An Information Brokering Service Provider (IBSP) for Virtual Clusters

Roberto Podesta', Victor Iniesta, Ala Rezmerita, and Franck Cappello

INRIA Saclay-Ile-de-France
Bat. 490 Université Paris-Sud
91405 Orsay, France
{podesta,Victor.Iniesta,rezmerit,fci}@lri.fr

Abstract. Virtual clusters spanning over resources belonging to different administration domains have to face with the computational resources brokering problem in order to allow the direct interaction among them. Typically, it requires some component collecting, elaborating and providing resources information. Currently, every virtual cluster project provides its own solution with a low degree of portability and isolation from the rest of the platform components. We propose the Information Brokering Service Provider (IBSP), a general approach which wants to represent an uniform solution adaptable to a wide set of existing platforms. Its pluggable and flexible design is based on a decentralized architecture where each of the cooperating nodes is composed by a resource interface exposing the brokering service, and a portion of a distributed information system. In this paper, we present the motivations, principles and implementation of IBSP as well as the performance of IBSP serving the Private Virtual Cluster system.

Keywords: Brokering, P2P, Virtual Clusters.

1 Introduction

The convergence between Peer to Peer (P2P) and Grid Computing [1] contributed to the advent of projects aiming to port the cluster abstraction of computational grids on top of common machines. Virtual cluster projects like VioCluster [2], Wide-area Overlay of virtual Workstations [3] (WOW), Private Virtual Cluster [4] (PVC), Cluster On Demand [5] (COD), etc. address this challenge. Typically, they rely on overlay networks of Internet-connected machines and aim to provide a transparent view of a single cluster where parallel applications may run unchanged.

Similar concepts are also adopted in the emerging cloud computing [6]. A transparent view of grouped machines is obtained by a cloud interface masking a distributed back-end where cluster virtualization techniques are adopted. The computational resources leveraged by clouds are physical clusters rather than common Internet-connected machines. Compared to P2P-based cluster virtualization, cloud computing gives a greater relevance to a clean user interface which exposes methods to configure the whished computational environment. Samples are the widely known Amazon

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 165–182, 2009.

Elastic Compute Cloud [7] (EC2), based on a proprietary technology, and the open source Nimbus [8] and Eucalyptus [9].

P2P-based cluster virtualization presents not trivial issues related to the pooling of the computational resources. The arising problems may span from networks barriers to the lack of knowledge of resources location. These practical obstacles imply the adoption of components able to perform resources inter-mediation, information management and discovery. Such tasks involve a set of needs we define as brokering issues. Currently, every project provides its own customized solutions. It is difficult to individuate a reference model because the implemented mechanisms are often not completely isolated from other aspects of the platforms. Their level of reusability is consequently very low, even if the intrinsic nature of the handled problems is usually shared. A standardized and uniform solution might greatly simplify the deployment of current virtual cluster platforms and would free the development of future virtual cluster projects from a not negligible burden. In this paper, we present a possible solution to this challenge. Upon a categorization of brokering issues, we adopt the Service Oriented Architecture (SOA) [10] model to design a software entity providing a general resources brokering service, an Information Brokering Service Provider (IBSP) for virtual clusters. The widely known SOA pattern implies a single entity acting as the Service Broker, which exposes the information of services published by Service Providers. Service Consumers get that information from the Service Broker, and they can subsequently invoke the services on the Service Providers. In our case, the services become computational resources. The system architecture is built to maximize the flexibility in order to fit a wide range of possible requirements. Therefore, it leverages a distributed structure, which can grow and shrink depending on the optimization needs of the served platforms. The system design adopts a pluggable approach to allow the dynamic set-up of the modules supporting specific platforms.

The remainder of the paper is organized as follows: in section 2, we draw the categorization of brokering needs in a representative set of scenarios; section 3 describes the operational model, the software design and some implementation highlights of IBSP; section 4 presents the real applicative scenario we used to test the IBSP reference implementation, and the test results we obtained; and, finally, section 5 concludes with some remarks and future works.

