Using Overlay Networks to Build Operating System Services for Large Scale Grids

Emmanuel Jeanvoine[†‡], Louis Rilling[†*], Christine Morin[†#], Daniel Leprince[‡]
[†]IRISA/Paris Research group, [‡]EDF, [*]Université de Rennes 1, [#]INRIA
{*Emmanuel.Jeanvoine,Louis.Rilling,Christine.Morin*}*@irisa.fr, Daniel.Leprince@edfgdf.fr*

Abstract

Using grid resources to execute scientific applications requiring a large amount of computing power is attractive but not easy from the user point of view. Vigne is a grid operating system designed to provide users with a simplified view of a grid. Vigne deals with the huge number of nodes in a large-scale grid and with the nodes' dynamic behavior by using peer-to-peer overlays as a keystone. In this paper, we show why it is highly desirable to use structured and unstructured peer-to-peer overlays for building the high-level services of Vigne grid operating system. To show the interest of our approach, we detail the features of two Vigne services built on top of peer-to-peer overlays. We also present experimental results obtained on the Grid'5000 testbed showing the scalability of the Vigne infrastructure based on overlays and its practical interest for the implementation of Vigne distributed services.

1. Introduction

Numerical simulation has an important place in several scientific fields where real experimentation is expensive or impossible. Many scientific applications require a large amount of resources that a single workstation cannot provide. Since the last decade, clusters and grids have become attractive for the execution of such applications. A challenge for the institutions needing a large computing power is to harness a huge amount of volatile resources distributed over many sites in a grid. As a first issue, the large number of resources and their physical distribution make it difficult to locate good resources. We propose to offer a *Single System Image* (SSI) view of the resources to overcome this issue. As a second issue, node additions, removals, and failures can occur frequently, which is hard for users and administrators to manage. Thus, we propose to design a self-organizing and a *self-healing* system to relieve users and administrators from large scale and dynamic behavior issues.

To simplify the use of a grid, we propose a grid operating system, called Vigne, designed with SSI, self-organizing and self-healing properties. To obtain these properties, we have based several services of Vigne on an infrastructure composed of decentralized self-healing overlays.

In this paper, we present the design principles of Vigne Grid operating system and demonstrate why it is highly desirable to use jointly structured and unstructured overlays in the Vigne infrastructure. The contribution of this paper is the following. We show how two kinds of overlays can be used to build two scalable, self-healing grid operating system services. The experimental results obtained executing Vigne on the Grid'5000 French national testbed show the scalability of the Vigne infrastructure and its practical interest. In particular, this infrastructure allows us to build a service to execute applications reliably on behalf of users, which originality is to be decentralized and self-healing.

The rest of this paper is as follows. In Section 2, we give an overview of the Vigne operating system and present its design principles. We explain the requirements of the Vigne services in terms of distributed object naming and location mechanisms in Section 3. In Section 4 we present the overlays used in the Vigne system and we detail two services based on these overlays. In Section 5, we present an experimental evaluation of the scalability of Vigne overlay infrastructure and provide preliminary performance results for the Vigne resource discovery service. We present related works in Section 6 and we finally conclude in Section 7.

2. Overview of Vigne

We are building the Vigne Grid Operating System (GOS) in order to relieve users and programmers from the burden of dealing with highly distributed resources over a set of volatile nodes. In this section we present the design principles we adopted to achieve this goal, and we give an overview of Vigne's services.

Proceedings of The Fifth International Symposium
on Parallel and Distributed Computing (ISPDC'06)
0-7695-2638-1/06 $20.00 © 2006 IEEE

Design Principles To design the services of Vigne, we followed three main principles. First, to provide users and programmers with *simple* abstractions of the distributed physical resources, the services of Vigne provide *Single System Image* (SSI). Users and programmers have the illusion to work on a single computer that gathers the resources of all the nodes of the grid. This property is important because the distribution of the resources over a large number of nodes makes these resources very difficult to observe, select, and use for humans. This difficulty is all the more increased that these resources are shared between numerous users who run applications which characteristics vary greatly, and that the nodes hosting these resources are volatile.

