Distributed Database Management Systems and the Data Grid

Heinz Stockinger
CERN, European Organization for Nuclear Research, Geneva, Switzerland
Institute for Computer Science and Business Informatics, University of Vienna, Austria
Heinz.Stockinger@cern.ch
tel +41 22 767 16 08

Abstract

Currently, Grid research as well as distributed database research deals with data replication but both tackle the problem from different points of view. The aim of this paper is to outline both approaches and try to find commonalities between the two worlds in order to have a most efficient Data Grid that manages data stored in object-oriented databases. Our target object-oriented database management system is Objectivity/DB which is currently the database of choice in some existing High Energy Physics (HEP) experiments as well as in next generation experiments at CERN. The characteristics of Data Grids are described, especially within the High Energy Physics community, and needs for Data Grids are defined. The Globus toolkit is the Grid middle-ware on which we base our discussions on Grid research.

1 Introduction

Grid computing in general comes from high-performance computing, super computing and later cluster computing where several processors or work stations are connected via a high-speed interconnect in order to compute a mutual program. Originally, the cluster was meant to span a local area network but then it was also extended to the wide area. A Grid itself is supposed to connect computing resources over the wide area network.

The Grid research field can further be divided into two large sub-domains: Computational Grid and Data Grid. Whereas a Computational Grid is a natural extension of the former cluster computer where large computing tasks have to be computed at distributed computing resources, a Data Grid deals with the efficient management, placement and replication of large amounts of data. However, once data are in place, computational tasks can be run on the Grid using the provided data. The need for Data Grids stems from the fact that scientific applications like data analysis in High Energy Physics (HEP), climate modelling or earth observation are very data intensive and a large community of researchers all around the globe wants to have fast access to the data.

In the remainder of this paper we concentrate on the specific needs of High Energy Physics which can be regarded as a representative example for other data intensive research communities. In particular, we focus on the data intensive Large Hadron Collider (LHC) experiments of CERN, the European Organization for Nuclear Research in Geneva, Switzerland. At CERN, recently the DataGrid project [1] has been initiated in order to set up a Data Grid. One of the working groups explicitly deals with data management in a Data Grid [2], i.e. in the DataGrid project. The tasks to be solved include data access, migration and replication as well as query estimation and optimisation in a secure environment. In this paper we deal with the replication aspects that need to be solved in the DataGrid project. The Globus toolkit [3] is the middle-ware which we will use for the Grid infrastructure.

Scientific, data intensive applications use large collections of files for storing data. As regards the HEP community, data generated by large detectors have to be stored persistently in mass storage systems like disks and tapes in order to be available for physics analysis. In some HEP experiments, databases are used to store Terabytes and even Petabytes of persistent data. The usage of databases is still a unique feature for a Data Grid. Let us compare this to the climate modelling community: in that research domain large collections of files are available and stored in so called “flat files” without databases. This requires additional data management tasks like keeping a catalogue of available files whereas in some physics experiments in the HEP community the database management system (DBMS) takes care of this.

Currently, some new experiments in HEP use object-oriented databases management systems (ODBMS) for data management. This is true for

BaBar (an experiment at the Stanford University where currently about 150 TB of data are available in Objectivity/DB [4]), as well as for the CERN experiments, CMS and Atlas. For the CERN experiments the final decision about the DBMS (object-oriented or relational, which vendor, etc.) has not been made yet, but the current software development processes uses an object-oriented approach and Objectivity for storing data persistently. Consequently, we base our work on object-oriented database management systems and in particular Objectivity/DB.

Recently, Grid research as well as distributed database research tackles the problem of data replication but from a different point of view. The aim of this paper is to outline their approaches and to find commonalities in order to have a most efficient Data Grid that can manage Petabytes of data stored in object-oriented databases. We will provide a first basis for such an effort. Since Data Grids are very new in the research community, we see a clear need for identifying the characteristics and requirements of Data Grids and how they can be met in a most efficient way. Special attention will be given to data consistency and communication issues.

Optimising data replication and access to data over the WAN is not addressed sufficiently in database research. In a DBMS there is normally only one method of accessing data. For instance, a data server serves pages to a client. For the Data Grid, such a single access method may not be optimal. Using an ODMBS also has some restrictions, which are pointed out here and some possible solutions are given.

We elaborate on different data consistency models and how global transactions can contribute to this. Global transactions are built on top of transactions provided by a database management system at a local site. As opposed to database research, a separation of data communication is required. In particular, global control messages are exchanged by a message passing library whereas the actual data files are transported via high-speed file transfer protocols. A single communication protocol is usually used in a commercial ODBMS for exchanging small amounts of data within database transactions. This communication mechanism is optimised for relatively small transactions but may not be optimised for transferring large files over the WAN with high performance control information exchanged between distributed sites. We will propose solutions for efficient, asynchronous replication and policies with different levels of data consistency in a Data Grid.

