# A Model for Distributing and Querying a Data Warehouse on a Computing Grid*

Pascal Wehrle, Maryvonne Miquel, Anne Tchounikine
*LIRIS – INSA de Lyon*
*pascal.wehrle@insa-lyon.fr, maryvonne.miquel@insa-lyon.fr, anne.tchounikine@insa-lyon.fr*

## Abstract

*Data warehouses store large volumes of data according to a multidimensional model with dimensions representing different axes of analysis. OLAP systems (OnLine Analytical Processing) provide the ability to interactively explore the data warehouse. Rising volumes and complexity of data favor the use of more powerful distributed computing architectures. Computing grids in particular are built for decentralized management of heterogeneous distributed resources. Their lack of centralized control however conflicts with classic centralized data warehouse models. To take advantage of a computing grid infrastructure to operate a data warehouse, several problems need to be solved. First, the warehouse data must be uniquely identified and judiciously partitioned to allow efficient distribution, querying and exchange among the nodes of the grid. We propose a data model based on "chunks" as atomic entities of warehouse data that can be uniquely identified. We then build contiguous blocks of these chunks to obtain suitable fragments of the data warehouse. The fragments stored on each grid node must be indexed in a uniform way to effectively interact with existing grid services. Our indexing structure consists of a lattice structure mapping queries to warehouse fragments and a specialized spatial index structure formed by X-trees providing the information necessary for optimized query evaluation plans.*

## 1. Introduction

Applications requiring analysis of large volumes of data increasingly rely on data warehouses to efficiently organize and manage the data. The original purpose of organizing data according to a multidimensional model in a warehouse is to make the huge volumes of data generated by production information systems available to decision support systems. The "dimensions" of the multidimensional data model represent different axes of analysis. This approach can also be applied to more general scenarios of analysis applications on large distributed databases. OLAP (OnLine Analytical Processing) tools provide the ability to interactively explore the stored data by making it available in the form of so-called "data cubes" presenting both detailed and aggregated data to OLAP clients.

The growing need for computational power and storage capacity caused by the construction and operation of data warehouses with increasing size and complexity makes the use of distributed systems an appropriate choice. However, classic distributed architectures fail to significantly improve performance when it comes to scenarios with numerous simultaneously connected users, because the centralized control and management instances are maintained. The more recent concept of computing grids offers a decentralized approach to building high performance infrastructures in a very effective, economic and scalable way. Their standardized management and information services provide transparent access to distant heterogeneous resources in order to deliver a "non trivial" quality of service.

The architectural model proposed in this paper aims at seamless integration of a data warehouse on a grid computing infrastructure. The obvious benefits from deploying large amounts of detailed data and precomputed aggregates on a grid are the possibilities for parallel computing, decentralized accesses, storage and exchange of both original data and query results. From a user oriented point of view the main objective is to provide a virtual data warehouse to specialists connecting to different access points of the computing grid and to satisfy their specific needs in terms of content, axes of analysis and levels of detail.

The rest of this paper is organized as follows: in section 2 we present related work on data warehouses, their deployment on distributed system and data management on computing grids. Section 3 presents our approach to global data identification and fragmentation of the data warehouse on a computing grid. Indexation of the identified fragments is described in section 4 and section 5 exemplifies its advantages for distributed query evaluation. Section 7 forms the conclusion of our proposed solution and section 6 finally presents some future work on this particular model.

*This work is supported by the French Ministry for Research ACI-GRID project (http://www-sop.inria.fr/aci/grid/public/)

## 2. Related work

The initial purpose of data warehousing was first described by W.H. Inmon [1] as "a subject oriented, integrated, time variant, nonvolatile collection of data in support of management's decisions". The data is usually extracted from databases in production use, transformed and stored according to a multidimensional model in which the "dimensions" are specific axes of analysis relevant to future analysis and a cell represents the data associated to specific values of the dimensions. OLAP systems [2] represent the warehouse data in form of a hypercube through which the user can navigate.

As an example, figure 1 represents a hypercube with 3 dimensions: location, product and time. A cell holds the number of sold units for a given product, in a given location at a given time.

