Vigne: Towards a Self-Healing Grid Operating System

Louis Rilling

IRISA/Université de Rennes 1/ENS Cachan, Brittany site - PARIS research group
Louis.Rilling@irisa.fr

Abstract We consider building a Grid Operating System in order to relieve users and programmers from the burden of dealing with the highly distributed and volatile resources of computational grids. To tolerate the volatility of the nodes, the system should be self-healing, that is continuously adapt to additions, removals, and failures of nodes. We present the self-healing architecture of the Vigne Grid Operating System through three of its services: system membership, application management, and volatile data management. The experimental results obtained show that our approach is feasible.

1 Introduction

Grids gather large sets of services over a large set of resources provided by many independent organizations. The nodes of such distributed systems are in essence volatile: organizations may unilaterally decide to add or remove nodes at any time, and the failure rate increases with the number of nodes.

We consider building a Grid Operating System (GOS) in order to relieve users and programmers from the burden of dealing with such highly distributed and volatile resources. To achieve this goal, a GOS should provide users and programmers with simple abstractions of physically highly distributed resources, and transparently handle additions, removals, and failures of nodes.

We consider building *self-healing* systems. The self-healing property is a variant of fault-tolerance in which the system proactively maintains its degree of fault-tolerance. The mechanisms of the system must continuously adapt to additions, removals, and failures of nodes. This is an important property since assuming that human interventions quickly restore failed resources can not scale to large numbers of nodes. Moreover, no service of the system should depend on the stability of any set of nodes during the whole system's lifetime. However, current approaches like Globus [1] still rely on static hierarchies, defined by system administrators, and that prevent the system from being self-healing.

In this paper we present the self-healing architecture of the Vigne GOS, through the design of three of its services. One of the main contributions of this architecture is the application management service which decentralizes application control and provides applications with generic and transparent fault-tolerance policies. We implemented most of these three services and present in this paper experimental results obtained by simulations, which show the feasibility of our approach.

The paper is organized as follows. We precise our model of distributed system in Sect. 2 and give an overview of Vigne in Sect. 3. Then we present three self-healing services of Vigne, namely system membership in Sect. 4, application management in Sect. 5, and volatile data management in Sect. 6. We present an experimental evaluation of the self-healing properties of volatile data management in Sect. 7, and discuss related work in Sect. 8. Finally, Sect. 9 concludes.

2 System Model

The nodes of the system belong to many independent organizations. For this reason we consider a (large scale) distributed system composed of nodes which can fail, recover (or be added), and be gracefully removed (the last two events are called reconfigurations in the paper). Nodes fail in a fail-stop manner. Failures can be detected using (unreliable) failure detectors. We do not consider byzantine failures, as they are relevant to security. We focus our work on the scalability and self-healing aspects, and expect security issues to be tackled in future work.

Users run distributed as well as sequential applications on this system. Many users may run many applications simultaneously, using the system as a computational power provider.

3 Overview of Vigne

We consider building a Grid Operating System (GOS). As any operating system, a GOS virtualizes the physical resources to provide users and programmers with simple abstractions. A set of services is depicted in Fig. 1. In this paper we focus on the self-healing aspect of a GOS.

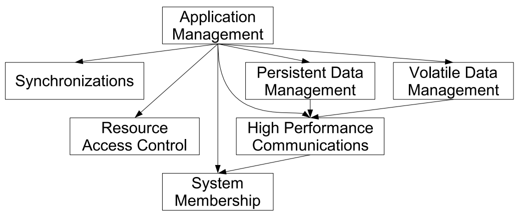

Figure 1. Services of a Grid Operating System

The application management service (AMS) is the top-level service of the system. This service controls applications executions, and is the main service with which users interact. The AMS runs each application under the control of a dedicated self-healing agent called *application manager*. An application manager acts on behalf of the user to run efficiently the application and to ensure that it terminates correctly, despite node removals and failures.

The persistent data management service stores data in logical files that have location-independent names. The lifetime of these files is independent from the lifetime of applications. Conversely, the volatile data management service (VDMS) manages volatile data that is private to applications. The VDMS offers abstractions of shared data to build distributed applications communicating through the shared-memory paradigm. Since data managed by the VDMS is volatile and private to a single application, the VDMS can apply fault-tolerance mechanisms that are less costly in resources and performance than the mechanisms needed for persistent data. The VDMS is based on fault-tolerant *consistency protocols* allowing to replicate shared data to improve performance [2].