The paper is organised as follows. Section 2 will give an introduction to related work in the database as well as the Grid community. The issues raised there will be further analysed in later sections of the paper. Section 3 deals with replica catalogues and directory services that are used in Grid research and points out how these techniques can be mapped to the database approach. Section 4 discusses Objectivity, its replication option and some general ODBMS aspects. Since we assume that the usage of Grid applications will not be fully transparent to the end user, we dedicate Section 5 to possible implications. Section 6 discusses data consistency and replication methods known in database research and practise. Possible update synchronisation solutions are given in Section 7, which is followed by concluding remarks.

2 Related work on data replication

We first identify some selected related work in the database community as well in the Grid community. From this we derive commonalities and discuss briefly what the contribution for an efficient Data Grid can be. The common aspects will be dealt with throughout this paper.

2.1 Distributed DBMS research

Distributed database research basically addresses the following issues.

- Replica synchronisation is based on relatively small transactions where a transaction consists of several read and/or write operations. In contrast, in HEP a typical data production job can write a whole file and thus a transaction should be relatively "large".
- Synchronous and asynchronous replication (more details can be found in Section 6): Most of the evaluation techniques are based on the amount of communication messages needed.
- Cost functions for network or server loads are rarely integrated into the replica selection process of a DBMS.
- Rather low amount of data as compared to the Petabyte scale of HEP.

Data access over the WAN and optimisation of replicas is rarely addressed in database research. In a DBMS there is normally only one internal method of accessing data. For instance, a data server serves pages to a client. In detail, a client sends a request for data to the DBMS, and the DBMS (in particular the data server) uses the implemented method for serving the requested data. For read and write access the same method is used, independent of the amount of data accessed. For the Data Grid, a single access method is not optimal depending on the number of objects to be accessed from a file. For accessing large amounts of data of a remote file, the usage of a file transfer mechanism to transfer the entire file and access the file locally can be more efficient in terms of response time than remotely accessing (and thus streaming) many single objects of a file. More details will be given in Section 5. Based on cost functions it can be even more efficient to send the application program to the remote data [5].

2.2 Grid research

The Globus project provides tools for Grid computing like job scheduling, resource discovery, security, etc. Recently, Globus is also working on a Data Grid effort that enables fast and efficient file transfer, a replica catalogue for managing files and some more replica management functionality based on files. Replica update synchronisation is not addressed. In the Grid community there is a general tendency to deal with replication at the file level, i.e. a single file is the lowest granularity of replication. This has the advantage that the structure of the file does not need to be known by the replication manager that is responsible for replicating files from one site to another sites over the WAN. This is also a valid simplification for most of the HEP replication use cases and, thus, is also the main focus of this paper. Related work in the HEP community can be found in the two Data Grid projects PPDG [6] and GriPhyN [7]. There is still the possibility to deal with replication at the object level which requires more sophisticated techniques than file level replication. Object handling is addressed in [8]. However, as soon as data items in a replicated file are changed, these changes have to be propagated to all other replicas. This requires the knowledge about the data structure of the file. Possible solutions to this problem are given in Section 6.

2.3 Commonalities – the usage of a replication middle-ware

Research results can be combined from both communities by introducing a replication middle-ware layer that manages replication of files (and possibly some objects) by taking into account that each site in a Data Grid will manage data locally with a database management system. Thus, the middle-ware is responsible for site to site (or *inter-site*) replication and synchronisation whereas the local DBMS takes care of transactions on local data.

In Grids there are many tools for monitoring applications and network parameters. These can be used for filling the gap. Hence, a hybrid solution between database and Grid research can be identified.

One of the main questions to be answered is to what level the replication middle-ware will be able to replace a pure distributed DBMS approach where also inter-site replication issues are dealt with. Clearly, with a replication middle-ware there are many more restrictions for update synchronisation and transparent access to data. There will also be a performance penalty for replica synchronisation. However, an aim of the replication middle-ware is to provide several relaxations of the concept of transparent data access and data consistency. For the remainder of the paper we assume to use such a replication middle-ware for a Data Grid.

3 Replica catalogues and directory services

Access to data is most important for data analysis in the HEP as well as in any other scientific community. Access to replicated data requires specific data and meta data structures and can be addressed in different ways. A DBMS has an internal access method whereas in a Grid environment these data structures are provided explicitly as a service to the end user. In this section, emphasis is put on how to combine techniques from a Data Grid and a database environment.

The management of replicas requires certain meta data structures that store information about the distribution of and access to data. Whereas an object location table [9] can be the data structure for managing objects, for files we can use a directory service holding a replica catalogue.

There are many ways to deal with replica catalogues and many different protocols and implementations are available. We want to limit ourselves here to the approach used by Globus. Since in the Globus Data Grid effort the replica catalogue is implemented by an LDAP directory service, we dedicate this section on analysing how a directory service can be used for file replication where files are managed by a local DMBS.

