Building Fault-Tolerant Consistency Protocols for an Adaptive Grid Data-Sharing Service

Gabriel Antoniu, Jean-François Deverge, Sébastien Monnet

▶ To cite this version:

Gabriel Antoniu, Jean-François Deverge, Sébastien Monnet. Building Fault-Tolerant Consistency Protocols for an Adaptive Grid Data-Sharing Service. Proc. ACM Workshop on Adaptive Grid Middleware (AGridM 2004), Sep 2004, Antibes Juan-les-Pins, France, France. inria-00000989v2

HAL Id: inria-00000989

https://hal.inria.fr/inria-00000989v2

Submitted on 11 Jan 2006

Building Fault-Tolerant Consistency Protocols
for an Adaptive Grid Data-Sharing Service

Gabriel Antoniu, Jean-François Deverge, Sébastien Monnet
IRISA/INRIA and University of Rennes 1
Campus de Beaulieu, 35042 Rennes, France
Contact: Gabriel.Antoniu@irisa.fr

Abstract

We address the challenge of sharing large amounts of numerical data within computing grids consisting of clusters federation. We focus on the problem of handling the consistency of replicated data in an environment where the availability of storage resources dynamically changes. We propose a software architecture which decouples consistency management from fault-tolerance management. We illustrate this architecture with a case study showing how to design a consistency protocol using fault-tolerant building blocks. As a proof of concept, we describe a prototype implementation of this protocol within JUXMEM, a software experimental platform for grid data sharing, and we report on a preliminary experimental evaluation.

1. Introduction

Data management in grid environments is currently a topic of major interest to the grid computing community. However, as of today, no sophisticated approach has been widely established for efficient data sharing on grid infrastructures. Currently, the most widely-used approach to data management for distributed grid computation relies on *explicit data transfers* between clients and computing servers. As an example, the Globus [12] platform provides data access mechanisms based on the GridFTP protocol [1]. Though this protocol provides authentication, parallel transfers, checkpoint/restart mechanisms, etc., it is still a transfer protocol which requires *explicit* data localization. On top of GridFTP, Globus integrates data catalogs [1], where multiple copies of the same data can be manually registered. Consistency issues are however at the user's charge. IBP [6], provides a large-scale data storage system, consisting of a set of buffers distributed over Internet. Transfer management is still at the user's charge and no consistency

mechanisms are provided for the management of multiple copies of the same data. Finally, Stork [17] proposes an integrated approach allowing the user to schedule data placement just like computational jobs. Again, data location and transfer are at the user's charge.

Within the context of a growing number of applications using large amounts of distributed data, we claim that *explicit management of data locations* by the programmer arises as a major limitation against the efficient use of modern, large-scale computational grids. Such a low-level approach makes grid programming extremely hard to manage. In contrast, we have proposed the concept of *data sharing service* for grid computing [3], which aims at providing *transparent access to data*. This approach is illustrated by the JUXMEM software experimental platform. The user only accesses data via a global identifier. The service handles data localization and transfer without any help from the programmer. However, it is able to use additional hints provided by the programmer, if any. The service also transparently uses adequate replication strategies and consistency protocols to ensure data availability and consistency. These mechanisms target a large-scale, dynamic grid architecture. In particular, the service supports events such as storage resources joining and leaving, or unexpectedly failing. This is the framework within which we conducted the study presented in this paper.

Handling consistency of replicated data. The goal of a data-sharing service is to allow grid applications to access data in a distributed environment. We are considering scientific applications, typically exhibiting a code-coupling scheme: e.g. multiple weakly-coupled codes running on different sites and cooperating via periodical data exchanges. In such applications, shared data are *mutable*: they can be read, but also *updated* by the different codes. When accessed on multiple sites, data are often replicated to enhance access locality. Replication is equally used for fault tolerance, since grid nodes may crash. To ensure that read

operations do not return obsolete data, consistency guarantees have to be provided by the data service. These guarantees are defined via *consistency models* and are implemented using *consistency protocols*.

