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

Emmanuel Jeanvoine, Louis Rilling, Christine Morin, Daniel Leprince

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1773

USING OVERLAY NETWORKS TO BUILD OPERATING SYSTEM SERVICES FOR LARGE SCALE GRIDS

EMMANUEL JEANVOINE AND LOUIS RILLING AND
CHRISTINE MORIN AND DANIEL LEPRINCE

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

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

Emmanuel Jeanvoine [*] and Louis Rilling [**] and Christine Morin [***] and Daniel Leprince [****]

Systèmes numériques
Projet Paris

Publication interne n ° 1773 — Novembre 2005 — 16 pages

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 a simplified view of a grid to users. Vigne deals with the very large 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.

Key-words: grid computing, overlay networks, application manager, resource allocation, Vigne

(Résumé : tsvp)

[*] Emmanuel.Jeanvoine@irisa.fr
[**] Louis.Rilling@irisa.fr
[***] Christine.Morin@irisa.fr
[****] Daniel.Leprince@edfgdf.fr

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes
Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Utilisation de réseaux logiques pour construire les services d'un systme d'exploitation pour des grilles de grande taille

Résumé : L'utilisation des ressources d'une grille pour exécuter des applications scientifiques qui requièrent une grande puissance de calcul est attractive mais compliquée du point de vue de l'utilisateur. Le système d'exploitation Vigne a été conu pour fournir une vision simplifiée de la grille aux utilisateurs. Vigne traite de la grande échelle d'une grille et du comportement dynamique des nuds en utilisant des réseaux logiques pair-à-pair comme fondation. Dans ce papier, nous montrons pourquoi il est très intéressant d'utiliser des réseaux logiques structurés et non structurés pour construire certains services de haut niveau du système d'exploitation Vigne. Pour montrer l'intérêt de notre approche, nous détaillons les fonctionnalités de deux services de Vigne basés sur des réseaux logiques pair-à-pair. Nous présentons aussi des résultats expérimentaux obtenus sur la grille d'expérimentation Grid'5000 qui montrent la capacité de l'infrastructure de Vigne, basée sur des réseaux logiques, à passer à l'échelle et son intérêt pour l'implémentation de services distribués.

Mots clés : grid computing, réseaux logiques, gestionnaire d'application, allocation de ressources, Vigne

1 Introduction

Nowadays, 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 cannot be provided by a single workstation. 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 very large amount of volatile resources distributed on different sites in a grid.

We propose a grid operating system, called Vigne, to simplify the use of a grid. One of the main features of Vigne is that its distributed services are based on an infrastructure composed of decentralized 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. We describe the way the two kinds of overlays are used in two services of Vigne operating system. We also discuss experimental results obtained executing Vigne on the Grid'5000 French national testbed and that show the scalability of the Vigne infrastructure and its practical interest on the example of the Vigne resource allocation service.

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 discuss the interest of using jointly structured and unstructured peer-to-peer overlays as a foundation for Vigne services. In Section 5 we detail two services implemented in Vigne system that use different peer-to-peer overlays. In Section 6, we present an experimental evaluation of the scalability of Vigne overlay infrastructure and provide preliminary performance results for the Vigne resource allocation service. We discuss related works in Section 7 and we finally conclude in Section 8.

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.

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.

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 sev-

eral near-coincident reconfigurations and failures, and reconfigure themselves to tolerate reconfigurations and failures again. In particular, all services must be decentralized.

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. Application managers are presented in Section 5.2.

To select the nodes on which an application should run, the application manager uses a resource allocator. 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 allocator is presented in Section 5.1.

3 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 in Section 4 that these requirements are satisfied by overlay networks designed in recent peer-to-peer research.

3.1 Naming and Exact Location

Building SSI services on top of a wide area distributed system needs location-independent global naming schemes. For instance, users should be able to access files without needing to know which nodes host these files. Similarly, users should be able to get the status of their running applications without knowing which nodes run their applications. 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. 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 presented in Section 2, the naming schemes used by the GOS services should be self-healing and content with local knowledge of the system state.