The system membership service (SMS) is the basis on which all other services are implemented. The SMS connects the nodes of the system in a scalable, decentralized, and self-healing manner. The SMS of Vigne is based on a *structured overlay network* designed in recent research in the peer-to-peer field [3].

The other services are not discussed in this paper. We briefly describe them. The synchronization service provides applications with synchronization primitives comprising distributed semaphores and barriers. The high performance communication service provides applications as well as higher level services with communication primitives that adapt to parallel communication links and to the various security policies used on the nodes. The resource access control service enforces the resource sharing policy defined by organizations for their nodes.

The main principle driving our approach is to simplify as much as possible the job of users and programmers without restricting the field of applications. In particular the GOS should relieve users and programmers from dealing with failures. The next sections describe the self-healing properties of Vigne's membership, application management, and volatile data management services.

4 System Membership

The system membership service of Vigne must achieve two goals despite continuous reconfigurations and failures: maintain the nodes of the system in connection, and deliver accurate membership information to higher level services. The system membership service is based on a structured overlay network built using the structure and routing algorithms of Pastry [3], and the maintenance algorithms of Bamboo [4]. The basic mechanism implemented by the overlay network is key-based routing, which allows to build self-healing distributed hash tables (DHT). The keys of such DHTs can be used as location-independent names, as the overlay network routes a message to a key without needing that the sender knows which node hosts the key. DHTs are a sound basis to build self-healing higher-level services. This is illustrated in the next sections for the application management service and the volatile data management service.

Structured Overlay Network. Pastry connects the nodes of the system in a logical ring. Nodes have numerical names, called ID and represented in hexadecimal, and are placed in clockwise order on the ring. Pastry maps a key (also represented in hexadecimal) to the node having the numerically closest ID (see Fig. 2, left part). Pastry routes messages to keys following the logical ring and shortcut links that allow to limit the average number of routing hops to $\log_{16} N$ with only $O(\log N)$ links per node, where N is the number of nodes connected to the system. The overlay network is made self-healing using redundant links to the neighbors in the ring, and gossiping protocols to refill the routing tables [4].

Distributed Hash Tables. On top of this structured overlay network we have implemented a distributed hash table (DHT) service that provides generic management of self-healing DHTs. This service allows higher level services to define any number of DHTs. DHTs are generically made self-healing by automatically moving and replicating keys using a per-DHT defined replication degree. The nodes hosting the replicas of a key are the numerically closest neighbors (clockwise and counter clockwise) of the node to which the overlay networks maps the key (see Fig. 2, right part). Automatic replication management of the keys can be customized by higher level services (for instance application management).

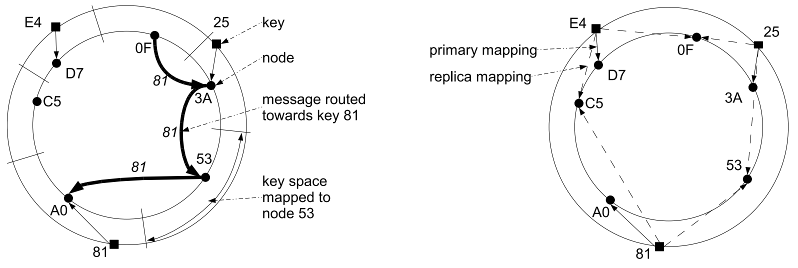

Figure 2. Basic topology of a Pastry-based structured overlay network and mapping from keys to nodes (left part). Replication of keys in a self-healing DHT (right part).

5 Application Management

The application management service is the main interface of the system for users. This service controls the execution of all applications in order to minimize the execution time and to reliably execute each application, that is to ensure that the application correctly terminates despite failures. A discussion on minimizing execution time is out of the scope of this paper. To reliably execute applications, the application management service of Vigne controls the execution of each application through a dedicated self-healing agent called application manager. In this section we present the design of these application managers.

Application managers have three main features: control applications execution in a decentralized manner, transparently handle failures and reconfigurations, and allow to flexibly define fault-tolerance policies for each application.