In principle, a DBMS provides, by definition, a catalogue of files that are available. As for Objectivity/DB this is an explicit file catalogue. An LDAP directory service basically does the same for flat files which may or not have connections (associations between objects in two or more files) to each other. Now the questions arise what is the use of LDAP when data are stored in a DBMS? In order to answer this question, we identify the key problems that need to be addressed in a heterogeneous Data Grid environment with many sources of data: managing replicas of files and dealing with heterogeneous data stores.

- **Managing replicas:** As for Objectivity, a file catalogue exists. The granularity for replication is now assumed to be on the file level. When multiple copies of a file exist, they have to be introduced to a global replica catalogue which is an extended version of the native file catalogue of the DBMS. Furthermore, a name space has to be provided that takes into account multiple replicas. Both features are enhancements to a DBMS that only considers single sources of information. Note that we assume here that the underlying DBMS does not support file replication. Using the LDAP protocol and a database backend, the required replica catalogue information can be stored. Another approach is to simply build the catalogue and the access methods with the native schema of the DBMS and store the replica information in a particular file of the DBMS.
- **Heterogeneous data stores:** The diversity of the HEP community requires an approach which supports heterogeneous data stores since different kinds of data may be stored in different formats and also in data stores of different vendors. Even the two different database paradigms, relational and object-oriented, should be supported. In this case, a standard protocol for directory information like LDAP seems to be a good choice since replica location information can be accessed in a uniform way without the knowledge of the underlying data store.

Combining a DBMS with a directory service means to expose the replica catalogue to a larger user community, where it is not necessary to have a database specific front end to access the directory information. However, the LDAP protocol still needs to have a database backend where the file information is stored and concurrency mechanisms have to be established. The database backend can either be a vendor specific and LDAP specialised database, or any database of choice like Oracle or Objectivity/DB. The usage of a directory service allows an open environment and hence allows the integration of data sources of different kinds. Note that the problem of accessing heterogeneous data sets is not addressed here and may require mediators for data integration. However, introducing LDAP is at least a starting point for extensibility.

Another point important point in replica management is synchronisation of single sites and keeping replica catalogues up to date. Using the LDAP directory service also implies that each site, that stores replica catalogue information, runs an LDAP server locally. LDAP commands can be used for synchronisation. For a homogeneous database approach, where only a single DBMS is used to store the entire data of the Data Grid, it may not be necessary to have an LDAP server for synchronisation - any communication mechanism for exchanging synchronisation information between sites in the Data Grid is sufficient. However, since each data site using a directory service will have its own LDAP server, this server can also be used as a synchronisation point for distributed sites. Whenever a file is introduced to a site, this information has to be made public within a global name space spanning all the other sites in the Data Grid. The replica synchronisation protocol to use depends on the data consistency requirements imposed by the application.

We conclude that the usage of LDAP in combination with a DBMS seems to be useful in a heterogeneous environment and for synchronisation of sites holding replica catalogue information.

4 Objectivity/DB

Since Objectivity is the main ODBMS system we are referring to in this paper, we dedicate this section to explaining more details of this product and problems that we were confronted with when using Objectivity for wide area data replication. Some of the problems mentioned are specific to all object-oriented data stores while others are Objectivity specific. However, the usage of an ODBMS allows us some simplifications like a global namespace and a global schema information.

Objectivity/DB is a distributed, object-oriented DBMS which has a Data Replication Option called DRO. This option is optimised for synchronous replication over a local area network. In this section, we briefly describe DRO and its drawbacks, and state complications that one comes across when persistent objects have to be replicated.

4.1 Objectivity's data replication option

DRO provides a synchronous replication model on the file level, i.e. entire Objectivity databases which are mapped to physical files can be replicated. Note that in the following section we use the term *replica* to refer to a *physical instance* (a copy) of an Objectivity file. There are basically two ways to use DRO: data can be written first, and then synchronised and replicated (**populate - replicate**). Multiple replicas can be created and synchronised and the data can be written into all the replicas at the same time (**replicate - populate**). Once a replica is synchronised with another one, all the database transactions on a single replica are synchronised with other replicas.

Objectivity/DRO performs a dynamic quorum calculation when an application accesses a replica. The quorum which is required to read or write replicas can be changed, which provides some flexibility concerning data consistency. The usage of a quorum method is well established in database research and goes back to an early paper [10].

There is one possibility to overcome the immediate synchronisation. A replica can be set to be off-line. However, only if a quorum still exists, data can be written to the off-line replica. There is no clean way in DRO to perform a real asynchronous or batch replication where you specify a synchronisation point in time. An asynchronous or batch replication method allows replicas to be out of sync for a certain amount of time. Reconciliation of updates is only done at certain synchronisation points, e.g. every hour. The lack of such an explicit asynchronous replication method is one of the reasons why DRO is not considered as a good option for WAN replication.