3.2 Attribute-Based Distributed Search

To do efficient resource allocation, the operating system should provide large scale search mechanisms. For example, to run an application, the resource allocation service must find a suitable set of nodes in the grid. Because the description of resources required by an application is quite complex, resource search can not be performed with an exact location mechanism. With an exact location mechanism, the content of the queries must 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 information 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 for resource discovery in a large scale environment.

In contrast with the location and naming requirements shown in Section 3.1, the search 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. This underlines the complexity of searches.

4 Using Jointly Structured and Unstructured Peer-to-Peer Overlays

According to our design principles, the services of a GOS should provide single system image, be self-healing, and content with local knowledge of the system. We presented in Section 3 the requirements that these principles generate on basic naming and location mechanisms. In this section, we show that peer-to-peer overlays suite these requirements. First, we present a short overview of decentralized peer-to-peer overlays. Next, we show that we should combine two types of overlay network. Finally, we present interesting optimizations to reduce the overhead of using two peer-to-peer overlays.

4.1 Two types of Peer-to-Peer Decentralized Overlays with Different Aims

Here, we only presents decentralized overlays because centralized ones may suffer of single point of failure and scalability problems. Thus, we introduce the concepts of *structured* and *unstructured* peer-to-peer overlays.

Structured Overlays The main concept in structured overlays is key-based routing. Such a routing implies that nodes select their neighbors according their names. In a structured overlay, each node is responsible of a set of keys. In Pastry [14], keys are organized around a logical ring and each node is spread on the ring in a clockwise order.

In a structured overlay, it is possible to find any key from any node in a limited number of hops. For example, in Pastry the average number of hops is $log(n)$, where n is the number of nodes in the system.

Structured overlays can be enriched by adding distributed hash tables to provide self replication to entities.

The main asset of structured overlays relies on their ability to locate with exactness any entity named with a key in the system.

As a main drawback, it is difficult to find an entity knowing only a description not including its key. This issue is critical if the description includes several criteria, especially if some of the criteria are expressed as ranges of values or left unspecified. In this case, load balancing issues may occur because it is difficult to spread entities around all keys, i.e., around all nodes.

Unstructured Overlays In contrast to structured overlays, in unstructured ones like Gnutella [8], nodes connect themselves to the system without taking care of their neighbors name. However, nodes connections may take into account load balancing criteria or other policies.

As soon as a node joins the system, it establishes several connections with other nodes, called neighbors. 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.

The main advantage of unstructured overlays relies on their ability to perform complex searches, even those based on multi-criteria range queries. Nevertheless, the cost of flooding the network might be prohibitive if the depth is high. To decrease this cost, optimizations have been studied [16, 4].

The main drawback of search in unstructured overlays is that it is impossible to certainly find a given entity or to make an exhaustive search.

4.2 Why Use Both Type of Overlays ?

In Section 4.1, we show that structured and unstructured peer-to-peer overlays have divergent aims. As a single kind of overlay can not address all requirements for building operating system services, we choose to use both overlays as a base for designing Vigne.

Requirements in terms of naming and exact location can be fulfilled by a structured overlay. For this reason, we implemented a structured overlay based on the Pastry routing algorithms [14] and on 256 bits naming spaces. Such naming spaces are large enough for identifying nodes in the Grid or other objects like application managers or shared memory segments.

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.

The need for complex search through a large scale system is fulfilled by unstructured overlays. We implemented an unstructured overlay whose membership protocol is based on Scamp [6]. 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 the structured overlay implemented in Vigne, each node has a routing table that contains $log(n)$ other nodes (where n is the number of nodes in the system), and a leaf set of 24 neighbors. Considering size of the routing table, messages can

be routed to any node in about $log(n)$ hops. Furthermore, leafset is large enough to prevent the system from partioning in case 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).