Decentralized Control of Applications. Decentralization is achieved by placing application managers as keys in the application manager DHT (which is implemented using the system membership service, see Sect. 4). The keys are distributed over the whole system using secure hash functions, like SHA-1, to define the key's numerical name. Decentralization not only allows to avoid contention on certain nodes because of the load generated by application control, but also allows to limit the cost of a node removal or failure to the reconfiguration of few application managers. The cost of a node removal or failure is limited thanks to the locality properties of DHTs: only the application managers having a replica located on the removed (or failed) node are affected.

Transparent Handling of Failures and Reconfigurations. To relieve users from dealing with failures and reconfigurations, application managers transparently handle failures and reconfigurations from the point of view of users. Indeed, from the point of view of a user, a running application is represented by its application manager. To achieve transparency, an application manager is reachable through a location-independent name (its key in the application manager DHT), is self-healing by replicating itself using a group communication system and the DHT service, and, to handle all removals or failures of nodes that host components of the application, applies a fault-tolerance policy.

A group communication system is used to actively replicate application managers. The messages sent to an application manager are atomically multicast to the group of replicas of the application manager. Provided that an application manager can be defined as a deterministic input / output state machine, this ensures that all replicas output the same sequence of messages. To ensure this determinism, the failure detection mechanisms used by an application manager to monitor an application interact with the application manager through the group communication system (see Fig. 3).

The nodes hosting the replicas are automatically chosen using the DHT service, which makes application managers self-replicating and self-healing. However, to keep the replicas synchronized, creating replicas must be done under the control of the group communication system. To this end, the DHT service only informs application managers of the nodes on which they should replicate. For this reason, we chose to build a group communication system based on the architecture defined in [5], which offers the required flexibility.

Application Fault-Tolerance Policies. To relieve users from dealing with failures, an application manager applies a fault-tolerance policy defined for the application. Thanks to this feature, application managers are a powerful and flexible mechanism to provide applications with generic fault-tolerance with minimal efforts from users and programmers. The fault-tolerance policy can be a generic predefined policy, for instance based on checkpointing and restart, or a policy specifically designed for the application, for instance make the application rebuild lost data using data from a previous computing step [6]. In each case, the application manager enforces a fault-tolerance policy by reacting to suspicions of nodes hosting components of the application (see Fig. 3, right part).

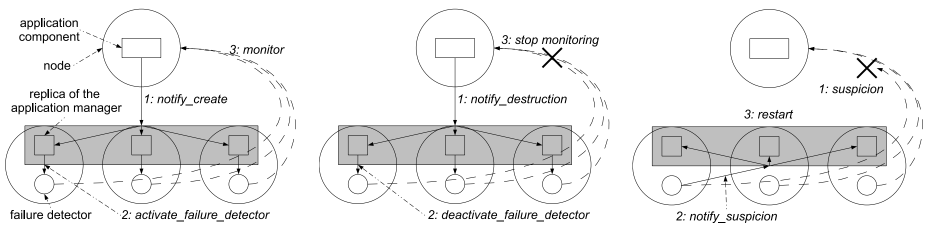

Figure 3. Communications between application components, failure detectors, and the replicas of an application manager, when creating (left) or destroying (center) components, or suspecting nodes (right).

6 Volatile Data Management

The volatile data management service (VDMS) helps programmers to build distributed applications that use the shared memory paradigm to communicate, offering to these programmers abstractions of shared objects to which processes can access using location-independent names. To achieve this the VDMS of Vigne includes consistency protocols to provide the programmer with consistency models on the values of the copies of a shared object.

We have studied two protocols ensuring atomic consistency. These protocols are based on the write-invalidate scheme, in order to obtain performance (see [2] for a discussion). Before granting write access rights to a copy, the protocols ensure that all other copies are invalid. In our protocols, at each time one (and only one) copy is distinguished as the master copy. Other copies become valid by retrieving the value of the master copy.

These protocols are based on protocols designed by K. Li [7]. K. Li's protocols were designed for a static system having reliable FIFO communication channels. We improved K. Li's protocols to handle multiple and simultaneous reconfigurations and failures, and to tolerate unreliable communication channels. To handle reconfigurations and failures, we leverage the application management and system membership services. We tolerate unreliable communication channels in order to improve the scalability of memory consumption. We have proved in [8] that in our approach the amount of memory consumed per node does not depend on the number of nodes in the system, whereas this cost is linear in the number of nodes if communication channels are made reliable in the communication layer.