2 Drawing a Categorization of Brokering Issues

The resources brokering problem is not only present in the domain of cluster virtualization. In fact, similar issues are tackled in several domains, and different tools solve them with approaches and architectural solutions which have to be considered. Furthermore, interesting issues are coming from the cloud computing adjacent field, where many cluster virtualization concepts are adopted. Table 1 lists a representative set of domains with some tool examples and the relative brokering issues. The last column shows the broker software architecture of those tools. The highlighted categorization of brokering issues has a double aim: first, to ease the idea of uniform brokering service provider by a clear list of needs in virtual cluster platforms; and, second, to explore possible intersections in other domains.

2.1 Virtual Cluster Platforms

VioCluster consists in a combination of the VIOLIN [11] network virtualization and XEN [12] machine virtualization and it is targeted to build a computational resources trade among computational clusters belonging to different administrative domains. The brokering issue in VIOLIN is bounded to the overlay namespace resolution which reflects a three layer hierarchy of entities (virtual routers, virtual switches and virtual machines). The WOW project aims to provide a communication infrastructure overlaying the Internet and routing messages among a set of world wide distributed virtual workstations. The brokering needs of WOW consist in the resource enrollment, detection and lookup. The adopted system consists in a distributed DHCP implementation. The DHCP client residing on each VM is redirected to the TAP virtual driver and encapsulated in a SOAP request, which accesses to a distributed database containing the configuration of the virtual ID (e.g. a class of private IP addresses). The PVC system builds an overlay network by the adoption of a virtual ID system corresponding to IP addresses belonging to the experimental class E. A central registry maps those virtual IPs to real IPs thus allowing the communication between resources. Moreover, the registry works also as resource connectivity enabler to allow the direct connection between peers. The COD [13] project uses a resource broker to manage the resource leases. Other aspects like resource identification, lookup and detection are managed by its overlay network. Virtuoso [14] is a virtual cluster system pooling dynamically virtual machines connected through an overlay network named VNET. The brokering issues in Virtuoso refer to virtual machines identification, lookup, detection and information. They are bounded to the overlay routing ruled by a

Table 1. Categorization of brokering issues

DOMAIN	TOOL	BROKERING ISSUE	SOFTWARE DESIGN
Web Services	UDDI	Yellow pages; White pages; Green pages	Centralized / UBR
Meta-computing	H2O	Service lookup	Centralized
Enterprise	JNDI	Remote Object Lookup	Centralized
Sensor Network	Data Fusion	Semantic based asynchronous object lookup	Centralized
Sensor Network	JAIN SLEE	Asynchronous object lookup	Centralized
Overlay Network	Narada	Tree-based routing	Hierarchically distributed
Virtual Cluster	Viocluster (VIOLIN)	Resource enrollment, identification, detection, lookup; network information	Hierarchically distributed
Virtual Cluster	PVC	Resource enrollment, identification, detection, lookup; network information	Centralized
Virtual Cluster	WOW	Resource enrollment, identification, detection, lookup; network information	Distributed DHCP
Virtual Cluster	COD	Resource enrollment, identification, detection, lookup; network information; resource information	LDAP
Virtual Cluster	Virtuoso (VNET)	Resource enrollment, identification, detection, lookup; network information	Hierarchically distributed
Virtual Cluster	ViNe	Resource enrollment, identification, detection, lookup	Distributed
Cloud	Nimbus	Hypervisor management; VM Images localization; resource information	Centralized registry
Cloud	Eucalyptus	VMs group management / configuration; VM Images localization; resource information	Hierarchically distributed
Cloud	Enomaly	Hypervisor management; VM Images localization; resource information	Centralized registry

statically configured two-level hierarchy of elements using an ID system based on MAC addresses tunneling. The ViNe[15] project is a communication infrastructure overlaying internet similar to WOW. It uses private IP addresses as ID systems which are configured through IP aliasing on the machine NIC interface. The resource lookup and detection is ruled by virtual routers onto which all the traffic among resources is redirected.