Difficulty: handling consistency in a dynamic context.
The problem of sharing mutable data in distributed environments has intensively been studied during the past 15 years within the context of Distributed Shared Memory (DSM) systems [18, 20]. These systems provide transparent data sharing, via a unique address space accessible to physically distributed machines. When the nodes modify the data, some consistency action is triggered (e.g., invalidation or update), according to some consistency protocol. A large variety of DSM consistency models and protocols [7, 13, 15, 20, 23] have been defined, their role being to specify which remote nodes have to be notified of the modification, and when. They provide various trade-offs between the strength of the consistency guarantees and the efficiency of the implementation.

However, traditional DSM systems have generally demonstrated satisfactory efficiency (i.e., near-linear speedups) only on *small-scale* configurations: in practice, up to a few tens of nodes [20]. This is often due to the intrinsic lack of scalability of the algorithms used to handle data consistency. Most of the time, they rely on global invalidations or global updates of all existing data copies. On the other hand, an overwhelming majority of protocols assume a *static* configuration where nodes do not disconnect nor fail. It is clear that these assumptions do not hold any more in the context of a *large-scale*, *dynamic* grid infrastructure. Faults are no longer exceptions, but they become part of the general rule; resources may become unavailable and eventually become available again; finally, new resources can dynamically join the infrastructure. In such a context, consistency protocols cannot rely any more on entities supposed to be stable, as traditionally was the case. A new approach to their design is definitely necessary, to integrate these new hypotheses.

This idea is at the core of the design of our grid data-sharing service [3], which we have defined as a hybrid system inspired by DSM systems (for transparent access to data and consistency management) and P2P systems (for their scalability and volatility-tolerance). In this paper, we propose an approach allowing *consistency protocols* to take into account *fault tolerance* by decoupling the management of these two aspects. The motivations and the general principles are presented in Section 2. In Section 3 we describe the detailed architecture and we show how to use traditional group communication components of fault-tolerant distributed systems [11, 19] as building blocks for consistency protocols. We illustrate the approach in Section 4,

with a case study explaining the design of a fault-tolerant consistency protocol. Section 5 shows how this protocol has been implemented in the JUXMEM platform and presents a preliminary experimental evaluation. Some concluding remarks and future directions are given in Section 6.

2. Approach: decoupling fault tolerance management from consistency management

Let us first note that both fault tolerance mechanisms and consistency protocols are traditionally implemented using *replication*. However, the underlying motivations are totally different for each of the two uses.

Replication in consistency protocols. Consistency protocols use data replication for *performance* issues, to allow multiple nodes to read the same data in parallel via local accesses. However, when a node modifies a data copy, the consistency protocol is activated, e.g. the other copies must be updated or invalidated, to prevent subsequent read operations from returning invalid data. Note that P2P systems also use replication to enhance access locality, but most of them do not address consistency issues, since data is generally immutable.

Replication for fault tolerance. Replication is also commonly used by fault-tolerance mechanisms [21] to enhance *availability* in an environment with failures. When a node hosting a data copy crashes, other copies can be made available by other nodes. Various replication strategies have been studied [14], leading to various trade-offs between efficiency and the level of fault-tolerance guaranteed.

In distributed systems where both consistency *and* fault-tolerance need to be handled, replication can be used with a double goal. Consequently, depending on whether these two issues are addressed separately or not, two architectural designs are possible.

Integrated design. A possible approach consists in addressing consistency and fault tolerance at the same time, relying on the same set of data replicas. For instance, data copies created by the consistency protocols to enhance data locality can serve as backup if crashes occur. Conversely, backup replicas created for fault tolerance can be used by the consistency protocol. This approach has a major disadvantage: the design of the corresponding software layer is very complex, as illustrated by some fault-tolerant DSM systems [16, 22].

Decoupled design. A different approach consists in designing the consistency protocol and the fault-tolerance mechanism separately. This approach has several features. First, the design of consistency protocols is simplified, since the protocols do not have to address fault tolerance issues at a low level. Therefore, it is possible to leverage existing consistency protocols. Only some limited interaction between the consistency protocol and the fault-tolerance mechanism needs to be defined (see Section 3.2). Second, consistency protocols and fault-tolerance strategies can be developed independently. This favors a cleaner design, each of the two components being dedicated to its specific role. Finally, this approach provides the ability to experiment multiple possibilities to couple various consistency protocols with various fault-tolerance strategies.

The goal of this paper is to discuss how to manage consistency and fault tolerance at the same time, in a *decoupled* way, using this second approach.

