



ELSEVIER

Available online at [www.sciencedirect.com](http://www.sciencedirect.com)

SCIENCE @ DIRECT®

Nuclear Instruments and Methods in Physics Research A 534 (2004) 24–28

NUCLEAR  
INSTRUMENTS  
& METHODS  
IN PHYSICS  
RESEARCH  
Section A

[www.elsevier.com/locate/nima](http://www.elsevier.com/locate/nima)

## Replica consistency in a Data Grid

Andrea Domenici<sup>a,b</sup>, Flavia Donno<sup>b,c</sup>, Gianni Pucciani<sup>a</sup>, Heinz Stockinger<sup>c,\*</sup>,  
Kurt Stockinger<sup>c</sup>

<sup>a</sup>*DIIEIT, University of Pisa, v. Diotisalvi 2, 56122 Pisa, Italy*

<sup>b</sup>*INFN Pisa, Edificio C, Polo Fibonacci, Via F. Buonarroti, 2 Pisa, Italy*

<sup>c</sup>*CERN, European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland*

Available online 30 July 2004

---

### Abstract

A Data Grid is a wide area computing infrastructure that employs Grid technologies to provide storage capacity and processing power to applications that handle very large quantities of data. Data Grids rely on data replication to achieve better performance and reliability by storing copies of data sets on different Grid nodes. When a data set can be modified by applications, the problem of maintaining consistency among existing copies arises.

The consistency problem also concerns metadata, i.e., additional information about application data sets such as indices, directories, or catalogues. This kind of metadata is used both by the applications and by the Grid middleware to manage the data. For instance, the Replica Management Service (the Grid middleware component that controls data replication) uses catalogues to find the replicas of each data set. Such catalogues can also be replicated and their consistency is crucial to the correct operation of the Grid. Therefore, metadata consistency generally poses stricter requirements than data consistency. In this paper we report on the development of a Replica Consistency Service based on the middleware mainly developed by the European Data Grid Project. The paper summarises the main issues in the replica consistency problem, and lays out a high-level architectural design for a Replica Consistency Service. Finally, results from simulations of different consistency models are presented.

© 2004 Elsevier B.V. All rights reserved.

PACS: 89.20.Ff

Keywords: Grid computing; Simulation; Replication; Data consistency

---

### 1. Introduction

Several Grid data management middleware systems address replica management solutions that

are usually file based [1]. Often, replica consistency is not an issue when data is treated as read-only, which implies that inconsistency may only be due to system failures or accidental data corruption.

However, as Grid solutions are used by an increasing number of applications, requirements

---

\*Corresponding author.

E-mail address: [heinz.stockinger@cern.ch](mailto:heinz.stockinger@cern.ch) (H. Stockinger).

arise for mechanisms that maintain the consistency of replicated data and metadata that can change over time. An earlier work [2] reports on some models for such mechanisms and provides a basic background to the replica consistency service presented in this article. In short, the replica consistency problem deals with the update synchronisation of multiple copies (replicas) of a file: one file is updated and all other replicas then have to be synchronised in order to have the same contents and thus provide a consistent view.

In the database as well as in the distributed computing communities, several solutions exist already [3] that are only partially applicable to Grid solutions. In contrast to a distributed database management system, a Data Grid usually has to deal with heterogeneous data whereas a database management system has homogeneous, low-level access and full control of all data it manages. For example, a database management system provides transactions, read/write operations, locking, etc., which result in a fully consistent view on the data. In other words, a database user can rely on the database management system to keep all data consistent. In addition, the database management system provides all data manipulation features like insert, update, etc. In contrast, Data Grids often deal with file system access and thus have limited control over data consistency. One can read a file while another one can write into the same file: file systems usually do not provide strict transactions or locking of data, thus making inconsistencies possible. This is the basic challenge a consistency service has to face and the basic question is how to design and develop such a service that is compatible with the underlying Grid resource management. Other important differences between Data Grids and distributed databases are the very large number of files a Grid is expected to handle (~500 M replicas), the highly dynamic Grid configuration, and the need for scalability.

In this article, we outline a Replica Consistency Service (RCS) (Section 2), discuss a simulation tool that models the basic architecture (Section 3) and then present preliminary results of the simulator in Section 4.

## 2. Architecture

In the following section we first outline the basic service interface and then isolate the functionalities of the process of replica update synchronisation. Based on these main functionalities, an architecture for the consistency service with all the main components is described.