2.2 A Look at Other Domains

The Universal Description Discovery and Integration [16] (UDDI) standard plays the role of service broker in Web Services technology. Sites exposing Web Services register them on an UDDI broker. Web Services consumers looking for a certain type of service perform a request to an UDDI broker and obtain a network reference enabling the direct communication. The service provided by UDDI offers three types of possibilities: the white pages (e.g. it replays to a query wanting a specific business name), the yellow pages (e.g. a certain type of business) and the green pages (e.g. a certain type of technical information about a business). Therefore, it provides more than simple lookup by providing elaborated information in order to allow the direct communication between business entities. Typically, UDDI registries are isolated centralized server. The UDDI Business Registry (UBR) project was the only effort targeted to build a federation of UDDI registries. This project was also known as UDDI cloud, and federated a limited number of UDDI servers belonging to the project partners (i.e. IBM, Microsoft, SAP and HP). The basic mechanism behind the federation was a periodical propagation of the entries onto all the registries [17]. However, UBR ended in 2006 and the partners shut down the UDDI cloud [18]. Among the others, a motivation of the UDDI cloud end was the poor performance of the federation mechanism [19] based on a not optimized, pure data replication. The Narada Broker [20] is a distributed system targeted to route messages among remote nodes by supporting publish/subscribe communication and P2P interaction. Narada provides compatibility with already developed asynchronous communication standards like Java Message Service and JXTA [21], and has been adopted in several applicative scenarios. Narada works as overlay transport layer by taking care of all the communication among nodes composing the application using it. It is a middleware working as the glue connecting remote peaces of a distributed application. Narada adopts a hierarchical topology with master brokers and normal brokers. The resources brokering problems in Narada result in registration and lookup. They are solved by the hierarchical namespace ruling also the routing on the overlay transport layer. The table contains further samples of brokering needs of tools operating in various domains. The widely known JNDI registry serves the remote object look-up in enterprise application based on Java. The meta computing framework H2O [22] uses an extension of the Java RMI registry to lookup the deployed services. Some interesting solution comes from the sensor network world. The Semantic Data Fusion for sensor networks [23] introduces an ontology-based broker in a publish/subscribe system to get a smart integration of the data coming from sensors. Publishers and subscribers communicate via declarative grammar their respective notifications and interests to the semantic broker. Afterwards, its engine performs the matching thus enabling the smart information exchange. The Activity Context of the Java API for Intelligent Network Service Logical Execution

Environment (JAIN SLEE) [24], a Java standard targeted to event driven system, is also familiar to this approach. The Activity Context allows a smart event routing through declarative programming and a JNDI registry thus enabling the connection between asynchronous network events sources and Java components.

2.3 Cloud Computing

Cloud computing infrastructures like Nimbus [1], Eucalyptus [2] and Enomaly [3] coordinate distributed resources by the adoption of centralized registries working in cooperation with remote agents deployed on the hypervisor installed on each machine. Even if there are some slight differences, basically they use a master-worker information system where the master is the registry collecting information about the workers represented by the daemon running on the managed machines. Nimbus and Enomaly build this mechanism on top of a two-level hierarchy condensing the master registry in the cloud interface. Eucalyptus decouples the cloud interface from a set of super registries with a hierarchical 3-tier architecture. A cloud-level registry mediates the composition of virtual cluster instances based on physical resources collected by cluster-level registries. These architecture types reflect the normal target of resources set hidden by a cloud interface, namely a computational cluster. The handled information assumes the form of meta-data describing the groups of virtual machines fitting a specific request coming from the interface exposed to the cloud users.

