Toward Replication in Grids for Digital Libraries with Freshness and Correctness Guarantees*

Fuat Akal, Heiko Schuldt, and Hans-Jörg Schek

University of Basel, Department of Computer Science
Bernoullistr. 16, 4056 Basel, Switzerland
{fuat.akal,heiko.schuldt}@unibas.ch
schek@inf.ethz.ch

Abstract. Building Digital Libraries (DL) on top of data Grids while facilitating data access and minimizing access overheads is challenging. To achieve this, replication in a Grid has to provide dedicated features which are only partly supported by existing Grid environments. First, it must provide transparent and consistent access to distributed data. Second, it must dynamically control the creation and maintenance of replicas. Third, it should allow higher replication granularities, i.e., beyond individual files. Fourth, users should be able to specify their freshness demands. Finally, all these tasks must be performed efficiently. This paper presents an on-going work with the ultimate goal of building a fully integrated and self-managing replication subsystem for data Grids which will provide all above features. Our approach is to start with an accepted replication protocol for database clusters and to adapt it to the Grid.

1 Introduction

Digital Libraries (DLs) have significantly evolved during the last years. They are no longer seen as just the digital counterparts of traditional libraries but are rather considered to be the main platforms for managing and sharing data and knowledge within communities. This poses new requirements on DL systems. In particular, DLs have to deal with a rich variety of data ranging from traditional documents to complex multimedia objects which combine text and audio-visual content, or even sensor data streams. In parallel, also the requirements in terms of access to data and documents in a DL have become more challenging, leading from simple Boolean queries based on structured metadata (e.g., date and location where a certain photo has been taken) to sophisticated multi-object multi-feature similarity queries based on automatically extracted object features (e.g, image and audio features of a video sequence). This leads to an increased use of DLs and the expansion to new user communities like eScience and eHealth [11, 12]. These new communities demand efficient and consistent access to DL content and increased availability of data and metadata managed in DLs.

* The work presented in this paper has been partly supported by the EU in the 6^{th} framework programme within the project DILIGENT (contract No. IST-2003-004260).

DILIGENT (a DIgital Library Infrastructure on Grid ENabled Technology) [4] is an ongoing EU-funded project that combines Grid and DL technologies. The DILIGENT system exploits the gLite [7] middleware provided by the EGEE (Enabling Grid for E-sciencE) Project [5]. One of the application domains addressed in DILIGENT is Earth Observation (EO). A typical activity in EO, as it is performed by the European Space Agency (ESA), is to generate periodical environmental reports. These reports are generated by using a complex information object model provided by the DILIGENT system. This model allows, for instance, to associate satellite images with several types of metadata, e.g., spatio-temporal metadata, image features, etc. Such reports should be kept as up-to-date as possible and require seamless access to different heterogeneous resources as well as processing huge amounts of data. While the gLite middleware provides basic data Grid functionality, the support for replication is rather limited. Basically, replication on gLite consists of the manual creation of replicas for files, i.e., for raw content only. Neither freshness characteristics of replicas (i.e., when has the replica been updated) nor consistent access to different replicas (i.e., guarantee that different objects have the same freshness level) is provided.

This paper presents ongoing activities within the context of the DILIGENT project that aim at combining data Grids with sophisticated replication protocols which have been developed in the context of distributed databases and database clusters. The objective is to build an advanced data Grid infrastructure that first is able to dynamically self-determine where and when to create (and also delete) replicas. Second, efficient and transparent access to replicas is provided. This includes the selection of the best replica of a file based on a cost metrics that considers network and storage latencies. Third, the replication system should support higher granularities, e.g., collections or sub-collections, which in turn will improve the efficiency and the manageability of the replica management. Fourth, users should be allowed to specify their freshness requirements while accessing data. Freshness measures up-to-dateness of data. Several freshness metrics can be found in literature [16]. The freshness level demanded by an application can be thought of a quality of service parameter. Depending on the user's freshness requirements, e.g., "give me at most thirty minutes old data", finding the best replica may not need to suffer from synchronization of replicas. Involving freshness level of data in the replica selection process provides a flexibility with the user to make his choice. Finally, maintenance of replicas must be performed efficiently even with high freshness and consistency requirements. Since synchronous replication is very expensive to afford in a Grid environment, a mix of replication mechanisms which comes with consistency and efficiency guarantees of synchronous and asynchronous replication models must be provided for efficient maintenance of replicas.