Both protocols eventually rely on the application manager to ensure fault-tolerance. However, we also handle reconfigurations and failures in the consistency protocols in order to limit the cost of the fault-tolerance mechanisms used by the application manager. For instance, in both protocols a copy may become useless when the node hosting it does not run processes of the application anymore. The removal or the failure of such nodes should not force the application manager to react (for instance by restarting the application). However, K. Li's protocols and their variants in the literature have to ensure that such copies are invalid before granting write access rights to another copy, and for this reason they block if a copy, even useless, can not send acknowledgments. Using these protocols without adapting them to dynamic reconfigurations and failures would force the application manager to perform costly fault-tolerance operations (for instance, restart the application) when no process of the application is lost.

In the first protocol, called STAT, object managers handle access requests from copies and redirect them to the master copy. In order to avoid contention, these object managers are distributed over a DHT. Compared to similar protocols, this allows the protocol to handle reconfigurations simply since object managers have location-independent names and remain reachable thanks to the self-healing management of the structured overlay network and of the DHT.

In the second protocol, called DYN, the copies organize themselves in chains of references towards the master copy. Compared to the STAT protocol, this avoids paying the latency of routing a message through the overlay network for each access request from a copy. However, reconfigurations break these chains. Therefore we added backup object managers, which are located in a DHT, and to which the master copy periodically publishes its location. A copy sends an access request to the backup object manager only when it suspects that its chain towards the master copy is broken.

7 Experimental Evaluation

We have implemented the membership, application management, and volatile data management services (VDMS) of Vigne, except advanced fault-tolerance policies and application manager replication which will be implemented and evaluated later. Based on this implementation, we present an evaluation of the self-healing property of the VDMS. An evaluation of the system membership service figures in [9].

We show the failure resilience of the consistency protocols of the volatile data management service, using a discrete event simulator coupled to the running Vigne prototype. To do this, we simulated the execution of a single writer multiple readers application on a set of 2000 volatile nodes. Node additions and failures were injected using traces collected in the Gnutella peer-to-peer file sharing application on the Internet [10], which represents an extreme case of volatility compared to an industrial grid environment (see the right part of Fig. 4 for the cumulative number of failures injected during the experiment).

Each component of the application runs 1000 loops composed of two phases. In the first phase, the writer writes a value to a shared object, and in the second phase all other components (the readers) read the value of the shared object. With this access pattern, each access from a component to its copy generates an access request. We ran experiments for 20 to 400 readers.

Upon a failure of a node hosting a component of the application, the application was immediately restarted. Coordinated checkpoints were taken before each iteration. In the simulator, coordinated checkpoints and restarts are done in null time, which allows us to observe the impact of node volatility on the other fault-tolerance mechanisms used in the protocols.

Figure 4 (left part) shows the progression of the application for both protocols and sample numbers of readers. The performance of the DYN protocol is much better than the performance of the STAT protocol. The progression of the application with the DYN protocol is almost linear even with a high number of readers (see Fig. 4, center part). In contrast, the performance of the STAT protocol degrades quickly when the number of readers increases. The nodes routing tables of the overlay network are continuously damaged and repairing which critically increases the latencies of the routed messages used in each access request. We also observe this effect with the DYN protocol each time the application is restarted, since messages are routed to reset the protocol (see Fig. 4).

These results suggest that the DYN protocol tolerates well frequent failures. Moreover, these results suggest that DHTs should be used with care in execution paths which are critical for application performance. These results also suggest that the STAT protocol is useless, but we showed in [8] that the STAT protocol exhibits better performance than the DYN protocol for applications having access patterns in which many write accesses are concurrent with other accesses.