Second, to ensure that the GOS is available and that applications run correctly despite the volatility of nodes, the services must be *self-healing*. Self-healing services tolerate several near-coincident reconfigurations and failures, and reconfigure themselves to tolerate reconfigurations and failures again. In particular, all services must be decentralized. This property is important to relieve administrators from complex system management when nodes are added or removed from the system or even fail, and to relieve users and programmers from dealing with these events when running and programming applications.

Third, to make the system scale to large numbers of nodes, the GOS services should content with *local knowledge* of the system state. This especially holds for the membership service, which is responsible for connecting the Vigne nodes. Indeed, maintaining a centralized global table of a large number of connected nodes despite the volatility of these nodes consumes too much bandwidth to run distributed applications efficiently.

The self-healing and local knowledge principles drive our choice of using peer-to-peer overlay networks as basic building blocks to design Vigne's services. In Section 3, we present how our three design principles induce requirements for basic object location mechanisms. These needs drive our choice of using jointly a structured and an unstructured overlay network.

Overview of Vigne's Services As any operating system, a GOS virtualizes the physical resources to provide users and programmers with simple abstractions. To this end, Vigne is composed of high level distributed services comprising application scheduling, persistent data management, volatile data management [13], and of low level services comprising resources access control and membership. In this paper we present the application scheduling service of Vigne which abstracts computing resources. The two main mechanisms of the application scheduling service are application managers and a resource allocator. These mechanisms are built on a peer-to-peer-like infrastructure.

To abstract computing resources, the scheduling service runs each application under the control of a dedicated reliable 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 nodes leaving the system or undergoing failures. To run the application efficiently, the application manager automatically selects the nodes on which the components of the application run, and moves components to nodes owning better resources when such nodes are discovered. Node leaves are handled by moving components running on these nodes onto other nodes. Node failures are handled by a configurable application fault-tolerance policy, which for example consists in restarting the application from a checkpoint. The benefit of overlay networks for application managers is presented in Section 4.3.

To select the nodes on which an application should run, the application manager uses a resource allocator based on scalable and *distributed resource discovery*. Given a description of needed resources, for example a number of CPUs of a given binary architecture and their minimum speed, the resource allocator discovers a set of suitable resources, and selects the most appropriate resources found. The resource discovery service of Vigne is presented in Section 4.2.

In Section 3 we draw requirements for these services in terms of basic object location mechanisms. We show that these requirements are met by self-healing and decentralized overlay networks.

3. Requirements in Basic Object Location

Since the services of a GOS handle objects that are distributed over the whole system, these services have to rely on distributed naming and location mechanisms for the objects they handle. In this section, we draw the requirements for these naming and location schemes from the design principles we have adopted to build Vigne. We show that these requirements are satisfied by overlay networks designed in recent peer-to-peer research.

3.1. Naming and Exact Location

A first class of requirements concerns location-independent naming of objects. We show that these requirements can be fulfilled using structured overlay networks.

Requirements Building SSI services on top of a wide area distributed system needs location-independent global naming schemes. For instance, users should be able to get the status of their running applications without knowing which nodes run their applications. Similarly, users should be able to access files without needing to know which nodes

Proceedings of The Fifth International Symposium
on Parallel and Distributed Computing (ISPDC'06)
0-7695-2638-1/06 $20.00 © 2006 IEEE

host these files. Location-independent names also simplify the design of system services: location information about an object is isolated in a specific part of the service providing the object, and need not be updated elsewhere in the system services. Location-independent names are all the more desirable that the nodes of the system are volatile.

Two reasons drive the need for location-independent names: objects may need to change their location, and objects may need to be replicated over several nodes. For persistent objects, like files, it is not reasonable to notify all nodes of the system of location changes because broadcast protocols do not scale well to high numbers of nodes.