3. Using fault-tolerant components as building blocks for consistency protocols

Traditional consistency protocols for DSM systems rely on stable entities in order to guarantee that data accesses are correctly satisfied. For instance, a large number of protocols associate to each data a node holding the most recent data copy. This is true for the very first protocols for sequential consistency [18], but also for recent *home-based* protocols implementing lazy release consistency [23] or scope consistency [15], where a *home node* is in charge of maintaining a reference data copy. It is important to note that these protocols implicitly assume that the home node never fails.

Such an assumption cannot be made in a dynamic grid environment, where faults may occur. In such a context, the role of home node has to be played by an entity able to transparently react to faults and disconnections, in order to maintain a given degree of availability for the reference data copy. We propose to design such entities using some basic building blocks that have been defined within the context of fault-tolerant distributed systems [11, 19]: group membership protocols, atomic multicast, consensus, etc. We briefly introduce these blocks in Section 3.1. Then, Section 3.2 describes the "glue layers" through which the consistency protocol interacts with these fault-tolerant blocks.

3.1. Basic fault-tolerant components

We consider that two types of faults need to be addressed in a grid environment. First, nodes may crash, i.e. nodes act normally (receive and send messages according to their specification) until they fail (crash failures). Second, we assume messages can be delayed or lost (omission failures). We consider two main timing aspects: the communication delays and the computation times. We make the assumption that upper bounds upon these times exist but are not known. Classical fault-tolerance mechanisms are often built on these hypotheses, which are realistic in a grid context.

The *group membership* abstraction [11] is such a mechanism providing the ability to manage a set of nodes running the same service. In our case, it applies to a group of nodes that together play the role of home node. Requests sent to the group need to be delivered in the same order to all group members. This property is commonly called *atomic multicast*. As members of the group have to agree upon an order for message delivery, atomic multicast can be built using a *consensus* protocol. The consensus problem in asynchronous systems can be solved thanks to *unreliable failure detectors* [10]. The role of these detectors is to provide to higher layers a list of nodes suspected to be faulty. The consensus protocol can cope with the approximate accuracy of the list contents.

These blocks can interact with each other in many ways. In this paper, we consider a layered, decoupled design (Figure 1), inspired by [19]. Here, the *adapter* module allows higher-level software layers to register to the failure detection service and to filter the list of suspected nodes according to some user-specified quality of service, as in [8].
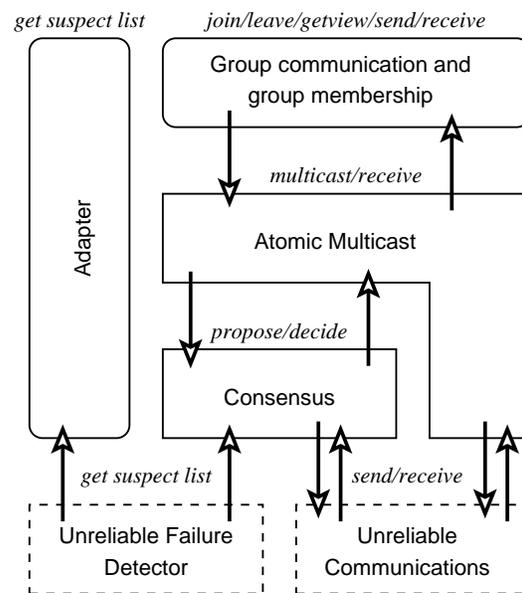

Figure 1. *An architecture for group communication and group membership protocols.*

3.2. A decoupled architecture: the big picture

Our idea is to use the abstractions described above to build fault-tolerant entities able to play the role of critical entities in consistency protocols. For instance, each *home node* can be replaced by a group of nodes handled via a group membership interface and supporting atomic multicast. However, some actions like *1) group self-organization* or *2) configuration of new group members* need to be handled by higher-level layers. Such actions are not necessarily specific to consistency protocols (i.e. they can apply to several consistency protocols). They are situated precisely at the "boundary" between fault-tolerance management and consistency management. Hence the need to introduce two interface layers in our architecture, as shown in Figure 2.
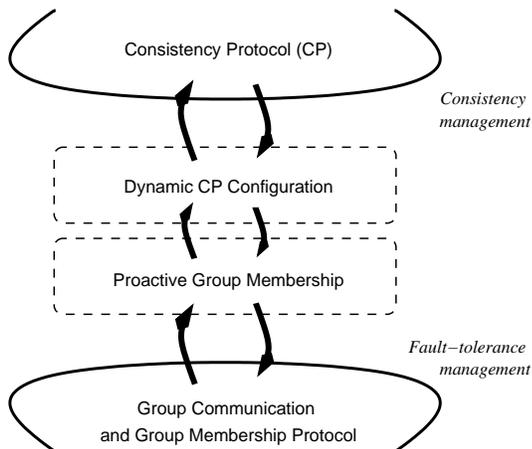