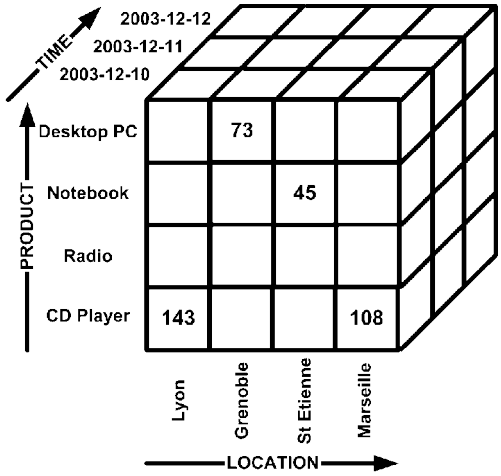


*figure 1 - OLAP hypercube*

A classical internal structure for a data warehouse is the so-called star schema [3] as shown in figure 2. It consists of a central fact table containing the most detailed data of the warehouse, the so-called facts. These are linked to several dimension tables which contain all data concerning the available axes of analysis.

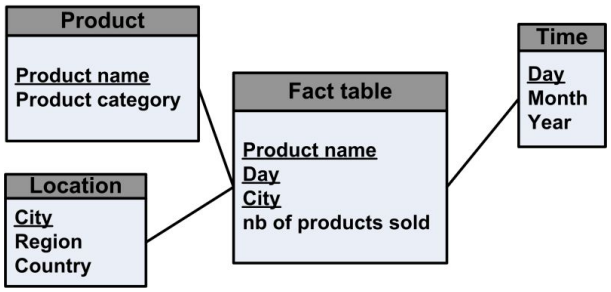


*figure 2 - data warehouse star schema*

In a multidimensional model a dimension is often organized according to a hierarchy described by a schema with containment relations between hierarchy levels. Each hierarchy level represents a level of detail for the fact data itself or corresponding aggregates. The example in figure 3 shows the hierarchy of the time dimension using a day, month and year schema. This particular instance of the dimension contains members from 2003 and 2004.

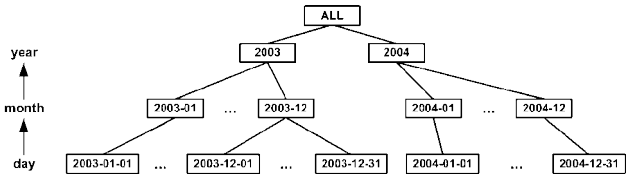


*figure 3 - example of the time dimension schema and hierarchy*

To improve query performance, several aggregations on the detailed facts are precomputed and stored within the data warehouse as well. To optimize query performance it is necessary to disperse the content of this large database among storage nodes of distributed systems in a suitable way. Constraints reducing the mobility of the warehouse data within the distributed system favor forms of vertical partitioning that keep the warehouse fragments closest to their respective sources as proposed in [4]. Given an already assembled data warehouse that has to be optimized according to the users' specific query profiles, it is more advantageous to apply a horizontal partitioning as initially described by [5] and used by [6], [7].

Client-side cache memory systems such as those proposed in [8], [9] aim at reducing response times of centralized data warehouses and [10] even introduces a peer-to-peer network between clients to improve reutilization of results. These systems have developed techniques to efficiently manage fractions of data contained in a warehouse by introducing "chunks" of data. These chunks can be uniquely identified and exchanged among peers or used to compute results from existing cache content.

Our assumptions about the structure of computing grids are best captured by I. Foster and C. Kesselman [11] who define a computing grid as "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities". According to [12], grids essentially "coordinate resources that are not subject to centralized control" and use "standard, open, general-purpose protocols and interfaces". Consequently the identifiers of data elements stored on grid nodes need to be published to make them available to querying and exchange with other nodes. Existing grid information systems like R-GMA [13] are well suited for this purpose. Efficient indexing of the locally materialized data becomes essential for this kind of distributed infrastructure and is one of the main requirements formulated by [14]. Distributed query processing on grid nodes also has existing solutions in the fields of service discovery [15] or biological analysis [16]. These

approaches focus on the concurrent evaluation of sub-queries on parts of the data which performs particularly well on horizontally partitioned data. Integration with these kinds of grid services consists in dividing an OLAP query into several tasks and subqueries depending on the location and availability of the result data.