Location changes are necessary for the high availability of objects. For instance, since nodes may leave the system at any time and should be able to do this without needing the approval of other nodes, long-running computations located on a leaving node should migrate to nodes remaining in the system in order to have a chance to complete.

Some objects need to be replicated on several nodes in order to remain available despite failures. For instance, users should be able to monitor and control their applications at any time. Therefore the system objects implementing these features should be highly available. Since network and node failures may occur at any time, replicating objects is the only way to keep them available. Depending on the replication scheme and on the object semantics, communications between an entity and a replicated object may involve all replicas of the object, or only a primary replica, or only the replica which is nearest to the entity. As a result, assigning a location figuring in the name of a replicated object may be a non-sense and may add complexity to fault-tolerance mechanisms when the location must change.

Finally, to enforce our design principles the naming schemes used by the GOS services should be self-healing and content with local knowledge of the system state.

Relevance of Structured Overlay Networks The naming and exact location requirements can be fulfilled using structured overlay networks. Indeed, these overlays are self-healing and decentralized, and implement key-based routing (KBR). Using its structure, the overlay dynamically maps keys to nodes and routes a message to a key without that the sender needs to know the actual node to which the key is mapped. For this reason, the keys of a structured overlay network are good candidates to implement location-independent names.

To implement KBR, structured overlays connect the nodes of the system according to their names and the structure adopted. In Pastry [14], nodes have numerical names and are connected in clockwise order to build a logical ring. A key has a numerical name and is mapped to the node having the numerically closest name.

KBR is made efficient using shortcuts links, which typically allows an overlay to route a message in $\log(n)$ hops while keeping only $O(\log(n))$ links per node, where n is the number of nodes connected in the system.

Structured overlay networks can also help to build self-healing (highly available) objects using self-replication. Structured overlay networks are typically used to build fault-tolerant distributed hash tables (DHT). These DHTs store values on the nodes their keys are mapped to, and replicate the values on the neighborhood of these nodes in the overlay. For instance, DHTs based on Pastry self-replicate a value on the nodes preceding and following the node responsible for the key in the logical ring. This scheme can be generalized to make objects accessed using KBR self-replicating, and hence self-healing.

3.2. Attribute-Based Distributed Search

A second class of requirements concerns attribute-based search features. We show that these requirements can be fulfilled using unstructured overlay networks.

Requirements Since we wish to provide SSI features for resource management, the GOS needs mechanisms to discover resources. Indeed, to run an application, the resource discovery service must find a suitable set of nodes in the grid. In contrast with the location and naming requirements shown in Section 3.1, the resource discovery mechanism must handle complex searches. Indeed, the system must find entities in the system without knowing their location or their name, but only using a description based on several criteria. It is important to note that a criterion is specified by a scalar value or by a range of values. As far as resource requirements for an application may be describe with such complex criteria, resource search cannot be performed with an exact location mechanism. With an exact location mechanism, the content of the queries must be hashed to produce a unique name. As far as attributes of a query may not have scalar values, it is very difficult to provide an hash function that conserve good load-balancing properties.

To deal with large scale, a centralized design must be avoided. Indeed, it could induce bottlenecks or single points of failure that would compromise the entire system. For the resource allocation service, a centralized database that contains all informations about existing resources in the grid would simplify the design of the system but we have to radically exclude it.

As a result, the GOS needs a distributed and fault tolerant search mechanism to locate resources in large scale environment just by knowing a set of constraints that must be fulfilled.

Relevance of Unstructured Overlay Networks A distributed attribute-based search mechanism can be built us-

ing unstructured overlay networks like Gnutella [5]. As soon as a node joins the system, it establishes several connections with other nodes, called neighbors. In this type of overlay, the nodes do not take care of their neighbor names. However, nodes connections may take into account load balancing criteria or other policies.

To search an entity in the system, a node sends a query to its neighbors. If a neighbor hosts the searched entity, it replies to the requester. Otherwise it forwards the query to its own neighbors, and so on until a given depth. This depth is similar to the time to live (TTL) of packets in IP networks. This type of search is called flooding search. As far as the cost of flooding the network might be prohibitive if the depth is high, several optimizations like [16, 1] have been studied by the peer-to-peer research community.