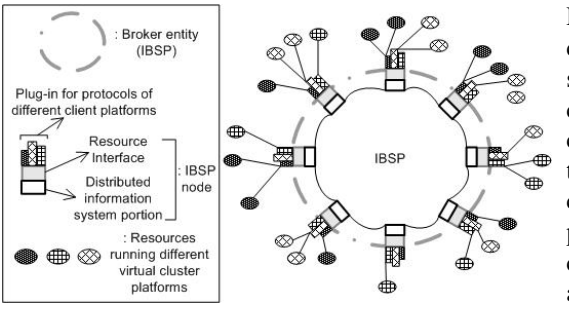

Fig. 1. IBSP Architecture

3 Information Brokering Service Provider (IBSP)

The operational model of IBSP derives from the categorization of the brokering issues shown in the previous section. We looked firstly at the P2P-based virtual cluster platforms which are the main target of our work. We extracted from that table the various brokering needs in order to compose a set of general functionalities. A look at the brokering issues in other fields is useful to reuse already developed concepts even if a completely re-usable solution does not exist. The table suggests also possible intersections with cloud platforms. However, the different kinds of used computational resources (i.e. computational clusters for cloud computing, common machines for P2P cluster virtualization) and the different targets (i.e. precise fulfillment of user requests for cloud computing, maximization of utilization of Internet-connected machines for P2P cluster virtualization) induced us to focus the broker functionalities definition on the needs of P2P-based virtual clusters we have shown in section 2.1 (i.e. VIOLIN, PVC, WOW, COD, VNET, ViNe).

3.1 IBSP Specification

The categorization shows several brokering issues shared among different platforms. Therefore, we extract the following list of functionalities: 1) resources enrollment, 2) resources identification system, 3) resources lookup, 4) resources presence detection, 5) resources fault detection, 6) resources connectivity enabler, and 7) resources information system.

Note that the virtual cluster platforms calling IBSP functionalities are referred as client platforms in the remainder of the paper. The rest of this sub-section examines the functionalities offered by IBSP. 1) The enrollment regards the registration of the resources into the broker. After this step, the broker is able to provide information about them. This functionality implies the exposition of an enrollment method for every type of resource required by a client platform. 2) The broker provides an ID system which is used to identify resources. An unique ID is assigned to a resource. An ID system has to be configured before the startup of the broker. The ID systems can be ranges of private IP addresses, namespaces or any other identification system required by a supported client platform. The assignment of IDs can be done by the broker (in a pseudo-DHCP way) or statically configured by the client platform. In both cases, the broker maintains the IDs consistence. Different ID systems can coexist within the broker in case of broker instance serving concurrently multiple platforms. This functionality is not bounded to the exposition of a specific method, while it is rather linked to the enrollment method. In fact, the invocation of the enrollment method ends with the memorization of an association between an unique ID and a resource representation, which includes at least its physical location. 3) The simple resource lookup is provided. Therefore, the broker exposes a simple lookup method to retrieve a target endpoint (e.g. physical location as an URL or an IP) when a resource is looking for another resource given a certain target ID. An extension of this method provides the additional information possibly associated to a given ID more than the mere location information. 4) The presence detection functionality serves to verify the actual participation of a given resource. A resource participates in a virtual cluster if it is enrolled into the broker. A resource presence detection method confirms or not if a resource is enrolled by a given ID. 5) The resource fault-detection may appear similar to the previous functionality. However, it refers to the notification of a resource failure. This functionality is offered though a subscription mechanism. A resource subscribes its interest in the aliveness of a target resource, and it is notified in case of fault of that resource. The subscription can be explicitly done by a resource or automatically performed by the broker which subscribes the interest of a resource in another one if the first one has previously looked up the second one. 6) The resources connectivity enabler refers to the actions performed by the broker to set-up the connectivity between two enrolled resources. It is an inter-mediation functionality, it supports the start-up of the connection between two resources, which otherwise would not be able to communicate. After this set-up phase, the resources communicate directly without any other broker intervention. The connectivity enabler functionality relies on schemas of messages relay between the requester resource and the requested resource through the broker. Those schemas may comprise a simple request/reply messages exchanges as well as complex inter-mediations composed by messages loops involving the requester, the callee and even third-part resources. Forwarded