Like many commercial databases, Objectivity does not provide any optimisation for accessing replicas. The replica catalogue only contains the number of copies and the location of each single copy of a file. Once an object of a replicated file has to be accessed, Objectivity tries to get the first copy of the file which appears in the catalogue. It does not examine the bandwidth to the site nor takes into account any data server load. Some operations such as database creation require all replicas to be available. When we talk about Objectivity in any other section or subsection of this paper we mostly ignore the fact that DRO exists and thus see Objectivity only as a distributed DMBS with single copies of a file. However, these single copies of a file can exist at distributed sites in the Data Grid and thus remote access to Objectivity data is possible.

4.2 Partial replication and associations

In this paper, as well as in many Grid environments, replication is regarded to be done on the file rather than on the object level and hence a file is regarded to be the smallest granularity of replication. In Objectivity, single objects are stored in containers. Several containers are stored together in an Objectivity database which corresponds to a file. Each object can have associations (also called links or pointers) to other objects which may reside in any database or container. Let us now assume that two objects in two different files are connected via one association. When an Objectivity database (a file) is replicated, this association gets lost when only one of the files is replicated. Note that there is still the possibility of remotely accessing objects. Hence, the replication decision has to be carefully made and possible associations between objects have to be considered. This may result in replicating a set of files in order to keep all the associations between files. Furthermore, this also imposes severe restrictions on partial replication where only particular objects of a file are replicated. Again, when a certain set of objects is selected which does not have associations to objects which are not in the set, the replication set is *association safe*. This is a particular restriction of object-oriented data stores and does not only hold for Objectivity.

This is also an import difference and complication compared to a relational DBMS. Whereas in an ODBMS an object can be stored in any file, in a relational database data items are stored in tables that have well defined references to other tables.

4.3 The Objectivity file catalogue

The integration of files into the native Objectivity file catalogue is done with a tool called **ooattachdb** which adds a logical file name of the physical location into the catalogue. Furthermore, it guarantees that links to existing Objectivity databases are correct. The schema of the file is not checked. This feature is used when a file created at one site has to be integrated into the file catalogue of another site.

Since the native catalogue only has a one-to-one mapping from one logical to one physical file, replicas are not visible to the local site (not taking into account DRO). Furthermore, it is possible to have single links from one site to another one. For instance, site 1 has a file called X and this file shall be shared (not replicated) between several sites. The file name and the location can be integrated into the local file catalogue and a remote access to the file can be established. Note that this requires an Objectivity Advanced Multi-threaded Server (AMS) running or the usage of a shared file system like AFS that connects both sites. The AMS is responsible for transferring (streaming) objects from one machine to another one and thus establishes a remote data access functionality. We want to address the more general solution where no shared file system is available.

In the HEP community it is generally agreed and proven that DRO is not optimised for the use in a WAN. Hence, all our discussions here neglect the DRO of Objectivity/DB and we conclude that the current implementation of DRO is not a feasible solution for the HEP community.

5 Implications for Grid applications

We claim that the usage of files that are replicated in a Data Grid will have implications for Grid applications that are new to the HEP user community. We want to address explicitly some possibilities of how a replicated file instance can be accessed and give some implications of read ahead assumptions.

5.1 Accessing replicated files

A standard way to access a database in Objectivity is to issue an "open" command on the physical file. This concept is unique for the object-oriented database concept and very much resembles the UNIX way of accessing files. However, if a single object is accessed, the file which it belongs to is transparent to the user. Once an object handle or reference is available, the object ID to this object contains information about the file to which it belongs. Since the native Objectivity "open" works on the Objectivity catalogue rather than on the global replica catalogue, the user does not have access to the whole name space in the Data Grid. The "open" has to be adapted to do the lookup and the data access via the directory service. Plug-ins are required to transfer the requested data to the user application. This is still an open issue and needs further research concerning:

- caching files locally
- adding the file to the local catalogue and creating a replica: this may only be useful when data are accessed frequently
- transferring the whole file versus only sub sets of a file

Objectivity provides functionality for transparently accessing data stored on tapes. This is done by using an extension of the AMS backend that checks if a file is locally stored on disk and fetches the file from tape if it is not found on disk. This functionality can be extended to go to a global replica catalogue and fetch files from remote sites. This would provide a transparent way of accessing replicas of files. We will further investigate this option.