4. Using Overlay Networks in Vigne

We present two services implemented in Vigne and based on the use of a decentralized peer-to-peer overlays: resource discovery and application management. We assume in the remainder that each node of the structured or unstructured overlay represents a node in the grid.

4.1. Vigne's Overlay Networks

In Section 3, we showed that Vigne's services require mechanisms for naming, exact location and attribute-based search. These requirements can be fulfilled using overlay networks. As a single kind of overlay cannot address all requirements, we choose to use two overlays as a basis for designing Vigne's services.

We showed that naming and exact location can be achieved using a structured overlay. For this reason, we implemented a structured overlay based on the Pastry routing algorithms [14], using the maintenance algorithms of Bamboo [12], with 256 bits naming spaces. Such naming spaces are large enough to identify nodes in the Grid or other objects like application managers and shared data objects. Furthermore, Pastry perfectly fits the requirements of reliable location of an entity in the system from any node and with a small number of hops.

We also showed that attribute-based search through a large scale system can be achieved using unstructured overlays. We implemented an unstructured overlay whose membership protocol is based on Scamp [3]. On such an overlay, multi-attributes and range queries can be performed because all attributes are forwarded and can be analyzed on the flooded nodes.

Both structured and unstructured overlays have good properties regarding the local knowledge constraint. In Vigne's structured overlay, each node has 24 neighbors in the ring (12 preceding, and 12 following), and $O(\log(n))$ shortcut links (where n is the number of nodes in the system). The shortcut links allow the overlay to route messages to any node in $\log(n)$ hops on average. Furthermore, the number of neighbors known in the ring is large enough to prevent the system from partioning in most cases of simultaneous failures. In the unstructured overlay, each node has a neighborhood of $\log(n)$ nodes. Thus searches can be performed with a good cover rate and without using a large TTL (typically from 5 to 7).

Scalability is intrinsic to decentralized peer-to-peer overlays. Regarding unstructured overlays, basic flooding might be a performance issue but a Random Walk mechanism [16] would sidestep it. By design, both overlays have good self-healing properties.

In Vigne, we reduce the cost of using jointly structured and unstructured overlays by using a lightweight unstructured overlay that uses several properties of the structured overlay. Assuming that, the management of the unstructured overlay in case of churn is really simple and its cost is negligible. Thus, Vigne benefits of both type of peer-to-peer overlays at a low network bandwidth cost.

In Vigne, the structured and unstructured peer-to-peer overlays are complementary in their use and are keystones to build high level services.

4.2. Resource Discovery Service

One of the aims of Vigne is to execute applications without users needing to worry about the real execution location. To this end, the GOS must be able to find free resources in the grid to run applications.

Resource discovery consists in finding a set of resources corresponding to a specification provided by the user in order to run her application in good conditions. The specification includes several requirements of the application such as processor architecture, number of processors, local operating system, cluster scheduler, libraries, memory capacity, free disk space, etc.

These requirements like memory or free disk space are not scalar criteria. Furthermore an application may use a library whose version number is included in a specific range. Searches can be performed with a great expressiveness with an unstructured overlay. As a consequence, we decided to use an unstructured peer-to-peer overlay to perform resource discovery in Vigne.

We implemented several protocols for resource discovery. The first one is a basic flooding protocol. The second one is a random walk protocol, less expensive than flooding in terms of bandwidth consumption. The third one improves the random walk protocol using a learning method to optimize the quality of results and bandwidth consumption.

Proceedings of The Fifth International Symposium
on Parallel and Distributed Computing (ISPDC'06)
0-7695-2638-1/06 $20.00 © 2006 IEEE

4.3. Application Managers

In the Vigne operating system, the application management service is a crucial point in relation to applications life. The application management service manages the lifecycle of applications, and ensures that applications complete correctly despite node departures or failures. For instance, when the service detects the failure of a node where an application was launched, it can re-execute the application on another node. Since users should be able to interact with this service at any time and despite node departures and failures, this service must be highly available, and in particular self-healing.