Finally, scalability is intrinsic to decentralized peer-to-peer overlays. Regarding unstructured overlays, basic flooding might be a performance issue but a Random Walk mechanism would sidestep it. Both overlays have good properties, by design, regarding dynamic behavior and fault tolerance.

4.3 Optimizations and Limited Overhead

We show here that using both types of peer-to-peer overlays is not wasteful. In Vigne, we reduce this overhead by taking into account the existence of the structured overlay for implementing the unstructured one.

First of all, the unstructured overlay shares the naming scheme of structured one. Thus, if a remote node is in both overlay neighborhoods of a node, the remote node informations are not duplicated on the local node.

Then, with Scamp protocol [6], there is a problem if all nodes use the same bootstrap because the neighborhood of all nodes will be empty. In Vigne, we solve this problem simply by choosing a random node in the neighborhood of the structured overlay. This is possible because our implementation of the structured overlay ensures that its neighborhood is never empty if the system is composed of more than one node and because a node joins the grid firstly by joining the structured overlay and later the unstructured one.

Furthermore, assuming that the neighborhood of a node in the structured overlay is never empty, Vigne uses a simple mechanism for the self-healing property of the unstructured overlay. Indeed, if the neighborhood in the unstructured overlay becomes empty, the node has just to re-join the unstructured overlay by choosing randomly a bootstrap in the structured overlay neighborhood.

In order to deal with dynamic behavior, both peer-to-peer overlays must detect node failures. In Vigne, we use the same failure detector for both overlays. Furthermore, if a distant node belonging to both overlays has a failure, this distant node will be discarded from both overlays as soon as structured or unstructured overlay detects the failure.

In Vigne, we reduce the cost of using structured and unstructured peer-to-peer overlays because we have implemented a lightweight unstructured overlay that uses several properties of the structured overlay. Thus, Vigne benefits of both type of peer-to-peer overlay at a low cost.

5 Examples of Services in Vigne

We present two services implemented in Vigne and based on the use of a decentralized peer-to-peer overlay: resource allocation and application management. We assume in the

remainder that each node of the structured or unstructured overlay represent a node in the grid.

5.1 Resource Allocation

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 and allocate free resources in the grid to run applications.

The resource allocation service has two main features: resource discovery and resource selection.

Resource discovery consists in finding a set of resources corresponding to a specification provided by the user in order to run his application in good conditions. The specification includes several requirements of the application such as the processor architecture, the number of processors, the operating system, the cluster scheduler, the libraries, the memory, the 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. In a structured peer-to-peer overlay, it is difficult to perform searches if queries have not scalar attributes. Indeed, it is hard to find a correct hash function for such attributes. Searches can be done with a great expressiveness with an unstructured overlay. That fulfills all of our requirements in terms of multi-criteria and range-value based searches. For these reasons, we decided to use an unstructured peer-to-peer overlay to perform resource discovery in Vigne.

We implemented three 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.

After a resource discovery step, a set of resources is retrieved and the most suitable resource is chosen according to a specified policy. Currently, our prototype chooses the less loaded suitable nodes but other policies can be implemented as well. For instance, it would be interesting to choose a resource at a short distance from a file server in order to minimize the cost of fetching application files. It would also be interesting to develop policies for resource co-allocation, for example to run code coupling applications.

5.2 Application Manager

In the Vigne operating system, the application manager service is a crucial point in relation to applications life. The application manager service ensures that applications complete correctly.

As the application manager must provide correct execution of applications, we introduce fault tolerance concepts. Adaptable policies can be used. For instance, when the application manager detects the failure of a node where an application was launched, it can re-execute the application on another node. This policy can be enhanced by periodically checkpointing

the application. Thus, the application may terminate correctly despite failures. Other policies like replicated execution are also possible.