## 3. Data identification and fragmentation

In order to be well adapted to a computing grid infrastructure, the data warehouse must be entirely distributed among grid nodes and managed in a completely decentralized way. To facilitate searching and exchanging data between grid nodes we introduce a technique for unique, global and relevant identification of warehouse data on the grid.

Unique identifiers can easily be found for the most detailed data in the warehouse by using the dimension members directly connected to the facts. It is important for any effective data management system that these identifiers can be comparable and ordered in a way that allows assembling them into greater entities. In order to achieve this we introduce a few requirements to the dimensions.

### Ordering dimension members

We note $\leqslant_{i0}$ a total order relation on the set $M_{i0}$ of dimension members at the most detailed level (level 0) for a dimension $D_i$. For dimension members with non-numeric values, so this order must be determined using semantics or simple methods like alphabetic sorting. We suppose such an order can be found for each dimension on the most detailed level. Because every subset of $M_{i0}$ inherits the total order relation, we can define an interval $R_{i0}$ of dimension members.

Let $M_{i0}$ be the set of dimension members at the most detailed level of the dimension hierarchy in one instance of dimension $D_i$. Then an interval $R_{i0} \subseteq M_{i0}$ is defined as follows:

$$R_{i0}=[m_l, m_u]=\{m \mid m, m_l, m_u \in M_{i0}, m_l \leqslant_{i0} m \leqslant_{i0} m_u\}$$

where $m_l$ ist the lowest member and $m_u$ the uppermost member of the interval.

*Example 1:*

On the example of the time dimension, one can use a chronological order on the set of members that represent the days and thus obtain arbitrary intervals in time:

*1st quarter 2004 = ["2004-01-01", "2004-03-31"]*
*fiscal year 2004 = ["2003-10-01", "2004-09-30"]*

We next aim at reproducing the total order relation on each hierarchy level. In order to be managed efficiently by our identification and indexation system dimension hierarchies need to be explicit (defined by an explicit schema of hierarchy levels), strict (no dimension member has more than one parent), onto (being represented by a balanced tree) and covering (excluding parent-child links that span more than one hierarchy level). If these conditions are met, each dimension member $m_{i0} \in M_{i0}$ on the lowest hierarchy level is associated to exactly one member on each higher level and every data element is represented by one member on each hierarchy level.

Based on the total order defined on the set of members at the lowest (most detailed) hierarchy level of every dimension instance we can now propagate the total order relation to the entire hierarchy. The notion of interval $R_{ij} \subseteq M_{ij}$ on these ordered sets is defined identically to intervals on level 0:

$$R_{ij}=[m_l, m_u]=\{m \mid m \in M_{ij}, m_l, m_u \in M_{ij}, m_l \leqslant_{ij} m \leqslant_{ij} m_u\}$$

*Example 2:*

Continuing the example based on the time dimension, we find that the order defined on the most detailed level is easily propagated to the month and year level:

*"2003-05"* $\leqslant_{i1}$ *"2003-07"; "2003"* $\leqslant_{i2}$ *"2004"*

### Base chunks and aggregated chunks

With the methods for efficient comparison and sorting of dimension members we can now create unique identification for both detailed and aggregated data elements. The notion of data "chunk" was introduced as subdivision of data for client-side OLAP result caching solutions in [8], [9] and [10]. We choose to apply the term "chunk" to a concrete unit of warehouse data at the finest grain possible and identify it by a unique combination of dimension members.

**Definition 1 – base chunk:**

Let *F* be the fact table of a data warehouse star schema and $D_1, \ldots, D_n$ be the associated dimensions. A **base chunk *c*** is equivalent to one tuple of the fact table, i.e. it contains a combination of dimension members $m_1, \ldots, m_n$, one for each dimension and one or more facts $f_1, \ldots, f_q$. We note $c=\langle\langle m_1, \ldots, m_n\rangle, f_1, \ldots, f_q\rangle$.