messages can be enriched and modified by the broker according to specified pattern depending on the client platform. This functionality can be useful for a variety of situation that can span from network information (e.g. bypass of routers and firewalls, overlay routing rules, etc.) to public security information exchange. This functionality is mapped to a set of connectivity methods depending on the client platform. Finally, 7) the broker has a flexible information system where resources information are stored. Basically, a resource is registered with an association between an unique ID and a representation of its actual location. However, typical virtual cluster platforms require more than just location (e.g. system information, leases, etc.). Therefore, it is possible to add additional information to an entry representing a resource. Such information can be registered at the enrollment stage as well as by an update method exposed to modify the entries content. Furthermore, the information system serves to provide a resource discovery not based on a precise ID. It can be useful for those cases requiring a resource search on the base of some feature without an a-priori knowledge of the resource ID. This kind of search can be considered rare compared to the ID-based search. However, it can be useful for a noticeable variety of cases (e.g. search for specific computational capabilities, specific software configuration, etc.). IBSP support this capability by a publish/subscribe based system. Every module supporting a client platform can implement an application grammar used by resources to publish topics which are useful for a not ID-based resource discovery. Resources can then subscribe to topics according to the module grammar. Such a grammar can be composed by a few keywords or based on some ontology. This approach resembles the semantic-based data fusion for sensor network mentioned in section 2.2.

IBSP structure wants to maximize its flexibility in order to be adaptable to a large set of client platforms. Mainly, it aims to: 1) avoid single points of failure, 2) provide multiple access points, 3) allow both loosely coupled and permanent connections between the broker and the resources. The first and the second constraints force a distributed structure where all the participating nodes have the same importance without any hierarchy. The third one represents an exception to the native SOA model. It is recommended for resources not able to receive incoming connections because they are hidden by NATs or firewalls.

3.2 Implementation Choices

The resulting architecture is a distributed set of cooperating nodes constituting the whole broker entity. Each node hosts a resource interface and a portion of a distributed information system. The first exposes the broker functionalities to resources, and different plug-ins implement the protocol syntaxes and the semantics required by the supported client platforms. Figure 1 shows a graphical representation of the architecture of IBSP. The number of composing nodes can dynamically grow and shrink. The assignment of this number can be done by some heuristic depending on the actual situation requiring IBSP. If it is serving just a single client platform, the number of IBSP nodes can be the optimal one calculated for that platform. If it is serving multiple client platforms, the maximum one can be used. However, the calculation of the optimal number of the IBSP instances required by a singe or more client platforms may depend on a wide set of factors. Just for a single client platform, reflections coming from the overlying parallel application as well as particular administrative situations can influence the number of the needed IBSP instances. Section 4 presents a set

of experiments we ran in different case studies which contribute to ease the assignment of the number of IBSP instances. We focused on P2P data distribution models to design the distributed information system to avoid hierarchical layers bringing to critical points of failure. We selected the structured overlay network approach which relies on Distributed Hash Tables [28,29](DHTs). The meaning of a DHT is a hash table distributed among a set of cooperating nodes. As a normal hash table it allows to put and get items as key-value pairs. IBSP stores into a DHT the ID generated for an enrolled resource as key. The value associated to a key has to provide a certain information about the resource. That value can be a string reassuming directly all the required data depending on the client platform (e.g. resource actual location, resource information) or a network pointer (e.g. an URL) to a file containing that information. It depends on the amount and the nature of the data. The first solution is obviously faster and, therefore, preferable if applicable. We prefer the DHT adoption to the unstructured overlay network because DHTs provide better scalability. Moreover, the usage scenario of the broker should not affect the main known drawback of DHTs. Their main capability is to provide a scalable logarithmic keys lookup time. The scalability is not a trivial issue in unstructured overlay networks being based on queries broadcasting. The solution consists in the layering of nodes into a hierarchical classification. However, it introduces critical points of failure in the structure and does not nullify completely the lack of scalability. Unstructured overlay networks have actually good capabilities as the possibility to perform range queries which are not possible with DHTs. To overcome this limitation, DHT implementations provide basic APIs to perform multicast communication based on the publish/subscribe pattern. This capability is particularly useful for the functionalities exposing methods replaying to not ID-based requests (e.g. fault detection and some of the information system functionalities). Current IBSP reference implementation is written in Java. A wrapper hides an existing DHT API, namely FreePastry [30]. It provides access to the insert/lookup operations and the multicast communication based on the publish/subscribe pattern. FreePastry is an open source project and the main Java DHT solution with Bamboo [31]. Even if it has been shown that Bamboo may be preferable in some cases [32], the differences are not so remarkable. An evaluation of the Java DHT implementations is out of the scope of this paper. The default configuration exposes HTTP-REST interfaces and loads a resources ID system based on a class of private IP addresses. Such configuration is not bounded to any specific platform. Declarative programming allows specifying the plug-ins to be loaded to support specific platforms. Plug-ins are deployed even at run-time through a descriptor Java properties file. It serves to set-up the proper classes fitting a client platform. A set of interfaces are provided to implement the plug-ins. They let to customize the following aspects: 1) the ID system; 2) the grammar for topic publish/subscribe; 3) the message model; 4) the wire protocol; and 5) optional DHT interactions through the wrapper. While we already mentioned the first and the second aspects in the functional description, it is useful to spend a few words about the others. The third and fourth aspects determine the way to invoke the methods exposed by IBSP. More precisely, they define the semantic and the protocol syntax of a supported client platform. The fifth aspect serves to customize the format of the data to be stored and retrieved to and from the DHT. They can be simply represented as values of DHT entries thus requiring some rule to compose and parse the values strings. Alternatively, they can be network pointers to specific remote files containing all the information associated to a