When a user submits an application, an application manager is created and is responsible for the life cycle of application. To simplify the user's interactions with the application manager in case of reconfigurations or failures, the application manager is represented by a key in a DHT. As a result, at submission time, the system computes a SHA-1 hash of the application binary MD5 sum, timestamp, name of host triplet. The result is a key in the structured peer-to-peer overlay naming space. Thanks to SHA-1 properties, collisions are extremely rare, and we can assume that each generated key is unique. If generated key already exists, error code is returned to user, which implies that he must re-submit the application. However, such collisions should occur seldom if ever. Once the unique key is obtained, a message is routed to the node in charge of the key in order to create the application manager.

Thanks to its unique key, an application manager can be reached by any entity of the system. For instance, if a user wants to know the progression status of the application execution, he routes a request to the application manager's key. A resource informs the application manager that the execution has completed using the same mechanism.

Since DHTs based on structured overlays provide location-independent names even if reconfigurations occur, an application manager is transparently moved to an other node when the node hosting it is disconnected from the system. For example, this may happen when an administrator must perform offline operations on a node. In this case, if the node hosts an application manager object, the application manager will be moved to the node newly responsible for the key of the application manager. As a result, when this application manager is contacted again, the message is routed to its new host node. With that property, structured peer-to-peer overlay ensures that the application manager service will work even in the event of a node departure.

As application managers are created anywhere, maybe on unreliable resources, they can die prematurely, before the end of the application execution. So, the GOS must also be able to rebuild disappeared application managers with the same state that before failure. As structured overlay offers conveniences for object replication in DHT, application managers can be automatically replicated. Thus when an application manager disappears, the GOS transparently uses a replica and the execution of the application survives despite failures.

5.3 Discussion

We have presented two services that use a different peer-to-peer overlay with different aims.

The resource allocation service is based on an unstructured peer-to-peer overlay because it must perform complex searches with high expressiveness.

Otherwise, the application manager service is based on a structured peer-to-peer overlay, that provides exact and reliable naming and location features. Thus, any node can lookup any application manager just knowing its key. Furthermore, structured overlay provides fault tolerance mechanisms like automatic duplication of objects. This is useful for duplicating application managers and deal with failures.

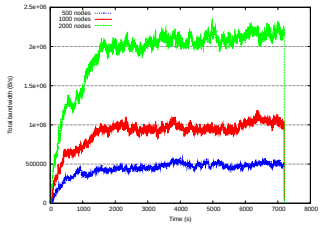

Figure 1: Global bandwitdh consumption

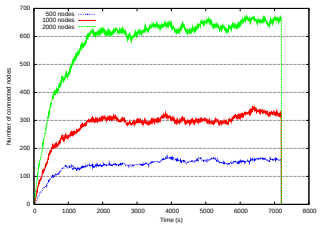

Figure 2: Number of nodes connected

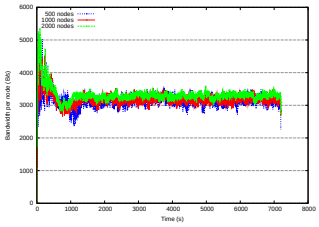

Figure 3: Bandwidth consumption per node

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

6 Experimental Results

In order to validate the efficiency of Vigne, we conducted two kinds of experiments, both on the Grid'5000 testbed [9]. Grid'5000 is a national testbed for grid software where nodes are currently spread over seven sites.

6.1 Scalability of the Overlay Infrastructure

We validate here our approach in terms of scalability. We present an experiment that characterizes the amount of consumed bandwidth by both peer-to-peer overlays. Indeed, the structured overlay must maintain routing tables and leafset and the unstructured overlay must maintain neighborhoods. This is fundamental in a dynamic environment.

We have used one hundred 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.

In this experiment, we have considered a given number of logical nodes (500, 100 and 2000). As the number of logical nodes is higher than the number of physical processors, more than one logical node run on a physical processor.

Nodes arrivals and departures in the system have been simulated with a Poisson law.

This experiment underlines the self-healing properties of Vigne. Indeed, with such a dynamic behavior, both peer-to-peer overlays must deal with multiple node failures and so, they must continually re-configure themselves. Thus, if Vigne is able to deal with this hostile dynamic behavior, it will be able to work fine in a less hostile and large scale environment, closer to reality in an industrial use of grid.