*Example 3:*

The base chunk containing the sales of Radios in the city of Lyon on the 12th december 2003 is represented as follows: $c'=\langle\langle$"Radio", "Lyon", "2003-12-12"$\rangle, 123\rangle$

Every base chunk can be uniquely identified through the combination of dimension members associated to it and contains the corresponding facts. Publishing their identifiers via grid information services offers a global view of the available detailed data on the grid.

Because a base chunk contains only data directly stored in the fact table, at this point our identification method remains limited to the keys of the fact table. The next step is to include aggregates that are often precomputed and stored in so-called views or OLAP cuboids. There is one potential OLAP cuboid for each combination of aggregation levels that can be applied to

the set of dimensions. This combination must be added to the identifiers in order to maintain their uniqueness.

**Definition 2 – aggregated chunk:**

An **aggregated chunk** $c_a$ contains aggregated data from a set of facts representing exactly one member on a hierarchy level $j_i>0$ for each dimension $D_i$. We note $L_m=\langle m_{1\,j_1},\ldots,m_{nj_n}\rangle$ the list of dimension members for $c_a$ and $L_d(c_a)=\langle j_1,\ldots,j_n\rangle$ its list of hierarchy levels. The information on the utilized aggregation functions $L_a=\langle a_1,\ldots,a_q\rangle$ is also added to the identification data. We note $c_a=\langle\langle m_{1\,j_1},\ldots,m_{nj_n}\rangle,v_1,\ldots,v_q,L_a\rangle$ , where $v_k=a_k(\{f_{k1},\ldots,f_{kr}\}),\quad k\in\{1,\ldots,q\}$ . The facts $\{f_{k1},\ldots,f_{kr}\}$ aggregated in $c_a$ are stored by a set of base chunks $\{c_1,\ldots,c_r\}$ .

*Example 4:*

As shown by figure 4, the aggregated chunk containing the sum of all computer products sold in December 2003 is created by computing the sum of sales for desktop PCs and notebooks on the three days of December for which data is available.

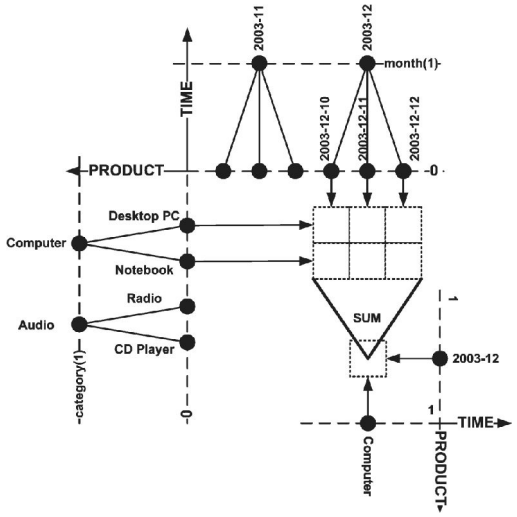


*figure 4 - creating an aggregated chunk*

The aggregated chunk contains the results of the aggregation operation on the facts extracted from the set of base chunks representing the elements of $L_m$. By limiting this approach to distributive and algebraic aggregation functions like *SUM* or *COUNT* we allow the calculation of aggregates from aggregates. In these cases aggregated chunks can be created from aggregated chunks of lower dimension hierarchy levels.

## Horizontal partitioning

Horizontal partitioning of a relational database as described by [5] consists in breaking up one large table into separate sets of tuples, called fragments. The content of these fragments is determined by a minimal and complete set of selection predicates extracted from a number of typical queries. The data contained in one fragment is therefore best suited for answering queries from a particular user group or a specialized environment. [6] and [7] extend this method to partitioning of entire star schemas. Selections are first made on the dimension tables and then transmitted to the fact table via a semi-join operation, thus creating separate instances of the star schema. Aggregated data is commonly stored in OLAP cuboids. Following our identification model, these higher cuboids are made of aggregated chunks sharing a common list of hierarchy levels $L_d=\langle j_1,\ldots,j_n\rangle$ . This similarity allows us to fragment them in the same fashion using parts of the dimension tables.