The major contribution of our application management service is to decentralize application management and make it self-healing. To achieve this, we extensively use the structured overlay network. To build this service, overlay networks help to decentralize application management while still allowing users to access the service with the only knowledge of the identifiers of their applications, and help to make the service self-healing.

To distribute application management, the application management service introduces application managers, which are agents dedicated each to the management of one application. Application managers are distributed over the whole system using the key-based routing (KBR) scheme of the structured overlay network. Each application manager is assigned a key which is the SHA-1 hash of the (application binary MD5 sum, timestamp, name of host) triplet computed when launching the application. This triplet corresponds to the application identifier (AID). An application manager runs on the nodes its key is mapped to.

Thanks to the KBR properties, an application manager can be reached by any entity of the system with the only knowledge of the corresponding AID. This property contributes to provide users and applications with single system image. For instance, if a user wants to know the progression status of the application execution, she routes a request to the SHA-1 hash of the AID. A resource informs the application manager that the execution has completed using the same mechanism.

Application managers are made self-healing using self-replication on the overlay network. More precisely, an application manager is made highly-available using active replication, and the locations of the replicas are automatically chosen in the overlay network using the same scheme as traditionnally used in DHTs (see Section 3.1). Thus a constant degree of replication is automatically maintained in the logical neighborhood of the KBR target node for the application manager's key, which makes the application manager highly-available despite continuous node additions, departures, and failures.

5. Experimental Results

In order to validate the efficiency of Vigne, we conducted two kinds of experiments, both on the Grid'5000 French testbed [6]. Grid'5000 is the French national testbed for grid software where nodes are currently spread over eight sites.

5.1. Scalability of the Overlay Infrastructure

First we show that using decentralized self-healing overlay networks is reasonable and scalable. To do this, we measure the network bandwidth consumed for the maintenance of the overlays in systems composed of highly volatile nodes. This volatility represents an extreme case compared to the expected relative stability of the nodes composing an industrial grid. As a result, the bandwidth consumption observed in the experiment should represent a worst case.

We have used 100 computers belonging to Rennes' Grid'5000 site. Computers are bi-processor either AMD Opteron at 2.2 GHz or Intel Xeon at 2.4 GHz.

We emulated successively 500, 1000, and 2000 logical nodes. The logical nodes were mapped to the physical computers in a round robin fashion. To measure the network bandwidth consumption, we measured the input traffic of each logical node.

Nodes arrivals and departures in the system have been simulated with two exponential laws. For each node, the mean time between two arrivals was 35 min (Poisson process), and the mean life time was 30 min.

Figure 1 and **??** respectively show the total bandwidth consumption and the number of nodes connected during the experiments. The bandwidth consumption observed confirms that emulating several nodes per computers does not hide congestion effects. If we take into account the total bandwidth divided by the number of nodes, this ratio is constant whatever the number of nodes (Figure **??**). Furthermore, the bandwidth consumption per node is reasonable with an average value of 3.3 KB/s.

Even if the dynamic behavior is hostile because peer-to-peer overlays have to re-configure them often, the system can scale regardless.

5.2. High Level Service Evaluation

We now evaluate the resource discovery service which is one of the high level services of Vigne. We show that this service, built on top of peer-to-peer overlays, is efficient.

We have used 472 computers belonging to four Grid'5000 sites (Bordeaux, Orsay, Rennes, and Sophia). Computers are bi-processor AMD Opteron at 2.0 GHz, AMD Opteron at 2.2 GHz and Intel Xeon at 2.4 GHz.