Figure 1 shows the total bandwidth consumption and Figure 2 shows number of nodes connected. So if we take into account the total bandwidth divided by the number of nodes, this ratio is constant whatever the number of nodes (Figure 3). Furthermore, the bandwidth consumption per node is reasonable with an average value of 3 KB/s.

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

6.2 High Level Service Evaluation

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

We have used 373 computers belonging to three Grid'5000 sites (Bordeaux, Orsay, Rennes). Computers are bi-processor AMD Opteron at 2.0 GHz, AMD Opteron at 2.2 GHz and Intel Xeon at 2.4 GHz.

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

Grid'5000 site	CPU used	Execution time
Bordeaux	88	1740 s
Orsay	416	1955 s
Rennes (Paraci)	128	1593 s
Rennes (Parasol)	112	2689 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.

We launched 746 jobs (as much as processors) materialized by the calculus code, every 20 seconds. We underline here the ability of the resource allocation service to find a free resource to run a job in order to avoid overload of a small number of processors and inactivity of others. Indeed if more than one job is launched on a processor, its execution time will strongly increase.

If all jobs are uniformly spread on all processors, the average execution time is the average execution time given in Table 1 and balanced according the number of processors per site. Thus, it is 1972 seconds.

In our experiment, the average execution time of a job is 1918 seconds. Our result is lightly better than in the case where jobs are uniformly spread. This shows the efficiency of the resource allocation service.

The better average run time in our experiment is due to the fact that jobs are not launched at the same time and faster processors are used to run more jobs than slower ones.

7 Related Works

We analyze related works by considering naming, resource discovery and application management grid services.

7.1 Naming and Exact Location

To locate replicas of files, Globus provides a Replica Location Service (RLS) base on a hierarchy of Replica Location Index (RLI) [5]. However the RLS is not self-healing: administrators have to specify on which nodes the RLIs run. In [2] a self-healing RLS based on a structured overlay network is proposed, but the proposed scheme remains limited to locating read-only files.

Legion and Globe are object-based systems that locate objects using a location-independent naming service. To communicate with an object, an entity first retrieves an address from the naming service, and then locally binds a reference to the object to the returned address. Legion [12] uses a hierarchical home-based location scheme in which class objects register the locations of all the objects of a given class. Besides the scalability issue on the number of objects of a given class, the location service of Legion ultimately relies on a root class object which is a single point of failure.

Globe addresses the scalability issue using a distributed search tree organized according to the network distances between nodes and objects [1]. Globe also supports replicated objects as a fundamental property. However the distributed search tree is statically defined by administrators, and only tolerates temporary failures.

GridOS uses a global naming service called gridspaces, that is targeted to resource discovery and allocation [11]. In each organization participating to the grid, nodes called router-allocators maintain a gridspace, that registers the names and properties of all the resources of the organization. Router-allocators exchange summaries of their gridspaces between organizations. Like unstructured peer-to-peer overlays, gridspaces are not well suited to access resources by location-independent names. Gridspaces 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.

With Active Names [17] an entity accesses an object through a chain of name resolvers. A name resolver processes the asked name and can transform and transfer the request to another name resolver. Moreover, a name resolver can add an after-method to the request, which is popped and executed once the requested has reached its destination. Active Names offer a flexible architecture to define various location mechanisms and policies to optimize request post-processing, but only provide a low-level architecture. The DHT of a structured peer-to-peer overlay could be implemented using Active Names.

7.2 Resource Discovery

To provide a resource discovery mechanism, Globus uses the Monitoring and Discovery System (MDS) based on XML documents and Xpath queries [7]. 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 XML databases, users can query the MDS with complex queries to get wanted resource location. Globus addresses the scalability issue by providing a hierarchical approach to MDS that uses *Agregators*. However, MDS neither provide a self organizing

nor a self-healing system. Indeed, resource administrators must say toward which MDS aggregator their resource is linked to.