The remainder of this paper is organized as follows: Sections 2 and 3 give an overview on replication management in Grid environments and in database clusters, respectively, together with related work. Section 4 presents the transition from database clusters to data Grids. Section 5 details our envisioned replication architecture. Section 6 concludes.

2 Replication in the Grid

Typical replication solutions for the Grid have the tendency of using a simple user-initiated replication model. In this model, typically a central (also distributed) file catalog exists where all replicas of a file are registered. This model pursues an *upload-replicate-download* cycle. The European DataGrid (EDG) [6] and the Enabling Grids for E-sciencE projects [5] are the known examples of this model. EDG provides a Grid middleware which contains a set of integrated data management services. It provides the Replica Location Service (RLS) which is a joint design between the Globus Alliance [8] and the EDG Project. It is used to maintain and provide access to information about the physical locations of the replicas. The RLS can be setup in a distributed manner by using as many components as desired, which may lead inconsistencies. EDG Replica Management Service provides a set of command line programs, e.g., edg-create-replica command, to manage replication. It also provide the Replica Optimization Service (ROS) whose objective is to monitor network and storage elements to facilitate the selection of the best replica by taking into account the network and storage latencies. Although the ROS is a big step to provide dynamic replication schemes, this component is not mature enough to enable making complex decisions as mentioned in Section 5. Besides, the EDG project is completed and the activities on EDG Replica Management are stopped.

This model suffers from two shortcomings. Firstly, creating a replica of a file is the choice of a user. The user decides on which node the replica is to be created. Another decision taken by the user is the number of the replicas. The user can create as many replicas as he desires. Although there is no theoretical limits on the number of the replicas to be created, keeping this number uncontrolled is not practical. Secondly, accessing a specific file (or one of its replicas) within a grid job is a static information given by the user. The user simply specifies which copy of the file to use, and he gives its physical address to the grid job for fetching the file.

Globus extends this model. It provides two types of solutions. These are *i)* core services, e.g., Replica Location Service (RLS) [3] which is a distributed mechanism for keeping track of the locations of replicated data on a Grid, and *ii)* higher-level services which are built atop the core services, e.g., Data Replication Service (DRS). The DRS was built on top of the Globus Replica Location Service and the Reliable File Transfer Service [9]. The ultimate goal of Globus is to provide more general, configurable and higher-level data management services which contain integrated replica management functionality. However, building higher-level replication mechanisms is a responsibility of the application developers.

For most of the available solutions, the replication is based on single files. Only few solutions provide replication on more abstract granules, e.g., replication based on user-defined collections of Grid files. The Storage Resource Broker (SRB) proposed by the San Diego Supercomputer Center is one of the most popular and comprehensive Data Grid Management systems [15]. SRB is a client-server middleware that provides applications with a uniform interface to access various distributed storage resources. SRB provides several features regarding replication

including physical aggregation in containers which corresponds to replication at higher granularity. However, it does not provide a dynamic replica management functionality as described at the beginning of this section.

Last but not least, a general opinion is that a Grid application does not have to care much about the consistency and the freshness of data. This opinion holds true for some scientific fields. For instance, consistency and currency of data may not be important for a physicist, for whom the first Grid solutions were developed. On the other hand, freshness of data may be crucial for a researcher, for instance, from the EO community. An example application in the EO domain may be a monitoring of the environmental changes in the Mediterranean Sea. Freshness of data to be processed by the application may be crucial for identifying oil spills rapidly and for taking necessary precautions before the damage to nature becomes disastrous.

3 Replication in a Database Cluster

Replication management in database clusters has come of age today. There are several well-established protocols in the database literature. Basically, there are two approaches to database replication management: *eager* and *lazy* [10]. *Eager* replication synchronizes *all* copies of an object within the same database transaction. *Lazy* replication management, on the other hand, maintains the replicas by using decoupled propagation transactions which are invoked after the "original" transaction has committed. Previous work on lazy replication focuses only on performance and correctness. In our previous work, we proposed the PDBREP protocol for database clusters which also considers freshness issues which is a property that is not necessarily satisfied by conventional lazy replication techniques [1]. It combines the performance benefits of lazy protocols with the up-to-dateness of eager approaches.