Proceedings of The Fifth International Symposium
on Parallel and Distributed Computing (ISPDC'06)
0-7695-2638-1/06 $20.00 © 2006 IEEE

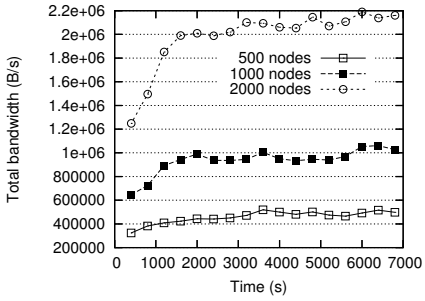

Figure 1. Global bandwitdh consumption

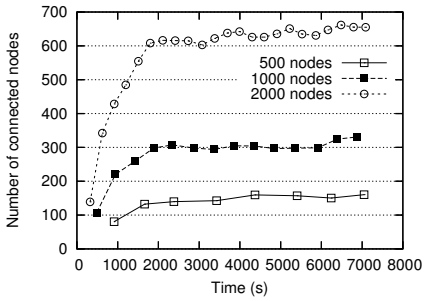

Figure 2. Number of nodes connected

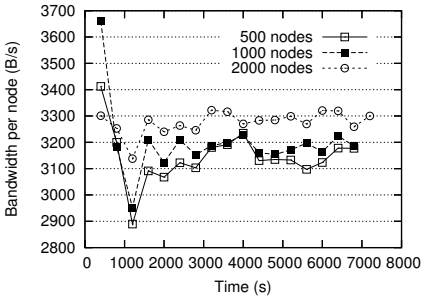

Figure 3. Bandwidth consumption per node

To produce an heavy load on a processor, we have used an iterative calculus code. Its execution time depends on the execution site.

Grid'5000 site	# of CPU	Execution time
Bordeaux	82	1740 s
Orsay	344	1959 s
Rennes (Paraci cluster)	98	1594 s
Rennes (Parasol cluster)	62	2689 s
Rennes (Paravent cluster)	198	2062 s
Sophia	160	1905 s
Total	944	1962 s

Table 1. Computers dispersion over Grid'5000 sites and characteristics

Table 1 shows the dispersion of processors across Grid'5000 sites. It shows also the execution time in seconds of the calculus code when the code is executed on a free processor, without concurrency.

The purpose was to evaluate the efficiency of the resource discovery service. To do that, we have measured the average execution time of the calculus process. If each of the 944 processes were simultaneously launched on 944 different processors, the average execution time of a process would have been 1962 seconds (see table 1). That corresponds to the execution time on a processor whose power would be the weighted mean of the powers of all the processors of our experiment.

To measure the efficiency of the resource discovery service of Vigne, we have performed five experiments where the task submissions have been spaced of 1, 2, 5, 10 or 20 seconds. We have not evaluated the global time of the execution of the 944 processes but the average time of a process execution. Thus, we have measured the ability for the resource discovery service to find the less loaded resources in order to execute the tasks.

Submission interval	Average execution time	Overhead
1 s	2248 s	14.6%
2 s	2143 s	9.2%
5 s	2057 s	4.8%
10 s	2042 s	4%
20 s	2039 s	3.9%

Table 2. Average execution time according to the submission intervals

Table 2 shows the average execution time for each experiment and the overhead of using our prototype compared to the average execution time of 1962 seconds. In these results, the larger the submission interval is, the smaller the execution time is. Indeed, if the submission interval is too small, the Vigne resource allocator is not able to evaluate accurately the load of the used resources. Thus, several processes may be executed on a single processor. In this case, the execution time is longer. However, the execution overhead is quite small regarding to the improvements in terms of automation or reliability. This overhead would be reduced with preemptive load balancing, because unbalances would be corrected after initial allocation. Furthermore, the time measured is larger than the real execution time. Indeed, detecting the end of an execution can last up to 60 seconds.

6. Related Work

We analyze related work in infrastructure choices, resource discovery and application management grid services.

6.1. Infrastructure

The architectural choices vary from centralized coordinators to fully decentralized and dynamic architectures.