5.2 Read ahead

Reading data ahead (pre-fetching) before using them is a commonly used optimisation technique. However, this imposes another problem, which we want to illustrate by an example. Let us assume a file size of 2 GB and a program that wants to access 5 objects in the file where each object has a size of 10 kB. It is obvious that we want to transfer the requested objects rather than the whole file, and having unnecessary information transferred over the network. There is a clear need for a mapping instance that maps requested objects to files in order to determine in which files the requested objects reside. Based on the type (class definition in the data definition language of the ODBMS) of the objects it can be roughly determined how much data needs to be transferred over the network (depending on how much dynamically allocated information is stored in the object). This problem cannot be solved when only the "open" file operator is used. Only at runtime the application can determine which data are needed and a pre-fetching of requested objects can tackle this problem. We can summarise this as a query optimisation problem where access to the data shall be optimised by providing only the necessary objects that are required. Related work can be found [11,12].

Pre-fetching can also be addressed at the file level. We assume that a user is aware that files are distributed to different sites in the Grid and the time to access data very much depends on the location of the requested file. We further assume that an object in a file can only be accessed when the whole file is available locally at the client site. This is assumed for simplicity and we neglect the fact of accessing some parts of a file remotely. In the worst case, all the requested files are available remotely and need to be transferred to the local site. If the size of a file is large and also the amount of requested files is high, the I/O part of an application takes a long time before the actual computation on the data can be done. The application can give a hint to the system to tell it how long it would need to serve the required files and when the computation can be started. Thus, data can be pre-fetched before the application is using the data. In the Grid environment, the necessary information for pre-fetching files can be provided by diverse monitoring tools and information services.

This is also a sociological aspect of Grid applications in the sense that a user of a data-intensive application requests its data in advance. One can also reserve the network bandwidth and start the application at a certain point in the future.

6 Data consistency and replication methods

One of the main issues in data replication is the consistency of replicas. Since a replica is not just a simple copy of an original but still has a logical connection to the original copy, we have to address the data consistency problem. The easiest way to tackle the consistency problem is in the field of read-only data. Since no updates of any of the replicas are done, the data is always consistent. We can state that the consistency can reach its highest degree. Once updates are possible on replicas, the degree of data consistency normally has to be decreased (provided we have read and write access to data) in order to have a reasonable good response time for accessing data replicated over the WAN. Clearly, consistency also depends on the frequency of updates and the amount of data items covered by the update. Thus, we can state that the degree of data consistency depends on the update frequency, amount of data items covered by the update and the expected response time of the replicated system, i.e. the Data Grid.

Let us now identify in more detail the different consistency options which are possible and which are reasonable for HEP and within the DataGrid project.

In this section we also introduce the term *global transaction* which has to be distinguished from a *local transaction*. A local transaction is done by the DBMS at the local site whereas a global transaction is an inter-site transaction which spans multiple sites and thus multiple database management systems. Furthermore, *local consistency* is maintained by the DBMS whereas the *global consistency* spans multiple sites in the Data Grid.

6.1 Synchronous replication

The highest degree of consistency can be established by having fully synchronous replicas. This means that each local database transaction needs to get acknowledgements from other replicas (or at least a majority of replicas). In practice, as well as in the database research community, this is gained by global transactions which span multiple sites. Objectivity/DRO supports such a replication model. Each time a single local write transaction takes place, the 2-phase-commit protocol and normally also the 2-phase-locking protocol are used to guarantee serialisability and global consistency. This comes at the cost of relatively worse performance for global writes compared to local writes with no replication at all. Consequently, one has to decide carefully if the application requires such a high degree of consistency.

We derive from this statement as well as from the current database literature that the type of replication protocol and hence the data consistency model has to be well adapted to the application. Data in the DataGrid project will have different types and not all of them require the same consistency level. In this paper, we try to point out briefly where different replication policies are required. Furthermore, we claim that a replication system of a Data Grid should not only offer a single policy but several ones which satisfy the needs of having different data types and degrees of consistency.

For a middle-ware replication system it is rather difficult to provide this high degree of consistency since global, synchronous transactions are difficult to establish. Within a DBMS a global, synchronous transaction can be an extension of a conventional, local transaction, i.e. the DBMS specific locking mechanism is extended to the replicas. Since a distributed DBMS like Objectivity has built-in global transactions, no additional communication mechanism like sockets or a message passing library is required. Hence, the performance for an integrated distributed DBMS is superior to middle-ware replication systems that have to use external communication mechanisms.

Now the following question arises: why not use a distributed DBMS like Objectivity to handle replication? Several points are already covered in Section 4 but there is another major point. Objectivity does not provide flexible consistency levels different kinds of data. Hence, we aim for a hybrid solution where a local site stores data in a DBMS which also handles consistency locally by managing all database transactions locally. A Grid middle-ware is required to provide communication and co-ordination between the local sites. The degree of independence of a single site needs to be flexibly managed. A form of global transaction system is necessary. Let us illustrate this by an example. Some data are allowed to be out of sync (low data consistency) whereas other types of data always need to be synchronised immediately (high consistency). Thus, global transactions have to be flexible and do not necessarily always have to provide the highest degree of consistency. Furthermore, a site may even want to be independent of others once data are available.