**Definition 3 - block of chunks:**

A multidimensional **block of chunks *B*** is a contiguous subset of warehouse data. It is represented by one interval $R_{ix}$ per dimension, i.e. a list of intervals $\{R_{1\,j_1},\ldots,R_{nj_n}\}$ covering all dimensions. A block of chunks contains all chunks $c=\langle\langle m_1,\ldots,m_n\rangle,f_1,\ldots,f_q,[L_a]\rangle$ with $m_k\in R_k, k\in\{1,\ldots,n\}$ . All chunks $c\in B$ share the same list of hierarchy levels.

*Example 5:*

One can imagine choosing all sales data in two months for a range of products for an arbitrary region. This set of data is represented by a block of base chunks defined by the following intervals:

$R_{10}=[\text{"Notebook"},\text{"CD Player"}]$ in products,

$R_{21}=[\text{"2003-11"},\text{"2003-12"}]$ in time and

$R_{30}=[\text{"St Etienne"},\text{"Marseille"}]$ in location. The block of chunks will thus contain all data between $c_1=\langle\langle\text{"Notebook"},\text{"2003-11-01"},\text{"St Etienne"}\rangle,f_1\rangle$ and $c_2=\langle\langle\text{"CD Player"},\text{"2003-12-31"},\text{"Marseille"}\rangle,f_2\rangle$ .

**Definition 4 - horizontal fragment:**

A horizontal fragment $HF_C=\{B_1,\ldots,B_p\}$ of an OLAP cuboid *C* is a set containing blocks of chunks with one common list of hierarchy levels $L_d(B_i)=L_d(B_1)\,\forall\, i\in\{1,\ldots,p\}$ .

We obtain the same type of fragments of the fact table as the original method and fragments of higher OLAP cuboids by using aggregated chunks. As chunks all have their corresponding dimension members tagged to them, the knowledge of the dimension hierarchy is sufficient to reconstruct the star schema equivalent to each fragment.

## 4. Indexing the data warehouse

Indexing the fragments of the data warehouse using a combination of a lattice structure and a spatial indexing structure based on X-trees ensures the availability of

detailed information on the storage location of result data and the possibilities for locally calculating results.

The underlying computing grid structure offers local storage units directly referenced by the indexing service. Pointers to locally available data delivered by the index service should be as direct as possible without invoking many additional data access and replica management services. It is necessary that all grid nodes participating in the operation of the distributed data warehouse have a complete knowledge of the dimension instances with all information concerning dimension hierarchies.

## A lattice indexing local data

The superordinate structure for indexing fragments of the data warehouse is inspired by a detailed form of the OLAP cuboid lattice with one vertex for each combination of dimension hierarchy levels [9]. An edge represents an aggregation operation on a dimension leading from one hierarchy level to the parent level.

To index a set of data warehouse fragments materialized on a grid node, fragments with identical lists of dimension hierarchy levels $L_d$ are unified. For each one of the resulting fragments one lattice vertex is created and connected to the rest of the lattice. The result is a partial instantiation of the maximum lattice (figure 5) that provides fast access to available categories of locally materialized warehouse data.

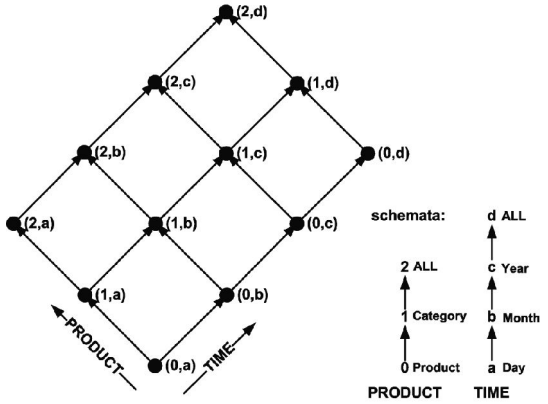


*figure 5 - maximum lattice structure for two dimensions*

## Spatial indexing of single fragments

The internal structure of a data warehouse fragment requires a well adapted indexing mechanism. All blocks of chunks contained in a fragment are described by intervals on sets of dimension members $M_{1j_1},\ldots,M_{nj_n}$ for all dimensions, where $L_d=\langle j_1,\ldots,j_n\rangle$ is the associated list of dimension hierarchy levels. The cartesian product $M_{1j_1}\times\ldots\times M_{nj_n}$ on these sets of dimension members forms a multidimensional data space. We assimilate the blocks of chunks to multidimensional objects in form of hyperrectangles located between a lower and an upper limit.