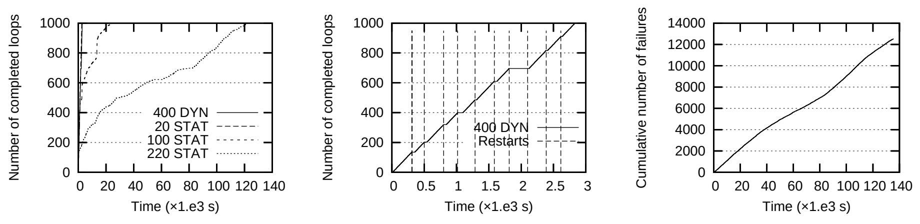

Figure 4. Performance of STAT and DYN in a highly dynamic configuration

8 Related Work

Grid Operating Systems. Many grid infrastructures, including Globus [1], Legion [11], GridOS [12], and 9grid [13], provide operating system-like services. Globus, Legion, and GridOS are designed as middleware to ease the portability on heterogeneous operating systems, whereas 9grid is an integrated grid operating system which design is simple partly because it enforces that all applications run on top of the services. Between these two extreme approaches, Vigne adapts existing operating systems to, on the one hand, keep legacy interfaces and run legacy applications, and on the other hand, enforce resource sharing policies and provide generic application management.

In all these infrastructures, fault-tolerance is addressed for the services, but only to a limited extent because the systems rely on static hierarchies defined by system administrators. In contrast, Vigne's services are designed to be fully self-healing in order to continuously tolerate failures of any node in the system, without needing any action from system administrators.

In current grid infrastructures, fault-tolerance is not addressed for applications, the main assumption being that this task should be left to application-specific services. In contrast, as a true grid operating system should do, Vigne provides generic application fault-tolerance services that should meet the needs of most of the use cases, and helps applications to define custom mechanisms for the other use cases.

Membership. We based our system membership service on a Pastry-like structured overlay network. Other works, including JXTA [14] and NaradaBrokering [15], aim at providing infrastructures to build high level peer-to-peer services. JXTA is built on an hybrid structured peer-to-peer network and provides a loosely consistent DHT, which model differs from the DHTs we are using. JXTA's DHT only stores advertisements for resources bound to peers. In particular, this DHT does not manage the location and the replication of the objects for which it stores advertisements.

NaradaBrokering provides a communication infrastructure including scalable event-delivery and publish-subscribe to build high level services. NaradaBrokering's features are complementary to the features of our system membership service. However, the brokering infrastructure's design assumes that a set of nodes remains relatively stable, and the volatility of the nodes is mostly considered for the clients of the brokering services.

Application Management. Few projects include generic application management services to execute applications reliably. Chameleon [16] and XtremWeb [17] provide fault-tolerance mechanisms for a variety of programming models. In particular, Chameleon provides users with generic mechanisms for various fault-tolerance policies. In both systems, application management relies on a centralized entity (the main fault-tolerance manager in Chameleon, or the coordinator in XtremWeb), which is itself made reliable by replication on a static set of nodes. As a major contribution, our application managers decentralize application management, which is better to avoid contention and to resist to massive failures. Moreover application managers are replicated on dynamic sets of nodes, which allows them to adapt to any reconfiguration in the system.

Shared Data Management. Several projects, like JuxMem [18] or Pastis [19], provide mutable shared data management in large scale distributed systems composed of volatile nodes. Compared to our volatile data management service, these systems consider persistent data, which prevents them from providing programmers with an integrated fault-tolerance solution taking programs and data into account. In JuxMem, applications have to adapt to the fault-tolerance mechanisms used for the data, which makes fault-tolerance not fully transparent to the programmers. In Pastis, the replication degree of the data is maintained to keep the data available, but no mechanism allows application fault-tolerance mechanisms to synchronize with consistent versions of the data. Fault-tolerant volatile data management has only been studied in the context Distributed Shared Memory systems [20], which only consider clusters of workstations composed of at most a few hundreds of nodes that rarely undergo reconfigurations or failures.

9 Conclusion

In this paper we have presented the design of three self-healing services of the Vigne Grid Operating System. The self-healing property is important to ensure the availability of the system and to relieve users and programmers from dealing with reconfigurations and failures. This paper brings two contributions. As the the volatile data management service illustrates, the system membership service and the application management service that we presented constitute a sound basis for a grid operating system. In particular, with application managers our application management service decentralizes application control and provides applications with generic and transparent fault-tolerance. Thanks to application managers, the consistency protocols of our volatile data management service are the first ones based on the write-invalidate scheme for performance and tolerating multiple simultaneous failures. The experimental results on highly dynamic configurations suggest that the chosen self-healing architecture is feasible.