Figure 2. *Decoupled architecture for managing consistency and fault tolerance.*

The *Proactive Group Membership* layer handles the composition of a group of nodes that together act as a home node. The layer decides when to remove from the group nodes reported to be faulty, by parameterizing the QoS of the failure detector. It also removes nodes that notify about their future disconnections. Following such removals, the layer adds new members to the group, to maintain the availability of the home node. To do so, it takes into account constraints specified at allocation time: the necessary memory size, the network performance, or the replication policy (expressed in terms of number of clusters where to spread data replicas, number of replicas per cluster, etc.). Various trade-offs could be expressed at this level (e.g. smaller group sizes to enhance communication efficiency vs. larger group sizes to increase the level of fault tolerance).

When some new node is added to the group that acts as a home node, the newcomer has to initialize his state in order

to be consistent with the state of the other members of the group. The *Dynamic Consistency Protocol Configuration* layer defines how to instantiate a consistency protocol on such nodes. The new node must first take into account the configuration messages generated by the other members of the group at the level of this layer, before reacting to external messages addressed to the group.

4. Case study: designing a hierarchical, fault-tolerant consistency protocol

The typical grid applications we target are loosely code-coupling applications, in which several codes run in parallel on different clusters and iteratively exchange data. These data exchanges can be carried out through read or write accesses to a data-sharing service, such as JuxMem [3]. The role of this service is to ensure consistent access to shared data, while transparently handling failures or volatility events. This is where fault-tolerant consistency protocols relying on the approach proposed in Section 3 are useful. As an illustration of this idea, this section describes how to build such a protocol starting from a *non fault-tolerant* protocol implementing the *entry consistency model*.

4.1. The entry consistency model

Previous experience with DSM consistency protocols has shown that relaxed consistency models can be implemented via efficient protocols at the price of restricted consistency guarantees. For instance, the programmer must use synchronization operations, such as `acquire`, to make sure the subsequent accesses are correctly satisfied, and `release`, to allow the local modifications to be (eagerly or lazily) propagated to remote nodes. This general requirement is valid for models like release consistency [13], entry consistency [7] or scope consistency [15].

In this paper, we focus on the *entry consistency model*. As opposed to other relaxed models, it requires an explicit association of data to synchronization objects. This allows the model to leverage the relationship between a synchronization object that protects a critical section, and the data accessed within that section. A node's view of some data becomes up-to-date only when the nodes enters the associated critical section. This eliminates unnecessary traffic, since only nodes that declare their intention to access data will get updated, and only for the data which will be accessed. Such a concern for efficiency makes this model a good candidate in the context of scientific grid computing.

The programmer has to observe two main requirements. First, all shared data have to be associated with

at least one guarding synchronization object. Second, exclusive accesses to shared data have to be explicitly distinguished from non-exclusive accesses by using two different primitives: `acquire`, which grants mutual exclusion; `acquireRead`, which allows non-exclusive accesses on multiple nodes to be performed in parallel. A detailed description of the model is given in [7].

4.2. A hierarchical consistency protocol

Our starting point is a *non fault-tolerant* protocol for entry consistency. We consider a *home-based* protocol, in which a *home node* is associated to each data. This node is responsible for maintaining a reference copy for that data. The home node also manages a lock associated to its data. When a process enters a critical section protected by such a lock, the associated shared data is updated on the node hosting that process (if necessary). On leaving the critical section, the local modifications (if any) are transmitted to the home node. Consequently, accesses to shared data involve some communication with the home node.

Let us note that, in a grid consisting of clusters federation, inter-cluster networks get generally lower communications performances than intra-cluster networks. In order to improve the protocol efficiency, a suitable approach can rely on minimizing the inter-cluster communications. This idea has been used in some DSM systems and has lead to the design of *hierarchical* consistency protocols. In CLRC [5], local caches are created on each cluster, to optimize the locality of consecutive accesses to remote data modifications.