The choice of a specialized spatial indexing structures that adapts well to high-dimensional data spaces is therefore necessary to efficiently index a data warehouse fragment. We use the X-tree introduced by [17]. It is an evolution of the R*-tree [18], optimized to minimize the number of splits in high-dimensional spaces that would increase the height of the tree and the number of overlapping subtrees.

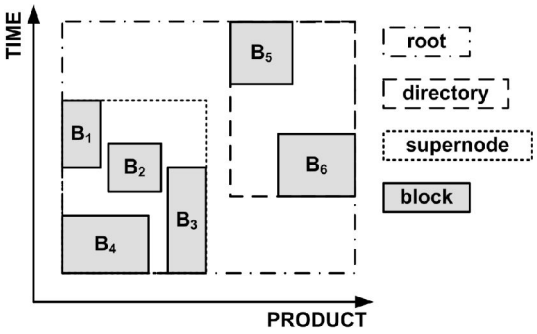


*figure 6 - X-tree indexing blocks of chunks in two dimensions*

Blocks of chunks are the objects indexed by the leaf nodes of the X-tree. Non-terminal directory nodes represent at least two child nodes through the minimum hyperrectangle containing them. Supernodes with a higher capacity avoid disadvantageous splits of directory nodes. Indexing a locally materialized fragment of the data warehouse is done by inserting its blocks into the X-tree structure of the corresponding node in the lattice structure.

## Integration of computable data

Once materialized fragments are indexed it is also possible to integrate information on aggregates that can be computed from existing data. Aggregates can only be computed from data retrieved from lattice nodes of inferior aggregation levels or from detailed data. The utilized method to determine if an aggregate chunk is computable from a set of chunks from another lattice node is inspired by the mechanism presented in [9]. It is based on a virtual counter for every chunk containing the number of paths in the lattice through which this chunk can be computed.

Let $c_1$ be an aggregated chunk and $C_1$ the set of chunks (aggregated or base) that represent $c_1$ on a lower hierarchy level. Then $c_1$ can be computed from any block $B_1$ of chunks with $C_1\subseteq B_1$. We therefore add a node representing $c_1$ in the corresponding X-tree that contains the number of blocks $B_1$ materialized in the lattice from which $c_1$ can be computed.

*Example 6:*

As shown by figure 7, the sum of monthly sales for a particular product stored in the block of chunks $B_2$ can be used to compute yearly sales represented by the virtual block of chunks $B_2'$.

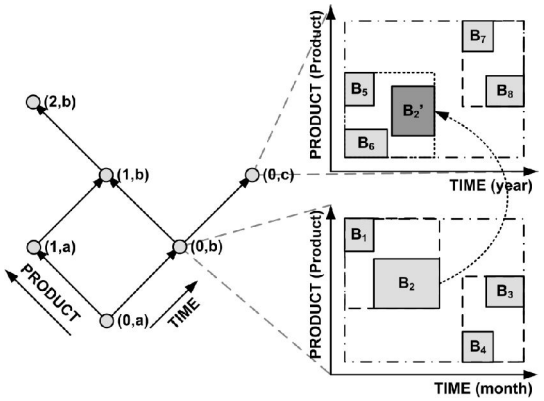


*figure 7 - combined indexing including computable data*

## 5. Query evaluation

OLAP query processing integrates both the results provided by the index and the capabilities of existing grid services in building distributed evaluation plans.