An overview of the PDBREP architecture is illustrated in Figure 1. PDBREP assumes a coordination middleware built atop a database cluster. All access to the cluster is done through this middleware which divides the nodes into two classes: (1) read-only and (2) update. Update nodes hold the primary (updateable) copies of the data objects while the secondary copies (read-only replicas) reside on the read-only nodes. Updates of a data object first occur at its primary copy, then they are propagated to secondary copies [2].

The changes of update transactions that occur at update nodes are serialized [2] and logged in a *global log*. We assume that the serialization order of the updates is equal to their commit orders. These changes are continuously broadcasted to all read-only nodes in the system and are enqueued in the local update queues. The broadcast is assumed to be reliable and preserves the global FIFO order.

Queries read data objects only from read-only nodes. PDBREP allows distributed executions of the read-only transactions. Moreover, it allows for arbitrary physical data organizations at the read-only nodes. This allows further classification of the read-only nodes as node groups which are customized for certain query types. The coordination middleware routes an incoming query to

a customized node group where it can be executed faster. As another important property of the PDBREP protocol, queries may specify the freshness of data they want to read. Figure 1 shows a query Q that wants to read data objects a and b with a minimum freshness requirement of *fr*. Some users may survive with stale data, e.g. *"fr:I am fine with 2 minutes old data"* while others may demand the freshest data, e.g. *"fr:I want fresh data"*.

When a query requests a younger version of a data object than the version actually stored at a read-only node, a *refresh transaction* is invoked in order to bring the node to the freshness requirement specified by the query. Figure 1 shows two refresh transactions invoked by Q on-demand on the node group NG_1. A refresh transaction first checks the local update queue to see whether all changes up to the required freshness level are already there. If yes, it fetches these changes from the local update queue and applies them to the database in a single bulk transaction. Otherwise, it retrieves all available changes in the local update queue and goes to the global log for the remaining part to create the bulk refresh transaction. After completing the refresh, the query is executed.

In order to improve the freshness of nodes, PDBREP uses another mechanism called *propagation transactions*. Propagation Transactions are used during the idle time of a node for propagating the changes accumulated in the local update queues to the secondary copies. Therefore, propagation transactions are continuously scheduled as long as the node is idle. Figure 1 illustrates propagation transactions scheduled in NG_2. The rationale behind this is to keep the secondary copies at the read-only nodes as up-to-date as possible such that the work of refresh transactions (whenever needed) is reduced and thus the performance of the overall system is increased. All updates are applied at the read-only nodes in the commit order of the global log.

We introduced a new locking scheme called *freshness locking* to guarantee correct executions of queries together with refresh and propagation transactions. Freshness locking allows to keep the data objects accessed by a query at a certain freshness level during the execution of that query. Further information regarding how the freshness locking is performed and the details of the PDBREP protocol can be found in [1].

4 Transition from a Database Cluster to the Grid

The Grid provides a powerful storage infrastructure. Our aim is to equip this infrastructure with a powerful replication management mechanism which was proven for database clusters. For this purpose, we propose a Grid replica management architecture that combines existing Grid data management components with PDBREP. In order to adapt the PDBREP protocol for the Grid environment, some of its basic assumptions have to be revisited.

Increased Number of Update Nodes: PDBREP assumes the existence of a global log component in which all update operations at the update nodes are somehow serialized. By nature, the Grid may contain several update nodes. In order to ensure consistent executions of distributed queries, all update operations

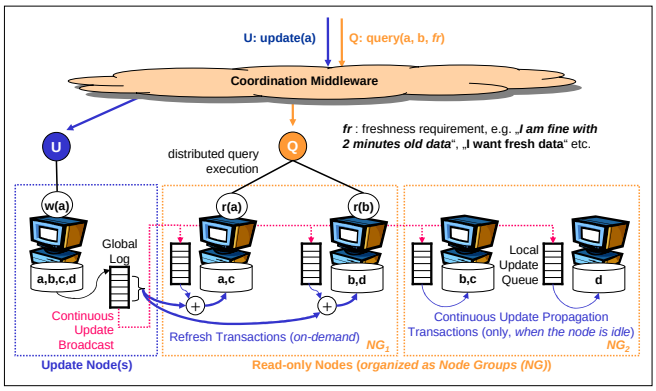

Fig. 1. Overview of PDBREP

must be serialized and be reflected on the replicas according to the serialization order. The solution proposed in [2] can be adapted. Note that, no Grid solution supports many update nodes with serialization guarantees yet.