Future work includes enhancing the volatile data management service to enable programmers to choose between various consistency models, still without having to handle failures. We will also implement the group communication system described in [5]. This will complete the self-healing architecture of Vigne and will allow us to evaluate the application management service. In particular, we will be able to evaluate various fault-tolerance policies.

10 Acknowledgments

The author would like to thank Emmanuel Jeanvoine for his participation in designing Vigne, and Christine Morin for her valuable advices and comments.

References

1. Foster, I., Kesselman, C., eds.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, CA, USA (1999)
2. Rilling, L., Morin, C.: A practical transparent data sharing service for the grid. In: Proc. Fifth International Workshop on Distributed Shared Memory (DSM 2005), Cardiff, UK (2005) Held in conjunction with CCGrid 2005.
3. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proceedings of Middleware 2001. Volume 2218 of Lecture Notes in Computer Science., Springer (2001) 329–350
4. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In: Proceedings of the USENIX Annual Technical Conference. (2004) 127–140
5. Mena, S., Schiper, A., Wojciechowski, P.: A step towards a new generation of group communication systems. In: Proceedings of Middleware 2003. Volume 2672 of Lecture Notes in Computer Science., Springer (2003) 414–432

6. Garbey, M., Ltaief, H.: Fault tolerant domain decomposition for parabolic problems. In: 16th International Conference on Domain Decomposition Methods. Lecture Notes in Computational Science and Engineering, Springer (2005) To appear.
7. Li, K., Hudak, P.: Memory coherence in shared virtual memory systems. ACM Transactions on Computer Systems **7**(4) (1989) 321–359
8. Rilling, L.: Système d'exploitation à image unique pour une grille de composition dynamique : conception et mise en œuvre de services fiables pour exécuter les applications distribuées partageant des données. PhD thesis, Université de Rennes 1, IRISA, Rennes, France (2005) In French.
9. Jeanvoine, E., Rilling, L., Morin, C., Leprince, D.: Using overlay networks to build operating system services for large scale grids. In: Proceedings of the fifth International Symposium on Parallel and Distributed Computing (ISPDC 2006), Timisoara, Romania (2006) To appear.
10. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking (MMCN) 2002, San Jose, CA, USA (2002)
11. Grimshaw, A.S., Wulf, W.A., Team, C.T.L.: The legion vision of a worldwide virtual computer. Communications of the ACM **40**(1) (1997) 39–45
12. Krauter, K., Maheswaran, M.: Architecture for a grid operating system. In: Proceedings of the First IEEE/ACM International Workshop on Grid Computing. Volume 1971 of Lecture Notes In Computer Science., Bangalore, India, Springer-Verlag (2000) 65–76
13. Mirtchovski, A., Simmonds, R., Minnich, R.: Plan 9 – an integrated approach to grid computing. In: 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop on High-Performance Grid Computing, Santa Fe, New Mexico, USA, IEEE, CS Press (2004) 273a
14. Traversat, B., Abdelaziz, M., Pouyoul, E.: Project JXTA: A Loosely-Consistent DHT Rendezvous Walker. http://www.jxta.org/docs/jxta-dht.pdf (2003)
15. Pallickara, S., Fox, G.: NaradaBrokering: A middleware framework and architecture for enabling durable peer-to-peer grids. In: Proceedings of Middleware 2003. Volume 2672 of Lecture Notes in Computer Science., Springer (2003) 41–61
16. Kalbarczyk, Z.T., Iyer, R.K., Bagchi, S., Whisnant, K.: Chameleon: A software infrastructure for adaptive fault tolerance. IEEE Transactions on Parallel and Distributed Systems **10**(6) (1999) 560–579
17. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V., Lodygensky, O.: Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. Future Generation Computer Systems **21**(3) (2005) 417–437
18. Antoniu, G., Deverge, J.F., Monnet, S.: How to bring together fault tolerance and data consistency to enable grid data sharing. Concurrency and Computation: Practice and Experience (2006) To appear.
19. Busca, J.M., Picconi, F., Sens, P.: Pastis: A highly-scalable multi-user peer-to-peer file system. In: Proceedings of Euro-Par 2005. Volume 3648 of Lecture Notes in Computer Science., Springer (2005) 1173–1182
20. Shafi, H., Speight, E., Bennett, J.K.: Raptor: Integrating checkpoints and thread migration for cluster management. In: Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS'03), IEEE (2003) 141–152