Let us now consider a *hierarchical* version of the protocol sketched out above. This version is very similar to the hierarchical, home-based protocol for release consistency described in [4]. The idea is to use a *two-level hierarchy of home nodes*. On each cluster, a *local home* will serve accesses from the local cluster, whereas a *global home* will serve data accesses to the clusters, i.e. to the local homes (Figure 3 (a)). When a client needs to access some data, it will require the associated lock to its local home. If this home owns the corresponding access rights to the data, it can satisfy the access. Otherwise, it will request the lock from the global home, with an updated copy of the data. Note that the global home only serves the requests issued by the local homes; it has no control on what requests are subsequently served by the local homes. However, to minimize inter-cluster communications, a local home serves local requests with higher priority than remote requests issued on other clusters, received via the global home. To avoid starvation, a limit is set on the number of consecutive accesses served by each local home, so that remote requests be served too. Details about this hierarchical lock management scheme are given in [4].

4.3. A *fault-tolerant* hierarchical protocol

In the protocol sketched out above, the local homes and the global home are clearly critical and must be available for the protocol to be operational. Since in a grid environment we cannot realistically assume that they can be implemented by failure-free nodes, this is where the approach proposed in Section 3 can be applied. Our proposal is to make these entities fault-tolerant using an enriched version of the *group membership* abstraction. Each local home is replaced by a set of nodes that we call *Local Data Group* (LDG). At a higher level, the global home is replaced by a *Global Data Group*, whose members are the LDGs (Figure 3 (b)). The GDG and the LDGs have the following properties: 1) All messages sent to such a group are received *by all members of the group, in the same order* (atomic multicast); 2) The groups are self-organizing: they maintain some user-specified replication degree by dynamically and adding new members when necessary in a "smart" way. The selection of the new members is handled by the *Proactive Group Membership* layer, whereas their initialization is managed by the *Dynamic Consistency Protocol Configuration* layer (as explained in Section 3.2).

The number of simultaneous faults supported by this solution depends on the implementation of the underlying fault-tolerant building blocks (consensus, atomic broadcast). Our current implementation supports up to $\lfloor n/2 \rfloor$ simultaneous failures within a LDG or GDG group, where n is the group size.

Note that the consistency protocol can use the GDG and the LDGs *exactly in the same way* it initially used the global and local nodes respectively. It assumes they are always available, but this property is now achieved transparently for the protocol, thanks to the implementation of the Proactive Group Membership abstraction. The consistency protocol and the replication-based fault-tolerance mechanism are thus clearly decoupled. Thanks to this approach, the consistency protocol implements *exactly the same distributed algorithm* as in its initial, non fault-tolerant version.

5. Implementation and preliminary evaluation

To experiment our approach, we have used the JUXMEM software experimental platform for grid data sharing, described in [3]. We have refined its architecture according to the decoupled approach proposed in this paper and we have implemented the fault-tolerant consistency protocol described in Section 4.3.

The general architecture of JUXMEM mirrors a federation of distributed clusters and is therefore *hierarchical* (Figure 4). It consists of node sets, called `cluster`
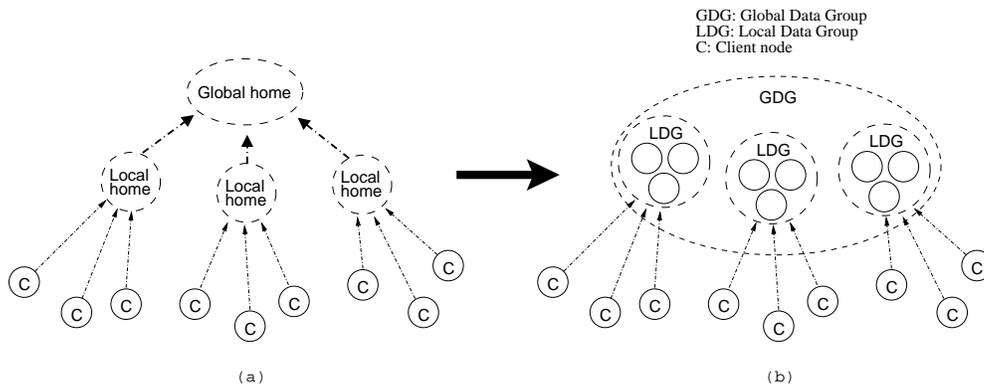
Figure 3. *A hierarchical architecture for the fault-tolerant consistency protocol.*