Replacing Broadcasting by a Publish/Subscribe Model: Since the nodes of a database cluster are connected with either a LAN or high performance interconnects, PDBREP uses reliable broadcasting of updates that occur at the update nodes to the replicas. The Grid, on the other hand, consists of nodes which are potentially distributed worldwide, which may make use of broadcasting impractical. Therefore, the broadcasting mechanism of PDBREP must be modified to carry updates to the replicas in the wide-area network. A suitable mechanism is a publish/subscribe model where a replica node subscribes for changes on its originating update node. This approach can reduce the network traffic and allows forwarding of changes only to the related replicas by enabling filtering.

Flexible Data Allocation Schemes: PDBREP supports arbitrary physical layouts of data. This property becomes crucial for data Grids if the replication granularity is wanted to be moved beyond single files. For instance, a collection of documents can be divided into partitions (sub-collections) and those partitions can be replicated separately.

Bulk Operations: PDBREP heavily uses bulk operations to improve the overall performance. Bulk operations can also be used when moving relatively small files around the Grid. Typically there is some bookkeeping performed by file transfer utilities before the transfer starts. This bookkeeping results in longer response times. Therefore, a natural way to transfer many small files is to package them as a bulk and transfer it at once.

Implementation: Since Web services provide a standard and language independent way to access services, WSRF seems to be a well-suited choice to access heterogeneous data held in data Grids. Therefore, the implementation of the new protocol will be compliant with the Service-Oriented Architecture (SOA) paradigm. In addition, available components will be exploited as black-box Grid components wherever it is possible.

5 Overview of the Diligent Replication Architecture

An important challenge DILIGENT aims at facing is that a DL requires high performance especially while searching and browsing content. Therefore, DILIGENT builds further functionality on top of the underlying gLite middleware. For instance, it distinguishes content, interrelates content in multiple ways, and allows content to be described with various application specific properties. Such additional functionality requires content management facilities which are capable of dealing with generic *information objects*, i.e., units of information that can be stored or fetched independently of what they actually represent, e.g. collection, content, metadata, etc. This allows DILIGENT to be flexible in terms of documents it can handle. That is, by using an information objet linking mechanism, complex document types can be supported to store in a DL.

In order to achieve this, DILIGENT provides a data management architecture consisting of three layers as sketched in Figure 2. The Content Management Layer provides the highest level of abstraction and the appropriate interfaces for the tasks of storing, retrieving, and organising information. The Storage Management Layer provides functionality to store the information objects, e.g., documents. The Base Layer encapsulates the low level details of implementing physical storage and file transfer. The Replication Service sits in the core of the Base Layer and orchestrates storage nodes scattered in the Grid.

A storage node is a Grid node which hosts data sets (replicas) and required components to manage data replication. A data set is a unit of replication and can be as small as a single file or a collection of files. Using data sets allows logically grouping of files and more flexible replication granules. Each storage node is equipped with four components. The serialization order of the updates is enforced on the replicas by using update queues. Replica Management Service (RMS) allows to manage replicas, e.g., create, delete, list replicas etc. Replica Selection Service (RSS) finds the appropriate replica, e.g., bestReplica via its cost estimation module and maintains access history for dynamic replication decisions (e.g., replicating data which is frequently accessed). The File Transfer Service based on the GridFTP protocol is used for moving data around the Grid.

A typical replication activity in DILIGENT can be illustrated as in Figure 2. When a change request to a document is received (1) at the Content Management Layer, it is passed to the Base Layer (2). This request is routed to a proper storage node which holds the updateable copy of the document (3). The request is then executed by the storage node (4). To ensure consistency, any change in the data must be propagated to all replicas (5). Furthermore, the DILIGENT system detects any changes on the content via a notification mechanism. Whenever a change occurs, a notification for it is generated (6). This notification is then consumed by external services (7). In this way, changes are propagated actively and the derived data is kept as up-to-date as possible which is an added value of DILIGENT unlike the conventional approaches where refreshing of derived data may take longer.