### 2.1. Overview

We start from the client's perspective and then go into the details of the update process. In general, the following functionalities are required:

*Client interface for the consistency service:* provides the basic operations invoked by clients of the service, that are typically other Grid services but may also be end-user applications.

*File update mechanism:* an important part of a consistency service is to update a file and apply the update to all its replicas. Several basic building blocks are required for such a task, including the File Update mechanism and the closely related Update Propagation protocol. The File Update mechanism is in charge of applying changes to a single replica.

*Update propagation protocol:* whereas the File Update mechanism takes care of a local update, an update propagation protocol is required that uses a transaction system for propagating updates to remote sites where replicas reside and then applies the File Update mechanism at the remote site. This protocol needs to be efficient over wide area networks and to take into account that a large number of replicas might exist.

### 2.2. Components of the consistency service

Based on the previous discussion, the following basic components are required for a consistency service:

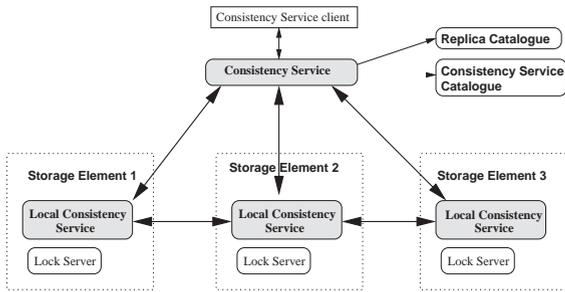


Fig. 1. Architecture of the consistency service.

**Consistency service:** is the service that provides the main entry point for a user via a consistency service client. Consequently, the Consistency Service needs to provide an interface to the end-user to update a file. However, the actual file update is then done by a Local Consistency service where the physical file resides.

**Transaction system:** for the update propagation protocol as well as for all other communication between components.

**Local consistency service (LCS):** that locally updates a file and then propagates the changes to remote sites using the Transaction System. The LCS needs to provide an internal interface to the Consistency Service to update a file and further provide an interface for other LCS to exchange update propagation requests.

**File lock service:** serialisation of file access (and in particular updates in a distributed system) requires that files are locked and thus access is denied to certain users. A Lock Service is required at each storage system and is responsible for holding and releasing locks for all files local to the storage system.

**Replica catalogue:** to retrieve the location of files in the storage systems.

**Replica consistency catalogue:** needs to store all meta data that is required for the replica update process. For instance, the catalogue needs to store file attributes like: master, state of a file (stale, up-to-date), and other information used by the RCS.

**Replica manager or File copier:** to transfer file updates to remote sites.

The interaction of the components involved in the Consistency Service is depicted in Fig. 1. For further design details, we refer the reader to [4].

### 3. Implementation

We have implemented a simulation tool that models the entire architecture discussed in the previous section. The tool is based on the Optor-Sim Grid simulator [5], extended with the Replica Consistency Service components. In this way, we can simulate job submission in a Grid environment together with replication and update synchronisation.

The simulator implements a *synchronous* model (all replicas are synchronously updated) as well as an *asynchronous* one (one replica is updated and the others are asynchronously modified at a later point in time) [2].

#### 3.1. Synchronous model

In the synchronous model we assume that the application uses a local replica and modifies it. Once the update has finished locally, all replicas are synchronised. However, as jobs may work on private replicas, conflicts may still occur. A job that meets a conflict must get a fresh replica and redo the operation.

#### 3.2. Asynchronous model

In [2] we presented several asynchronous replication models, and here we have selected a “single master approach” where only one replica can be modified by the end users and all others are synchronised (updated) by the RCS.

### 4. Experimental results

Simulations have been carried out to experiment with the two replica consistency models introduced

above. The Grid infrastructure we simulated consisted of 10 Storage Elements and 3 Computing Elements, with a simulated workload of 4 different user jobs requesting 5 logical files. In each simulation run 50 job instances were executed. A job issues requests for operations on *logical files* to a Replica Manager that finds or creates a suitable replica that the job can process.

Next, the job issues an update request to the RCS, using its replica as the update *source*. We use the term *logical* to stress the difference between logical files and physical replicas. Logical files and replicas have version numbers, and a replica is *stale* if its version is earlier than the one for the logical file. When a logical file is locked, all its replicas are locked as well.