groups, which correspond to physical clusters. These groups are included in a wider group, the `juxmem` group, which gathers all the nodes running the data-sharing service. Note that these *service groups* consist of different nodes with different states. They do not make up a replicated service and do not rely on the same abstractions (group membership, atomic broadcast) as the groups previously described, that act as home nodes. Any `cluster` group consists of *provider* nodes which supply memory for data storage. The memory available in the group is handled by a *cluster manager*. Any node (including providers and cluster managers) may use the service to allocate, read or write data as *clients*, in a peer-to-peer approach. This architecture has been implemented using the JXTA [24] generic P2P platform.

When allocating memory, the client has to specify on how many clusters the data should be replicated, and on how many nodes in each cluster. This results into the instantiation of the GDG and LDG entities used by the consistency protocol, as explained in Section 4.3. In the example shown on Figure 4, data is replicated across two LDGs created on two different clusters. Each LDG is made up of three physical nodes. The allocation operation returns a global data ID. To read/write a data block, clients only need to specify this ID. The platform transparently locates the corresponding local LDG or instantiates it if necessary. Subsequent accesses to data are directed to this LDG by the consistency protocol.

At the low level of our architecture, the LDG and GDS components have been implemented based on the fault-tolerant, leader-based group communication protocol proposed in [9]. Our implementation supports node crashes and link failures. In each LDG or GDG group, up to $\lfloor n/2 \rfloor$ failures are supported, where n is the group size.

Preliminary evaluation. For our preliminary experiments, we have used the JDF [2] deployment suite to run our tests over a 64-node cluster of 2,4GHz bi-Pentium IV with 1GB RAM, interconnected through a Fast-Ethernet network. We partitioned our physical cluster into 8 `cluster` groups, 8 nodes each. Our software environment is JUXMEM running over JXTA 2.2.1 and Java 1.4.2.

We first analyzed the impact of the replication degree on the cost of data allocation. The allocation procedure consists of 3 steps: 1) the client has to discover enough providers in the JUXMEM network to satisfy the replication degree; 2) the client sends allocation requests to a set of discovered providers, selected in order to satisfy the user-specified constraints (concerning replication degrees, locality, etc.); 3) the selected providers perform the actual allocation and instantiate the consistency protocol layer and the necessary group communication components; this results in creating the corresponding LDGs and GDG.

We have evaluated the impact of the replication degree on the allocation cost by varying the sizes of the GDG and LDG groups (Figure 5). We can note that: 1) the architecture initialization cost is largely overcome by the communication involved by the first two steps described above (discovery and allocation requests); 2) the discovery cost grows linearly with respect to the replication degree; 3) the cost of the actual allocation is quasi-constant despite the number of required replicas, because the client makes all these requests in parallel.

We have also measured the cost of the basic operations of the consistency protocol: data read and data update. These operations involve communications between a client and its local LDG. We measured the cost of these operations while varying the cluster-level replication degree (i.e. the LDG size). This is illustrated on Figure 6. First, we can note that the overhead due to replication is significant for small data sizes (e.g. 16 KB): the read and update operations are three
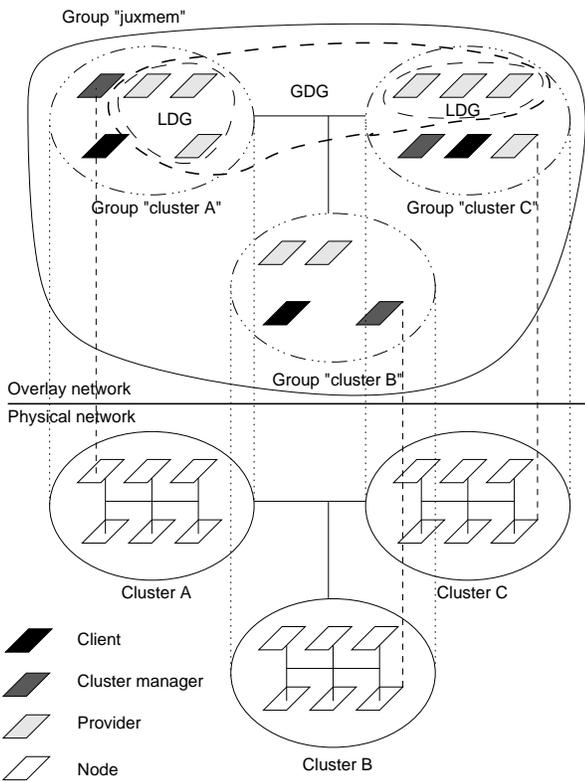
Figure 4. JUXMEM**:** *a hierarchical architecture for a grid data service.*