6.2 Asynchronous Replication

Based on the relative slow performance of write operations in a synchronously replicated environment, the database research community is searching for efficient protocols for asynchronous replication at the cost of lower consistency. Currently, there is no standard for replication available, but a few commonly agreed solutions:

- **Primary-copy approach**: This is also known as “master-slave” approach. The basic idea is that for each data item or file a primary copy exists and all the other replicas are secondary copies [13]. The updates can only be done by the primary copy which is the owner of the file. If a write request is sent to a secondary copy, the request is passed on to the primary copy which does the updates and propagates the changes to all secondary copies. This policy is implemented in the object data stores Versant [14] and ObjectStore [15]. Also Oracle provides such a feature. The primary-copy approach provides a high degree of data consistency and has improved write performance features compared to synchronous replication because the lock on a file has not to be agreed among all replicas but only by the primary copy.

- **Epidemic approach**: User operations are performed on any single replica and a separate activity compares version information (e.g. time-stamps) of different replicas and propagates updates to older replicas in a lazy manner (as opposed by an eager, synchronous approach) [16]. Thus, update operations are always executed locally first and then the sites communicate to exchange up-to-date information. The degree of consistency can be very low here and this solution does not exclude dirty reads or other possible database anomalies. Such a system can only be applied for non time-critical data since the propagation of updates can also result in conflicts which have to be solved manually.
- **Subscription and relatively independent sites**: Similar to the epidemic approach, another policy is that a site only wants to have data and does not care about consistency at all. When any other site does updates on a particular file, only certain sites are notified of the updates. This follows a subscription model where a site subscribes explicitly to a data producing site. A site that has not subscribed is itself responsible to get the latest information from other sites. This allows a site to do local changes without the agreement of other sites. However, such a site has to be aware that the local data is not always up-to-date. A valid solution to this problem is to provide an export buffer, where the newest information of a site is stored, and an import buffer, where a local site stores all the information that needs to be imported from a remote site. For instance, a local site has finished writing 10 different files and puts the file names into the export buffer. A remote site can than be notified and transfers the information from the export buffer to its local import buffer and requests the files if necessary. This approach allows more flexibility concerning data consistency and independence for a local site. A site can decide itself which data to import and which information to filter. Furthermore, a data production site may not export all the locally available information and can filter the export buffer accordingly. A valid implementation of this approach can be found in the Grid Data Management Pilot (GDMP) [17].

6.3 Communication and Transactions

As outlined above, there is a clear need for global transactions. Such a transaction does not necessarily need to create locks at each site, but at least a notification system is required to automate and trigger the replication and data transfer process. In general, there is a clear separation between exchanging control messages which organise locks and update notifications, and the actual data transfer. This is an important difference to current database management systems. Replication protocols are often compared by the amount of messages sent in order to evaluate their performance. The type of a message is dependent on the DBMS. A message of the same type is then sent to do the actual update using the same communication protocol. In Data Grids where most of the data are read-only, we can divide the required communication into the following two parts. This concept is also realised in GDMP [17].

- **control messages**: These are all messages that are used to synchronise distributed sites. For instance, one site notifies another site that new data are available, update information is propagated, etc. To sum up, replication protocols are using these control messages.
- **data transfer**: This includes the actual transfer of data and meta data files from one site to another one.

This separation is similar to the FTP protocol where we also have this clear separation between the two tasks [18]. The main point for this separation is to use the most appropriate protocol for a specific communication need. Simple message passing is appropriate for exchanging control messages whereas a fast file transfer protocol is required to transfer large amounts of (large) files. In a Grid environment both protocols are provided. To be more specific, in Globus the GlobusIO library can be used for control messages and Grid-FTP implementation [19], which is based on the WU-FTP server and the NC-FTP client, serves as the data transport mechanism. A single communication protocol used in an ODBMS like Objectivity may not be optimal for transferring large files over the wide area network.

6.4 Append Transactions

Based on the HEP requirements, we see a need to increase the traditional DBMS transaction system by a new transaction, called *append transaction*.

In a conventional DBMS there exist only two different kinds of transactions: read and write transactions. A write transaction itself can either write new data (append) or update existing data. In terms of data management, these two operations require different tasks. Whereas an update transaction has to prevent concurrent users from reading old data values, an append transaction only has to make sure that the data item to be added satisfies a *uniqueness condition.* A uniqueness condition is a condition which guarantees that a data item appears only once and can be identified uniquely. In HEP this condition can be satisfied by the following feature: sites often write data independently and do not change any previously written data items. Since different sites will write different data by definition (thus we can derive an inherent parallelism in the data production phase), this uniqueness condition will hold. This is true for HEP specific reconstruction software as well as simulated data created via Monte Carlo processes.