Figure 3 shows further details of the DILIGENT replication architecture. Primary (updateable) copies of the data sets are shown within a dotted circle in

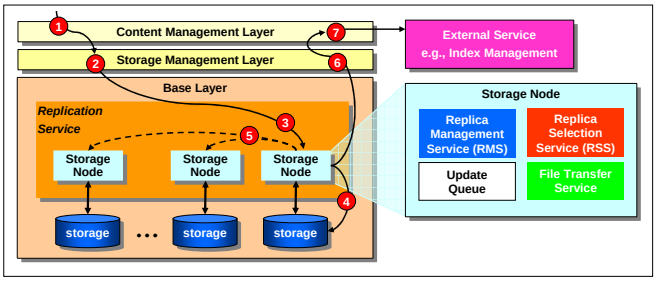

Fig. 2. Overview of Data Management in Diligent

which the serialization of the updates into the update queue occurs. In order to avoid clutter, update nodes are depicted close to each other. But an update node can be anywhere in the Grid. A typical log of an update operation can be modelled as a triple of TS, $writeOPs$ and DS, which correspond to the timestamp of the update, performed operations and the data set to be changed respectively.

Information regarding content of the storage nodes is available through the replica catalog which is used to locate the required data set. In order to read a certain file, the client has to go to the storage node where the data set which holds that specific file exists. For instance, the replica catalog in the figure indicates that data set DS_2 is available on both storage nodes 2 and 3.

Storage nodes subscribe for changes on the data set they hold. Continuous propagation of updates occurs only to the subscribing storage nodes for the subscribed data set. The scheduling of the propagation transactions as well as the refresh transactions remain the same as presented in Section 3. However, we can introduce additional components to facilitate scheduling of read-only transactions in the Grid:

i) freshness repository: It can be used to periodically collect freshness levels of the data sets lying around the Grid. In the figure, the last known (collected) freshness level of data set DS_3 on storage node SN_5 is equal to 0.6. These approximate freshness levels are used to select the data sets from which the required file will be read.

ii) load repository: It is used to determine an approximate load information regarding each storage nodes. This information can then be used to balance the load while routing read-only transactions to the storage nodes. Various metrics can be exploited to measure the load of a storage node. A simple metric is the CPU load as presented in Figure 3 where the load of the storage node SN_3 is 60%.

Introducing such components is indeed not a new invention. They rather can be found in the Peer-to-Peer process management literature already [14]. Similar to our idea of applying a technology that was originally devised for databases to the Grid, using peer-to-peer techniques to improve replication and read-only transaction scheduling perfectly fits into a Grid environment where the generalized PDBREP is applied.

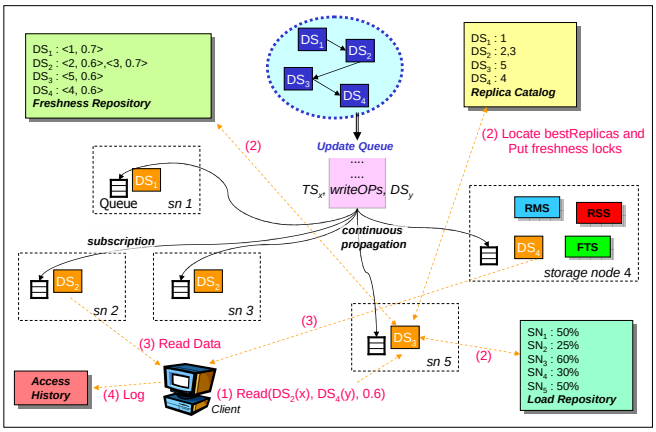

Fig. 3. Diligent Replication Architecture

Similarly, existing Grid monitoring tools can also be used to play the role the additional components. For instance, R-GMA can be used to collect information regarding storage devices in the Grid [13]. After all, the whole idea in DILIGENT is not only to invent new monitoring components like freshness repository but also use the available ones as black boxes to build a management architecture.