Figure 5. *Allocation costs depends on replication degree.*
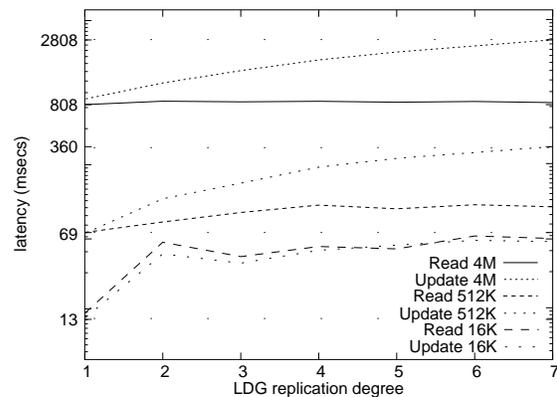
Figure 6. *Cost of the basic primitives: read/update.*

times slower, because our atomic multicast protocol uses a two-phase commit strategy. However, this cost increases very slowly with the replication degree. Second, for large data sizes (e.g. larger than 512 KB), the fault-tolerance overhead is negligible compared to the data transfer delay. The cost of update operations linearly increases with the replication degree. This is due to our leader-based implementation of the group communication protocol, where the leader node sends the data to all the group members across the network. We plan to perform further measurements to evaluate: 1) the service throughput; 2) the impact of failures on the performance of the service operations.

6. Conclusion

In this paper, we have addressed the problem of handling the consistency of replicated data in a grid data-sharing service. In such a context, the availability of storage resources changes dynamically. We have shown the advantages of a software architecture which decouples consistency management from fault-tolerance management. We have illustrated
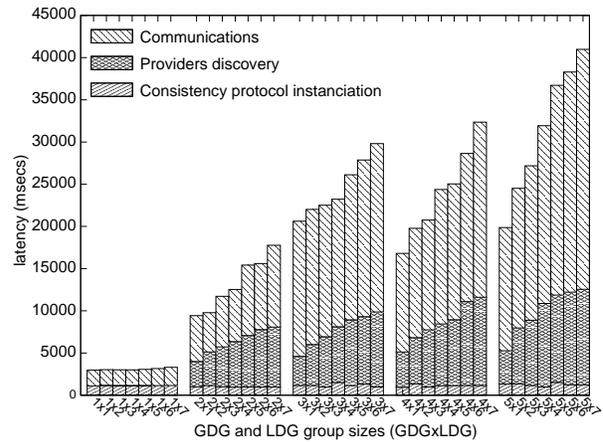
our approach by showing how to design a fault-tolerant consistency protocol which implements the entry consistency model. As a preliminary experimental validation, we have implemented a prototype of the proposed fault-tolerant consistency protocol within JUXMEM, a software experimental platform for grid data sharing.

The main advantage of the proposed approach is that it allows the consistency protocol and the replication strategy to be designed independently, while only a small interaction has to be defined through the *Proactive Group Membership* and the *Dynamic Consistency Protocol Configuration* layers. Thereby, existing consistency protocols can be made fault-tolerant by carefully defining this interaction. Different trade-offs (e.g., efficiency vs. level of fault-tolerance) can be obtained by tuning this interface. Such studies are part of our planned future work.

7

Naturally, if the policy implemented by the Proactive Group Membership layer is well-tuned in order to fit the characteristics of the physical architecture, the availability of the home nodes will be guaranteed most of the time. This is true as long as the assumptions made about the fault types and about the number of concurrent faults are correct. Otherwise, recovery will not be possible, and the user application will be informed about this by the consistency protocol. It is then its responsibility to react, according to its specific constraints (retry, rollback, etc.). Such events should however be extremely rare if the proactive group membership policy is correctly tuned. We are currently working on extensions of our approach, in order to define an extended semantics of the consistency protocol, which should take into account such cases.