BOINC-based projects and XtremWeb [2] target global computing using a centralized coordinator. This coordinator registers the resources ready to run some computations, and dispatches computations submitted by users. The coordinator is assumed to be stable and handles all the volatility of the resources. To achieve this stability, XtremWeb replicates the coordinator over a static set of nodes. These systems have proved to scale well in the context of the Internet but still impose a leader that hosts the coordinator, or a group of leaders hosting the coordinator's replicas. Moreover, the scalability of these approaches cannot be claimed for future use cases. In contrast, in our fully decentralized architecture one need not to assign any leadership to any node. Any node in the system can leave at will or even fail. Moreover, our local knowledge design principle is a stronger basis for scalability.

Most systems building scalable services use hierarchical approaches. For instance, Globus [4] extensively uses static hierarchies to build virtual organizations. Legion [10] uses a similar approach to name and locate objects. However, such architectures are very difficult to make self-healing, because the failure of one node in the hierarchy leads to the unavailability of all its sub-hierarchy. In particular, it is up to the administrators to define and repair these hierarchies. Moreover, without complex load balancing mechanisms, hierarchies suffer from contention in the high levels. In contrast, the overlay networks we adopted do not implicitly define hierarchies and tend to evenly distribute the load among the nodes.

GridOS [9] proposes to base naming, resource discovery, and resource allocation services on router-allocators, which are nodes that are organized in a similar way to routers in the Internet. The router-allocators of an organization maintain a global view of the organization's resources, and border allocators exchange summaries of their views between organizations. Like unstructured overlays, this mechanism is not well suited to access resources by location-independent names. Router-allocators could be made self-healing using the service mobility mechanism proposed by GridOS, but no automatic mechanism is provided to choose new nodes to run router-allocators.

Other works, including JXTA [17] and NaradaBrokering [11], 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 our use of key-based routing. 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 naming and resource discovery facilities we built. 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.

6.2. Resource Discovery

XtremWeb [2] provides a resource discovery mechanism based on a centralized coordinator. To run a job, a client sends a query that describes requirements based on attributes to a coordinator. When a resource is idle, it asks to the coordinator for a job whose requirements are fulfilled by resource. A such resource discovery mechanism insures good efficiency but is not transposable to a decentralized system.

To provide a resource discovery mechanism, Globus uses the Monitoring and Discovery System (MDS) [4]. Resources informations are stored in the MDS by resources and are periodically refreshed. In a WSRF-compliant way, entities called information sources provide a Web service interface for communications with MDS. With up-to-date attribute-based queries to get location of suitable resources. Globus addresses the scalability issue by providing a hierarchical approach to MDS that uses *Agregators*. However, MDS neither provides self organizing nor a self-healing features. Indeed, resource administrators must say toward which MDS aggregator their resource is linked to.

Punch (ActYP) [15] to perform the resource discovery stepresource allocation. ActYP is designed to optimize the response time of queries when queries are *similar*. ActYP is composed of one or more resource databases, a resource monitoring service that update regularly resource databases and a resource management pipeline. The resource management pipeline aggregates dynamically *similar* resources in order to optimize the response time of queries. Attribute-based queries are performed like classical queries in a database. As far as databases are distributed, ActYP has good scalability properties. However, the databases of ActYP rely on stable resource, what is not considered in the design of Vigne.

In [7], authors study the convergence between peer-to-peer and grid communities. They propose a framework along four axes to design resource discovery architecture. Since they propose to perform resource discovery in a grid over an unstructured overlay network, this work is the closer to our for the resource discovery service. They evaluate random walk based protocols through a simulator they have implemented. We have chosen a such distributed scheme to design the resource discovery service in order to profit of the self-healing and self-organizing properties that are intrinsic to such overlay networks.

Proceedings of The Fifth International Symposium
on Parallel and Distributed Computing (ISPDC'06)
0-7695-2638-1/06 $20.00 © 2006 IEEE

6.3. Application Management

Few projects include generic application management services to execute applications reliably. Chameleon [8] and XtremWeb [2] provide fault-tolerance mechanisms for a variety of programming models. 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.