When a user requests a copy of a file which resides in the Grid, the replication system should return the best replica to the user. The selection of the best replica can be based on one or a combination of many factors. For instance, a typical best replica selection for the available replication solutions is based on the closeness of a replica to the requesting user's location in terms of network transfer cost. However, this may not necessarily result in the most efficient access to the replica of that file since the combination of selection factors may change the decision. Such an implementation of replica selection based on more complex calculations may improve the access performance. For instance, picking up the least loaded storage node to fetch a file among two storage nodes which hold the replicas of that file and are at the same distance from the user's location will definitely lead to faster access. For instance, the client in Figure 3 wants to read the files x and y in the data sets DS_2 and DS_4 respectively. It specifies its freshness requirement as 0.6, which means the freshness of data to be read can be 0.6 at least (1). According to their freshness levels and loads, the *storage nodes* 2 and 4 are chosen and freshness locks are put on those nodes (2). Then, the data is accessed and transferred to the client's side (3). All accesses are logged in order to be used for future decisions on managing replicas (4).

6 Conclusions

In this paper, we presented the first steps of our on-going work whose ultimate goal is to come up with a fully integrated and self-managing replication subsystem

for the Grid. In order to achieve this goal, we want to combine existing powerful replication mechanisms from database clusters with the almost unlimited storage capacities that can be found in data Grids. In particular, we integrate the PDBREP protocol with the gLite Grid middleware. Firstly, a large part of the functionality of PDBREP is already inline with what a possible Grid replication system should support as we sketched in section 1. Secondly, the infrastructure related assumptions of PDBREP can be modified easily. Third, additional components presented in the envisioned architecture to facilitate scheduling of read-only transactions can be included in the PDBREP without requiring major changes.

In future work, we aim at further extending the Grid-enabled replication management with support for the dynamic parallelization of complex activities that make use of the presence of several replicas and advanced resource reservation in the Grid.

References

1. F. Akal, C. Türker, H.-J. Schek, Y. Breitbart, T. Grabs, and L. Veen. *Fine-Grained Replication and Scheduling with Freshness and Correctness Guarantees.* In *VLDB*, pages 565–576, 2005.
2. Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, and A. Silberschatz. *Update Propagation Protocols For Replicated Databases.* In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Philadelphia, Pennsylvania, USA*, 1999.
3. A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. *Performance and Scalability of a Replica Location Service.* In *HPDC*, pages 182–191, 2004.
4. DILIGENT: A DIgital Library Infrastructure on Grid ENabled Technology. `http://www.diligentproject.org/`. IST-2003-004260
5. *The Enabling Grids for E-sciencE Project (EGEE).* `http://www.eu-egee.org/`.
6. *The European DataGrid Project.* `http://eu-datagrid.web.cern.ch/`.
7. *gLite: Lightweight Middleware for Grid Computing.* `http://glite.web.cern.ch`.
8. *The Globus Alliance.* `http://www.globus.org/`.
9. *The Globus Alliance, Reliable File Transfer Service (RFT).* `http://www.globus.org/toolkit/docs/4.0/data/rft/`.
10. J. Gray, P. Helland, P. O'Neill, and D. Shasha. *The Dangers of Replication and a Solution.* In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, Canada*, 1996.
11. Ioannidis, Y. Digital libraries at a crossroads. International Journal on Digital Libraries **5**(4), pages 255–265, 2005.
12. Moore, R.W., Marciano, R. Building preservation environments. In Proc. of the 5th ACM/IEEE-CS joint conference on Digital libraries (JCDL), 2005.
13. *Relational Grid Monitoring Architecture (R-GMA).* `http://www.r-gma.org/`.
14. Schuler C., Türker C., Schek H.-J., Weber R. and Schuldt H. *Scalable Peer-to-Peer Process Management.* In International Journal of Business Process Integration and Management (IJBPIM) 1(2) : 129-142, 2006.
15. *The Storage Resource Broker.* `http://www.sdsc.edu/srb/`.
16. H. Yu and A. Vahdat. *Design and Evaluation of a Continuous Consistency Model for Replicated Services.* In Proc. of the 4th Symposium on Operating System Design and Implementation (OSDI), 2000.