References

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28(5):749–771, 2002.

[2] G. Antoniu, L. Bougé, M. Jan, and S. Monnet. Large-scale deployment in P2P experiments using the JXTA distributed framework. In *Proc. 10th International Euro-Par Conference on Parallel Processing (Euro-Par '04)*, volume 3149 of *LNCS*, pages 1038–1047, Pisa, Italy, Aug. 2004. Springer.

[3] G. Antoniu, L. Bougé, and M. Jan. JuxMem: An adaptive supportive platform for data sharing on the grid. In *Proceedings Workshop on Adaptive Grid Middleware (AGRIDM 2003)*, pages 49–59, New Orleans, Louisiana, Sept. 2003. Held in conjunction with PACT 2003. Extended version to appear in Kluwer Journal of Supercomputing.

[4] G. Antoniu, L. Bougé, and S. Lacour. Making a DSM consistency protocol hierarchy-aware: An efficient synchronization scheme. In *3rd IEEE/ACM International Conference on Cluster Computing an the Grid (CCGrid 2003)*, pages 516–523, Tokyo, Japan, May 2003. IEEE.

[5] L. B. Arantes, P. Sens, and B. Folliot. An effective logical cache for a clustered LRC-based DSM system. *Cluster Computing Journal*, 5(1):19–31, Jan. 2002.

[6] A. Bassi, M. Beck, G. Fagg, T. Moore, J. Plank, M. Swany, and R. Wolski. The Internet Backplane Protocol: A study in resource sharing. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, pages 194–201, Berlin, Germany, May 2002. IEEE.

[7] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. The Midway distributed shared memory system. In *Proceedings of the 38th IEEE International Computer Conference (COMPCON Spring '93)*, pages 528–537, Los Alamitos, CA, Feb. 1993.

[8] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN '02)*, pages 354–363, Washington, DC, June 2002.

[9] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, Nov. 2002.

[10] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, Mar. 1996.

[11] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469, Dec. 2001.

[12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[13] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *17th International Symposium Computer Architecture (ISCA 1990)*, pages 15–26, Seattle, WA, June 1990.

[14] R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4):68–74, 1997.

[15] L. Iftode, J. P. Singh, and K. Li. Scope consistency: A bridge between release consistency and entry consistency. In *Proceedings of the 8th ACM Annual Symposium on Parallel Algorithms and Architectures (SPAA'96)*, pages 277–287, Padova, Italy, June 1996.

[16] A.-M. Kermarrec, G. Cabillic, A. Gefflaut, C. Morin, and I. Puaut. A recoverable distributed shared memory integrating coherence and recoverability. In *The 25th International Symposium on Fault-Tolerant Computing Systems (FTCS-25)*, pages 289–298, Pasadena, California, June 1995.

[17] T. Kosar and M. Livny. Stork: Making data placement a first-class citizen in the grid. In *Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pages 342–349, Tokyo, Japan, Mar. 2004.

[18] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, Nov. 1989.

[19] S. Mena, A. Schiper, and P. Wojciechowski. A step towards a new generation of group communication systems. In *Proceedings of International Middleware Conference*, volume 2672 of *LNCS*, pages 414–432, Rio de Janeiro, Brazil, June 2003. Springer.

[20] J. Protić, M. Tomasević, and V. Milutinović. *Distributed Shared Memory: Concepts and Systems*. IEEE, Aug. 1997.

[21] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec. 1990.

[22] F. Sultan, T. Nguyen, and L. Iftode. Scalable fault-tolerant distributed shared memory. In *The IEEE/ACM SuperComputing conference (SC '00)*, pages 54–55, Dallas, Texas, Nov. 2000.

[23] Y. Zhou, L. Iftode, and K. Li. Performance evaluation of two home-based lazy release consistency protocols for shared memory virtual memory systems. In *Proceedings 2nd Symposium on Operating Systems Design and Implementation (OSDI '96)*, pages 75–88, Seattle, WA, Oct. 1996.

[24] The JXTA project. http://www.jxta.org/.