7. Conclusion

This paper describes a general approach to build a self-healing scalable fully-decentralized SSI operating system for grids composed of a huge number of nodes. This paper brings two contributions. First we show that peer-to-peer overlays are a sound basis for building the distributed services of a GOS with the above properties. Moreover, we demonstrate that both structured and unstructured overlays are needed to meet the requirements of a GOS in terms of location-independent naming and of attribute-based distributed search. The cost of maintaining both kinds of overlay is reasonable as it is possible to take advantage of the structured overlay to maintain the unstructured overlay. We have built Vigne, a prototype of the proposed GOS. Experimentations carried out on the Grid'5000 grid platform demonstrate that the overlay infrastructure scales regardless the number of reconfigurations, the bandwidth consumption due to the overlay maintenance being very limited and constant whatever the number of nodes in the system. A resource discovery service and application managers have been implemented on top of the overlay infrastructure. Preliminary experiments show the efficiency of the resource discovery service. As a second contribution, our application managers make reliable application management decentralized and self-healing.

In future work we will experiment the resource discovery service with complex distributed applications, and we will experiment application managers in a fully dynamic system with various fault-tolerance policies to run applications reliably.

References

[1] Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. of ACM SIGCOMM 2003*, pages 407–418, 2003.

[2] S. Djilali, T. Herault, O. Lodygenski, T. Morlier, G. Fedak, and F. Cappello. RPC-V: Toward fault-tolerant RPC for internet connected desktop grids with volatile nodes. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 39, Pittsburg, PA, USA, Nov. 2004. ACM/IEEE, CS Press.

[3] A. J. Ganesh, A.-M. Kermarrec, and L. Massouli. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), Feb. 2003.

[4] Globus. Web Page : http://globus.org/.

[5] Gnutella. Web Page : http://www.gnutella.com/.

[6] Grid'5000. Web Page : http://www.grid5000.fr/.

[7] A. Iamnitchi, I. Foster, and D. Nurmi. A peer-to-peer approach to resource location in grid environments. In *Proceedings of HPDC 2002*, page 419, Aug. 2002.

[8] Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi, and K. Whisnant. Chameleon: A software infrastructure for adaptive fault tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):560–579, 1999.

[9] K. Krauter and M. Maheswaran. Architecture for a grid operating system. In *Proc. of the First IEEE/ACM International Workshop on Grid Computing*, volume 1971 of *Lecture Notes In Computer Science*, pages 65–76, Bangalore, India, Dec. 2000. Springer-Verlag.

[10] M. Lewis and A. Grimshaw. The core Legion object model. In *Proc. of HPDC 1996*, pages 551–561. IEEE, CS Press, Aug. 1996.

[11] S. Pallickara and G. Fox. 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*, pages 41–61. Springer, 2003.

[12] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, pages 127–140, Boston, MA,USA, June 2004. USENIX.

[13] L. Rilling and C. Morin. A practical transparent data sharing service for the grid. In *Proc. Fifth International Workshop on Distributed Shared Memory (DSM 2005)*, volume 2, pages 897–904, Cardiff, UK, May 2005. Held in conjunction with CCGrid 2005.

[14] A. Rowstron and P. Druschel. Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.

[15] D. Royo, N. H. Kapadia, J. A. B. Fortes, and L. D. de Cerio. Active yellow pages: A pipelined resource management architecture for wide-area network computing. In *Proc. of HPDC 2001*, page 147, Washington, DC, USA, 2001. IEEE Computer Society.

[16] F. Spitzer. *Principles of random walks*. Springer-Verlag, New York, 1976.

[17] B. Traversat, M. Abdelaziz, and E. Pouyoul. Project JXTA: A Loosely-Consistent DHT Rendezvous Walker. `http://www.jxta.org/docs/jxta-dht.pdf`, Mar. 2003.

Proceedings of The Fifth International Symposium
on Parallel and Distributed Computing (ISPDC'06)
0-7695-2638-1/06 $20.00 © 2006 IEEE