirements. The method applies the XML language and we have given an XML presentation for OLAP cubes and OLAP cube schemata. An OLAP cube in XML form can be easily transformed to a form that OLAP servers can read. Finally, the actual data analysis is done by using an OLAP server product.

The prototype implementation constructs an OLAP cube into DB2 OLAP server from distributed Spitfire data sources according to the user's input. The aggregation calculation and data selection is done in parallel in remote databases as far as possible. All the input files are in XML and also the data from remote nodes are received in XML. The input files contain a definition for "universal cube" representing the data warehouse as an OLAP cube, the definition of the contents of the new OLAP cube, and the distribution schema of the distributed data warehouse. The system performs needed selections and aggregation computing and finally constructs the OLAP cube in the DB2 OLAP server.

The current implementation only supports horizontal distribution and the whole dimension table of each dimension must be stored in a single relation. Other limitations in the prototype are that only equality constrains are supported and there can be only one selection constraint per dimension in hierarchy levels other than the most detailed one. These limitations are purely because of our simple prototype implementation. Moreover, the XML language used as a query language in the definition of the contents of the new cube does not contain advanced operations, like deriving new values from the from the existing values. The ideal solution would be the use of a standard XML query method for OLAP but, as far as we know, such method does not exist at this moment.

The manipulation of the XML OLAP cube could also be done by using standard XML tools, e.g. XSLT transformations. An efficient way of using them is a subject of our future research.

References

1. S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In T. Vijayaraman et al., editor, *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 506–521. Morgan Kaufmann, 3–6 1996.
2. P. A. Bernstein and D.-M. W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM (JACM)*, 28(1):25–40, 1981.
3. The World Wide Web Consortium. XSL transformations XSLT, version 1.0, w3c recommendation 16 november 1999. Available on: http://www.w3.org/TR/xslt, 1999.
4. Microsoft Corporation. XML for analysis specification, version 1.0. Technical report, 2001. Available on: http://msdn.microsoft.com/library/default.asp?URL=/library/techart/ XML-Analysis.htm.
5. F. Dehne, T. Eavis, and A. Rau-Chaplin. Computing partial data cubes for parallel data warehousing applications. In *Computational Science - ICCS 2001, International Conference*, volume 2131, 2001.
6. M. Draoli, G. Mascari, and R. Puccinelli. *General Description of the DataGrid Project*, 2001. Available on:http://web.datagrid.cnr.it/introdocs/DataGrid-11-NOT-0103-1_1-Project_Presentation.pdf.
7. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
8. S. Goil and A. Choudhary. An infrastructure for scalable parallel multidimensional analysis. In Z. Özsoyoglu et al, editor, *11th International Conference on Scientific and Statistical Database Management*, pages 102–111. IEEE Computer Society, 1999.
9. M. Golfarelli, S. Rizzi, and B. Vrdoljak. Data warehouse design from xml sources. In *Proceedings of the fourth ACM international workshop on Data warehousing and OLAP*, pages 40–47. ACM Press, 2001.
10. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing

group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
11. M. Gunderloy and T. Sneath. *SQL Server Developer's Guide to OLAP with Analysis Services*. SYBEX Inc, CA, USA, 2001.
12. B. Thalheim H. Lenz. OLAP databases and aggregation functions. In *Proceedings of the 13th International Conference on Scientific and Statistical Database Management*, pages 91–100. IEEE Computer Society, 2001.
13. W. Hoschek and G. McCance. Grid enabled relational database middleware. In *Global Grid Forum, Frascati, Italy, 7-10 Oct. 2001*, 2001.
14. M. Jensen, T. Moller, and T. Bach Pedersen. Specifying OLAP cubes on XML data. *Journal Of Intelligent Information Systems*, 17(2/3):255–280, 2001.
15. J. Karppinen, T. Niemi, and M. Niinimaki. Mobile analyzer - new concept for next generation of distributed computing. The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, (CCGrid 2003), Japan, May 2003. Available on http://ccgrid2003.apgrid.org/online_posters, 2003.
16. H. Lenz and A. Shoshani. Summarizability in OLAP and statistical data bases. In Y. Ioannidis and D. Hansen, editors, *Ninth International Conference on Scientific and Statistical Database Management, Proceedings, August 11-13, 1997, Olympia, Washington, USA*, pages 132–143. IEEE Computer Society, 1997.
17. W. Liang and M. Orlowska. Computing multidimensional aggregates in parallel. *Informatica, An International Journal of Computing and Informatics*, 2000.
18. Microsoft Corporation. *Microsoft OLE DB for OLAP Programmer's Reference*, 1998.
19. T. Niemi, M. Niinimäki, J. Nummenmaa, and P. Thanisch. Applying grid technologies to XML based OLAP cube construction. Technical report, CERN Open Preprint series, 2002. Available on: http://doc.cern.ch/archive/electronic/cern/preprints/open/open-2003-004.pdf.
20. T. Niemi, J. Nummenmaa, and P. Thanisch. Constructing OLAP cubes based on queries. In J. Hammer, editor, *DOLAP 2001, ACM Fourth International Workshop on Data Warehousing and OLAP*, pages 9–11. ACM, 2001.
21. M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.
22. The Globus Project. Overview of the grid security infrastructure. Available on: http://www.globus.org/security/overview.html, 2001.
23. A. Shatdal and J. Naughton. Adaptive parallel aggregation algorithms. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 104–114. ACM Press, 1995.

Appendix: Algorithms

Algorithm 1. Collection Server
Input: A data warehouse schema, a distribution schema, a request for a new OLAP cube.
Output: The OLAP cube in XML.

Fact table part:
1: Divide the request into sub requests according to the fact table distribution.

2: Send the sub requests to the remote nodes. Each request contains the selection conditions and roll up operations relevant for the node at hand.
3: Receive results from the remote nodes.
4: Perform the final aggregations using the hash or sort methods.
5: Output the fact table part of the final OLAP cube.

Dimension table part:

1: Divide the request to sub requests according to the dimension table nodes.
2: Send a request to each dimension table node whose dimension information is needed in the final cube. The request contains selection conditions and information which dimension levels are needed. (All levels may not be required if roll up operations are performed.)
3: Output the dimension parts of final the OLAP cube.

Algorithm 2. Remote Nodes
Input: A distribution schema, a sub request for the part of the new OLAP cube.
Output: A part of the aggregated data for the OLAP cube and the requested dimension data in XML.

Fact table part:

1: Determine what dimension information is needed to perform selections and roll up operations and send requests containing the selection condition and the level attribute to which the selection of the roll up operation refers to dimension table nodes.
2: Perform selections in the fact table. If roll up operations exists, then, in the same pass, change the dimension keys of the dimension to be rolled up to the values of the corresponding level attribute.
3: Perform roll up operations using hash or sort methods.
4: Return the result to the collection server node.

Dimension table nodes:

1: Receive a request.
2: Return the rows determined by the selection conditions of the requested attributes.