The simulator provides the following data:

*read or write requests*: numbers of read ( $R_r$ ) and write ( $R_w$ ) requests for *logical files*;

*retried requests*: number  $R_{\text{retr}}$  of requests that have been re-issued at least once, due to conflicts (see below) or locks;

*conflicts*: number  $C$  of update requests having a stale replica as a source;

*logical file locks*: number  $L$  of update requests for locked files;

*stale read or write*: numbers of read ( $S_r$ ) and write ( $S_w$ ) requests for which the Replica Manager selects a stale replica.

The total number of requests on *replicas* (as opposed to *logical files*) is then  $T = R_r + R_w + C + L$ , and  $W = T - R_r$  is the number of write requests on replicas. We can then compute the following quantities:

*retried request rate*:  $R_{\text{retr}}/T$ ;

*stale read rate*:  $S_r/R_r$ ;

*stale write rate*:  $S_w/W$ ;

*logical file lock rate*:  $L/W$ ;

*conflict rate*:  $C/W$ ;

*conflicts/retried requests rate*:  $C/R_{\text{retr}}$ .

The simulations performed so far aimed at studying how the above rates vary with the frequency of write operations. With the simplifying assumption that all jobs have the same probability of issuing write requests, batches of simulation

runs have been executed, where in each batch the simulator was configured for a different write probability and a different consistency model. The averages across the simulation runs have then been plotted against the write probabilities. Figs. 2 and 3, for example, summarise the results for  $C/R_{\text{retr}}$  and  $R_{\text{retr}}/T$ . These two parameters tell us how often requests for logical files cannot be satisfied immediately, and how many times such requests result in version conflicts.

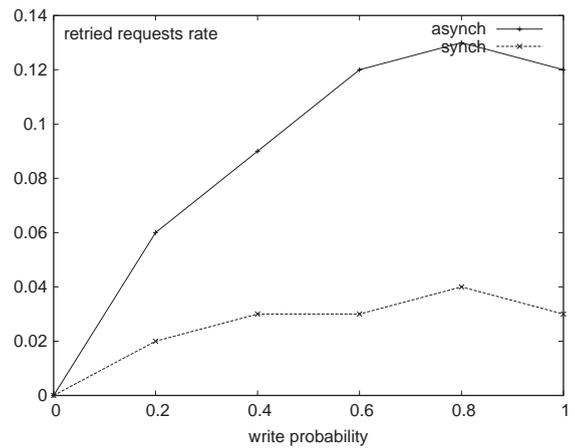


Fig. 2. Retried requests rate.

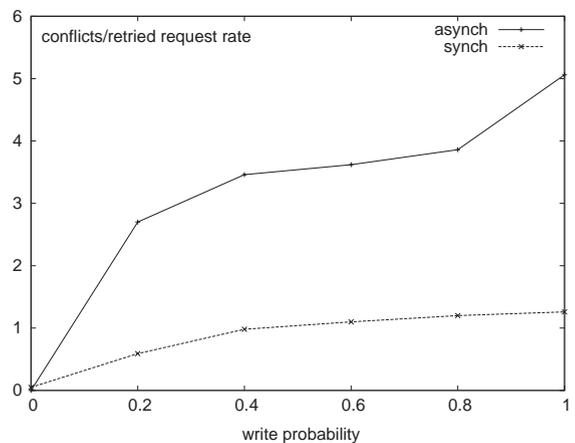


Fig. 3. Conflict/retried request rate.

## 5. Conclusions

We have presented an overview of the design of a RCS. Due to the complexity of the service we first studied its behaviour using simulation and then we will implement the service using a web service approach. Our simulation tool has outlined the basic behaviour, it shows promising results on what level of consistency can be gained, and provides valuable insights for the implementation of the service.

## References

- [1] P. Kunszt, E. Laure, H. Stockinger, K. Stockinger, Advanced replica management with rector, in: Fifth International Conference on Parallel Processing and Applied Mathematics, Czestochowa, Poland, September 7–10, 2003.
- [2] D. Düllmann, W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, K. Stockinger, Models for replica synchronisation and consistency in a data grid, in: Tenth IEEE Symposium on High Performance and Distributed Computing (HPDC-10), San Francisco, CA, August 7–9, 2001.
- [3] J. Gray, P. Helland, P.E. O’Neil, D. Shasha, The dangers of replication and a solution, in: SIGMOD Conference, Tucson, AZ, May 12–14, 1996.
- [4] H. Stockinger, A. Domenici, F. Donno, G. Pucciani, K. Stockinger, Replica Consistency Service (RCS)—design principles and basic architecture v0.3, Technical report—Draft, August 21, 2003.
- [5] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, F. Zini, Int. J. High Performance Comput. Appl. 17 (4) (2003).