Objectivity provides object IDs and unique database IDs for single files. Hence, it has to be guaranteed that newly created objects do not have the same OID (the same is true for database IDs). Since an append transaction does not have to lock the whole file but only the creation of new objects, it can allow multiple readers at the same time while an append transaction creates new objects. This again allows for having different consistency and response time levels.

6.5 File replication with Objectivity using a replication middle-ware

Driven by real world replication requirements in High Energy Physics, we want to give a possible approach for replication of read-only Objectivity database files. In principle, replication of Objectivity database files does not impose a big problem concerning data consistency. Each site has to have its Objectivity federation that takes care of managing files and the Objectivity file catalogue. There are several ways to deal with a file replication of read-only files. It has to be guaranteed that a unique Objectivity naming scheme is applied by all sites in the Data Grid. We outline two possible approaches.

Each site can create a database file. In order to provide unique database names, a global transaction has be called before the actual file creation. The transaction checks in each local Objectivity file catalogue if the file already exists. If not, the site creates the database locally and initiates a database creation at the remote sites as well. All remote replicas of a specific file have to be locked since only one site can write into a file at a time. Once the local site has completed the writing process, the lock on the remote replicas is released. For such a system we require a replica catalogue at each site in addition to the local federation catalogue. In the replica catalogue a flag is needed for initiating a lock on a file. The actual file locking has to be implemented with a call to the Objectivity database. Furthermore, each transaction on a file in the Objectivity file catalogue has to be done via the replication catalogue. Consequently, a database user is not allowed to use the conventional Objectivity “open” to create a new file but has to contact the replica catalogue in order to write a new database file. All database transactions have to go through a high level API that always contacts the replica catalogue first.

An easier approach, which is currently used in the CMS experiment, is the allocation of Objectivity database IDs to different remote sites. This guarantees only a unique database-ID allocation but not a unique name for a database. Consequently, in order to have a fully automatic system, a unique name space has to be provided. This can only be guaranteed if on creation of a database file the name is communicated to other sites and then agreed on. However, another solution is to provide a naming convention like adding the host and domain name of each local site to the database name. This also guarantees unique file names. The populate-replicate approach also does not require the locking of remote replicas which allows for a faster local write. This is a common approach for asynchronous replication.

7 Possible Update Synchronisation for a Data Grid

In the previous sections we have mainly addressed replication of read-only files. Although most of the data in the High Energy Physics are ready-only, there will be some replicated data that will change and thus need update synchronisation. In this section we provide some possible update synchronisation strategies which can be implemented using a replication middle-ware for the Data Grid.

We have already described that it is rather difficult for a middle-ware system to do replica update synchronisation on the object rather than on the file level since a middle ware system cannot access DBMS internals like pages or object tables. A common solution in database research is to communicate only the changes of a file to remote sites. Since an Objectivity database file can only be interpreted correctly by a native Objectivity process, the file itself appears like any binary file and a conventional process does not see any structure in the file. We call this the *binary difference approach*. As second possibility to update data stored in an ODBMS, we use an object-oriented approach which requires knowledge about the schema and data format of the files to be updated. Both approaches are outlined in this section.

7.1 Binary Difference Approach

There is also the possibility to use a tool called XDelta [20], which produces the difference between any two binary files. This difference can than be sent to the remote site. This site can then update the file, which is out of date, by merging the diff file with the original data file. XDelta is a library interface and application program designed to compute changes between files. These changes (deltas) are similar to the output of the "diff" program in that they may be used to store and transmit only the changes between files. However, unlike diff, the output of XDelta is not expressed in a human-readable format - XDelta can also apply these deltas to a copy of the original file(s).

7.2 Object-oriented approach

Another approach is to create objects that are aware of replicas. In principle, an object can be created at any site and the creation method of this object has to take care of or delegate the distribution of this object. The class definition has to be designed in a way that there is some information on the amount and the site of replicas. For instance, an object should be created at site 1 and replicated to the sites X and Y. A typical creation method can look like follows:

```
object.create (site1, siteX,
siteY);
```

The advantage of this approach is that all the necessary information is available to create the object at any site. When an update on one of the objects is done, the update function has to be aware of all the replicas and the sites that need to be updated. This can be compared to the stored procedure approach which is known in the relational database world. In principle, the model presented here has similar ideas. A local site may update immediately and can store the updates into a log file. Based on the consistency requirement of remote sites, the log information is sent to the remote sites which apply the same update function as the original local site.

The update synchronisation problem is then passed to a ReplicatorObject that is aware of replicas and the replication policy. The ReplicatorObject in turn can provide different consistency levels like updating remote sites immediately, each hour, day, etc. When large amounts of data are used, there may be most likely a scalability problem of managing all the logging information. However, since each object is identified by a single OID, only the parameters of an update method together with the OID have to be stored.

```
object.update_parameter_x (200);
// OID = 38-23-222-442
```

The log file stores the triple (x/38-23-222-442/200) where the first argument is the parameter of the object, the second the OID and the third the new value of the parameter.