Punch uses Active Yellow Pages (ActYP) [15] to perform the resource discovery step in resource allocation. ActYP is composed of one or more resource databases, a resource monitoring service and a resource management pipeline. The resource management pipeline aggregates dynamically *similar* resources in order to optimize the response time of queries. ActYP addresses the scalability issue but it is efficient only when queries are *similar*. According to authors, such situations often occur, for instance when a large class is working on a homework assignment.

XtremWeb [3] is based on a client/coordinator/worker pattern. When a worker joins the system, it contacts a coordinator and sends the resource characteristics that it provides. To run a job, a client sends a query that describes requirements to a coordinator. As XtremWeb is based on the *pull* model, idle workers ask the coordinator for jobs. As a consequence, clients queries for resource discovery can be fulfilled easily because comparison is made on coordinator nodes. XtremWeb addresses the scalability issue by using a hierarchy of coordinators in the system. However, coordinators are supposed to be run on a stable resource so fault tolerance is not took in account for coordinators.

7.3 Application Management

Chameleon [10] provides an infrastructure to execute applications in a fault tolerant context. Chameleon introduces ARMOR mobile objects to handle fault tolerance. For example, Chameleon has checkpoint ARMORs, heartbeat ARMORs, execution ARMORs or a fault tolerance manager (FTM) ARMOR. The FTM ARMOR oversees other ARMOR objects and applies a fault-tolerance policy to application execution according to the user requirements. FTM ARMOR is centralized but Chameleon provides a consistent backup FTM in case of failure of the main FTM ARMOR.

XtremWeb [3] provides a similar service for management of applications with the scheduler component of the coordinator. The scheduler instantiates services and application, manages their life cycle and provides fault tolerant policies. For multiple reasons discussed in [3], the coordinator is implemented in a centralized way.

8 Conclusion

This paper describes a general approach to build a self-healing scalable fully-decentralized single system image operating system for grids composed of a very large number of nodes. We show that peer-to-peer overlays are a sound basis for buidling 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 allocation service and an application manager have been implemented on top of the overlay infrastructure. Preliminary experiments show the efficiency of the resource allocation service.

References

[1] Gerco Ballintijn. *Locating Objects in a Wide-area System.* PhD thesis, Vrije Universiteit Amsterdam, 2003.

[2] Min Cai, Ann Chervenak, and Martin Frank. A peer-to-peer replica location service based on a distributed hash table. In *Proc. of ACM/IEEE SC 2004*, page 56. ACM/IEEE, CS Press, November 2004.

[3] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky. Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, 21(3):417–437, March 2005.

[4] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proc. of ACM SIGCOMM 2003*, pages 407–418, 2003.

[5] Ann L. Chervenak, Naveen Palavalli, Shishir Bharathi, Carl Kesselman, and Robert Schwartzkopf. Performance and scalability of a replica location service. In *Proc. of HPDC 2004*, pages 182–191. IEEE, CS Press, June 2004.

[6] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massouli. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), February 2003.

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

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

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

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

[11] Klaus Krauter and Muthucumaru 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, December 2000. Springer-Verlag.

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

[13] Louis Rilling and Christine Morin. A practical transparent data sharing service for the grid. In *Proc. Fifth International Workshop on Distributed Shared Memory (DSM 2005)*, Cardiff, UK, May 2005. Held in conjunction with CCGrid 2005.

[14] Antony Rowstron and Peter 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)*, November 2001.

[15] Dolors Royo, Nirav H. Kapadia, Jos A. B. Fortes, and Luis Diaz 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] Frank Spitzer. *Principles of random walks*. Springer-Verlag, New York, 1976.

[17] Amin Vahdat, Michael Dahlin, Thomas Anderson, and Amit Aggarwal. Active names: Flexible location and transport of wide-area resources. In *2nd USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, US, October 1999. USENIX.