### Querying the lattice

Different pieces of information on the requested result data are extracted from the initial OLAP query. The selection predicates used by the query contain the dimension members used to identify the result data and the hierarchy levels on the dimensions included in the selection. Together with the content of the "GROUP BY" clause one can determine the exact level of detail required to answer the query. The gathered information for a given query $Q$ is represented by a list of requested hierarchy levels $L_d(Q)=\langle j_1,\ldots,j_n\rangle$ . We query the lattice structure by searching for a vertex whose combination of hierarchy levels matches $L_d$. If no such vertex exists within the local lattice the index does not contain any reference to potential result data. If the required vertex exists, the evaluation can proceed to querying the X-tree connected to it.

*Example 7:*

We consider the following OLAP query $Q_1$ on our data warehouse:

*SELECT ProdName, Month, Region, SUM(NbProdSold)*
*FROM Facts, Product, Location, Time*
*WHERE ProdName="CD Player"*
*GROUP BY Month, Region;*

This query asks for an aggregate *SUM* of CD Player sales at a detail level of months for the time dimension and region for the location dimension. We obtain .

### Querying the X-tree

The query on the X-tree requires a transformation of the original OLAP query's selection predicates. The transformation consists of creating an ordered list of requested chunks and then grouping together contiguous parts of the requested data to create queries for hyperrectangles (blocks of chunks) on the X-tree. Processing the resulting list of chunk and block of chunk queries provides detailed information on chunks or blocks of chunks either containing result data that are materialized locally and directly referenced by the X-tree, or representing result data that can be (partially) computed from locally materialized data, or missing blocks of chunks that are materialized outside the scope of the local index and need to be retrieved from other nodes of the grid. If the result contains chunks or blocks of chunks marked as computable from locally materialized data, additional searching within the lattice is performed to find the source data for the requested aggregation.

### Constructing a distributed evaluation plan

The information obtained from evaluating the query on the indexing structure it used to generate a distributed evaluation plan for the query. It consists of a number of tasks for the different grid services. These tasks include loading locally available data into memory, computing and loading of aggregates from locally available detailed data and searching and transferring missing data from other grid nodes.

Computation of aggregates from local detailed data is only worthwhile if the service executing the query has access to a sufficient amount of computing capacity which can be negotiated with local resource brokers. The results can be delivered gradually as they arrive or after the complete response is available.

## 6. Conclusion

We have introduced a data model for managing a distributed data warehouse adapted to the decentralized and heterogeneous structure of computing grids. It is inspired by existing methods of distributed data warehouse management. To build a basis for effectively identifying the warehouse data on the grid we introduced total order relations on the sets of dimension members throughout dimension hierarchies. The elementary unit of warehouse data called a "chunk" was then introduced to itself form a basis for a global indexation method for the grid. We then extended the "base chunk" to aggregated data. These elements helped constitute contiguous blocks of chunks and entire fragments of the data warehouse. In order to provide efficient indexation and management of

the identified data structures, we developed a specialized indexing method. It combines a lattice structure with spatial X-tree indexation to handle the multidimensional space created by the warehouse data. A special capability of this system is that it can include aggregated data that can be computed from locally stored data. As a result we obtain a method to efficiently evaluate OLAP queries that best exposes the distribution and potential for on-site aggregation of data.

## 7. Future work

Future research on distributed data warehouses on grid infrastructures includes work on the partitioning and distribution of the data warehouse. Our approach is based on the assumption that one needs to deploy a centralized data warehouse on a computing grid. However it could also be adapted to scenarios where existing data stored on the grid that cannot be easily moved for storage capacity, security or property reasons needs to be integrated into one virtual data warehouse. Making the distribution of data evolve with changing query profiles, limited availability of data and dynamically changing conditions for computing and data transfer on the grid is also part of our goals. Updates and additions to the warehouse data need to maintain consistency between detailed data and aggregates computed from them. This kind of evolution mechanism would help achieve decentralized maintenance of the data warehouse.