The modus operandi for communicating the changes is like the following. A local site gains an exclusive, global lock on the file and updates the required objects. In parallel the log file is written. The file itself is transferred to remote sites with an efficient file transfer protocol whereas the remote sites are notified via control messages.

Since such a replication policy is rather cost intensive in terms of exchanging communication messages sending data, it should only be applied to a relatively small amount of data. In the HEP environment, there exist many meta data sources like replica catalogues, indices etc. which require a high consistency of data. For such data this approach is useful.

8 Conclusion

The data management efforts of the two research communities distributed databases and Grid deals with the problem of data replication where the Grid community specifically deals with large amounts of data in wide area networks. In the HEP community, data are often stored in database management systems, and it is appropriate to try to understand the research issues of both communities: distributed databases and Grid; analyse differences and commonalities, and combine common ideas to form an efficient Data Grid. We have presented research issues and possible solutions. Thus, we provide a first basis for the effort of combining both research communities.

Acknowledgement

We want to thank colleagues from the following groups for fruitful discussions: DataGrid work package "Data Management" (including CERN, Caltech, LBL and INFN), CMS Computing group (CERN, Princeton and Caltech), Globus project in Argonne and ISI, BaBar experiment at SLAC and colleagues taking part in the Data Grid discussions in the Grid Forum 5 in Boston.

References

[1] The European DataGrid Project: http://www.cern.ch/grid/

[2] Wolfgang Hoschek, Javier Jaen-Martinez, Asad Samar, Heinz Stockinger, Kurt Stockinger, Data Management in International Data Grid Project, *1st 1EEE, ACM International Workshop on Grid Computing (Grid'2000)*, Bangalore, India, 17-20 Dec. 2000.

[3] The Globus Project, http://www.globus.org

[4] Objectivity Inc., http://www.objectivity.com

[5] Heinz Stockinger, Kurt Stockinger, Erich Schikuta, Ian Willers. Towards a Cost Model for Distributed and Replicated Data Stores, *9th Euromicro Workshop on Parallel and Distributed Processing PDP 2001*, IEEE Computer Society Press, Mantova, Italy, February 7-9, 2001.

[6] Particle Physics Data Grid (PPDG), http://www.ppdg.net

[7] GriPhyN, http://www.griphyn.org

[8] Koen. Holtman, Peter van der Stok, Ian Willers. Towards Mass Storage Systems with Object Granularity. Proceedings of *the IEEE Mass Storage Systems and Technologies*, Maryland, USA, March 27-30, 2000

[9] Koen Holtman, Heinz Stockinger, Building a Large Location Table to Find Replicas of Physics Objects, *Proc. of Computing in High Energy Physics (CHEP 2000)*, Padova, Febr. 2000.

[10] D.K. Gifford. Weighted Voting for replicated data. *ACM-SIGOPS Symp. on Operating Systems Principles*, Pacific Grove, December 1979.

[11] Kurt Stockinger, Dirk Duellmann, Wolfgang Hoschek, Erich Schikuta. Improving the Performance of High Energy Physics Analysis through Bitmap Indices. *11th International Conference on Database and Expert Systems Applications*, London - Greenwich, UK, Springer-Verlag, Sept. 2000.

[12] L. M. Bernardo, A. Shoshani, A. Sim, H. Nordberg. Access Coordination of Tertiary Storage for High Energy Physics Applications. *IEEE Symposium on Mass Storage Systems*, College Park, MD, USA, March 2000.

[13] Yuri Breitbart, Henry Korth. Replication and Consistency: Being Lazy Helps Sometimes, *Proc. 16 ACM Sigact/Sigmod Symposium on the Principles of Database Systems*, Tucson, AZ 1997.

[14] Versant, Inc. http://www.versant.com/

[15] ObjectStore http://www.exceloncorp.com /products/objectstore.html

[16] Divyakant Agrawal, Amr El Abbadi, R. Steinke: Epidemic Algorithms in Replicated Databases (Extended Abstract). *PODS 1997*, 1997.

[17] Asad Samar, Heinz Stockinger. Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication, *IASTED International Conference on Applied Informatics (AI 2001)*, Innsbruck, Austria, 2001.

[18] J. Postel, J. Reynolds, RFC 959: File Transfer Protocol (FTP), October 1985.

[19] Globus Project, Universal Data Transfer for the Grid, White Paper, 2000.

[20] Joshua P. MacDonald, XDelta, http://www.XCF.Berkeley.EDU/~jmacd/xdelta.html

Contact Sheet

Heinz Stockinger
CERN
CMS Experiment, Computing Group
Bat. 40-3B-15
CH-1211 Geneva 23
Switzerland

Heinz.Stockinger@cern.ch
tel +41 22 767 16 08
fax +41 22 767 89 40