## References


[1] W.H. Inmon: *Building the Data Warehouse*, John Wiley&Sons, 1992

[2] S. Chaudhuri, U. Dayal, "An overview of data warehousing and OLAP technology", *ACM SIGMOD Record Volume 26 , Issue 1 (March 1997)*, pp.65-74

[3] R. Kimball: *The Data Warehouse Toolkit*, John Wiley&Sons, 1996

[4] M. O. Akinde, M. H. Böhlen, T. Johnson, L. V. S. Lakshmanan, D. Srivastava, "Efficient OLAP Query Processing in Distributed Data Warehouses", *Proceedings of the 8th EDBT Conference (EDBT 2002)*, March 2002, Prague, Czech Republic, Springer, pp.336-353

[5] M. T. Özsu, P. Valduriez: *Principles of distributed database systems*, Prentice Hall, 1991, ISBN 0-13-715681-2

[6] L. Bellatreche, K. Karlapalem, M. Mohania, "OLAP Query Processing for Partitioned Data Warehouses", *1999 International Symposium on Database Applications in Non-Traditional Environments (DANTE '99)*, November 1999, Kyoto, Japan, IEEE Computer Society, pp.35-42

[7] L. Bellatreche, M. Schneider, M. K. Mohania, B. K. Bhargava, "PartJoin: An Efficient Storage and Query Execution for Data Warehouses", *Proceedings of the 4th DaWaK Conference (DaWaK 2002)*, September 2002, Aix-en-Provence, France, Springer, pp.296-306

[8] P. M. Deshpande, K. Ramasamy, A. Shukla, J. F. Naughton, "Caching multidimensional queries using chunks", *Proceedings of the 1998 ACM SIGMOD Conference*, June 1998, Seattle, USA, ACM Press, pp.259-270

[9] P. Deshpande, J. F. Naughton, "Aggregate Aware Caching for Multi-Dimensional Queries", *Proceedings of the 7th EDBT Conference (EDBT 2000)*, March 2000, Konstanz, Germany, Springer, pp.167-182

[10] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, K.-L. Tan, "An adaptive peer-to-peer network for distributed caching of OLAP results", *Proceedings of the 2002 ACM SIGMOD Conference*, June 2002, Madison, USA, ACM Press, pp.25-36

[11] I. Foster (editor), C. Kesselman (editor): *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 2003, ISBN 1-55860-933-4

[12] I. Foster, "What is the Grid? A Three Point Checklist", *Grid Today, Vol. 1, No. 6*, 22 July 2002 http://www.gridtoday.com/02/0722/100136.html

[13] A. W. Cooke, A. J. G. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, J. Leake, M. Soni, A. J. Wilson, R. Corden, "R-GMA: An Information Integration System for Grid Monitoring", *Proceeding of CoopIS, DOA, and ODBASE - OTM Confederated International Conferences*, November 2003, Catania, Italy, Springer, pp.462-481

[14] L. Brunie, M. Miquel, J.-M. Pierson, A. Tchounikine, C. Dhaenens, N. Melab, E.-G. Talbi, A. Hameurlain, F. Morvan, "Information grids: managing and mining semantic data in a grid infrastructure; open issues and application to geno-medical data", *14th DEXA Workshop (DEXA'03)*, September 2003, Prague, Czech Republic, IEEE Computer Society, pp.509-516

[15] Wolfgang Hoschek, "A Unified Peer-to-Peer Database Framework for Scalable Service and Resource Discovery", *Proceedings of Grid Computing - GRID 2002, Third International Workshop*, November 2002, Baltimore, USA, Springer, pp.126-144

[16] J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. A. A. Fernandes, R. Sakellariou, "Distributed Query Processing on the Grid", *Proceedings of Grid Computing - GRID 2002, Third International Workshop*, November 2002, Baltimore, USA, Springer, pp.279-290

[17] S. Berchtold, D. A. Keim, H.-P. Kriegel, "The X-tree : An Index Structure for High-Dimensional Data", *Proceedings of the 22th VLDB Conference*, September 1996, Mumbai, India, Morgan Kaufmann Publishers/Elsevier Science, pp.28-39

[18] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", *Proceedings of the 1990 ACM SIGMOD Conference*, May 1990, Atlantic City, USA, ACM Press, pp.322-331