resource. In this case, this aspect allows specifying the way to access to remote sites and the handler to manage the adopted file format. A practical example showing the plug-in of an existing platform is included in the reference implementation. It regards the already mentioned Private Virtual Cluster (PVC) platform. The developed plug-in uses the PVC metric to indentify resources, namely the experimental class E of IP addresses. The messages are text based and exchanged on top of TCP. Apart the fault detection, all the available functionalities are used by PVC. The most remarkable implementation is a connectivity enabler method articulating a loop allowing the discovery of the traversing technique necessary for the direct connection with a resource. That loop involves different PVC entities, and implies a whole exchange of eleven messages. Given the small amount of data required by PVC, the wrapper stores PVC resources information directly as the DHT values instead of network pointers. A registered PVC resource is represented by a couple composed by its virtual class E IP address and an aggregate data including: peer real IP, resource nature, traversing technique required to access to itself, virtual hostname and optional data. Next session shortly provides some details about PVC to introduce the various set of case studies we tested with the IBSP plug-in for PVC.

4 Case Studies

PVC is a P2P middleware targeted to provide a computational cluster abstraction on top of machines connected to LAN or WAN networks through the TCP/IP suite. From the user point of view, the access to these resources is transparent, without changing the security policies of each participating domain. A PVC peer daemon runs on every participating machine. On top of those demons, a distributed application can run unmodified upon the virtual cluster. PVC defines a set of *interposition techniques* to intercept the network communications of the overlying application. Another set of techniques, called *connectivity techniques*, allows establishing direct connections between resources, which can be behind firewalls or NATs. The interposition techniques detects when a virtual resource attempts to establish a direct connection, triggering the suitable connectivity technique by means of the IBSP connectivity enabler functionality. Each interposition technique implies a constant overhead in the peers direct connection which is not under control of IBSP. Since our target is not the analysis of the PVC connectivity techniques impact, we used *Traversing TCP* in all the experiments we ran. The functionalities we measured are: the resource enrollment, the resource lookup and the resource connectivity enabler we describe before. The resource enrollment includes the interaction with the ID system. In the case of PVC, the connectivity enabler is called inter-connection establishment, and the term resource becomes synonym of PVC peer. They are the functionalities mainly used by virtual clusters as shown in table 1, and they produce the largest part of the overhead involved by the IBSP adoption. PVC does not use the presence detection and the fault detection methods, while the use of the information system is limited to a particular case which has not relevant effect on the performances. An implementation of the presence detection would behave similarly to the resource lookup because basically it would leverage the same primitives (as it is implemented in the neutral HTTP-REST interface). We ran three different stages of experiments to test IBSP serving PVC.

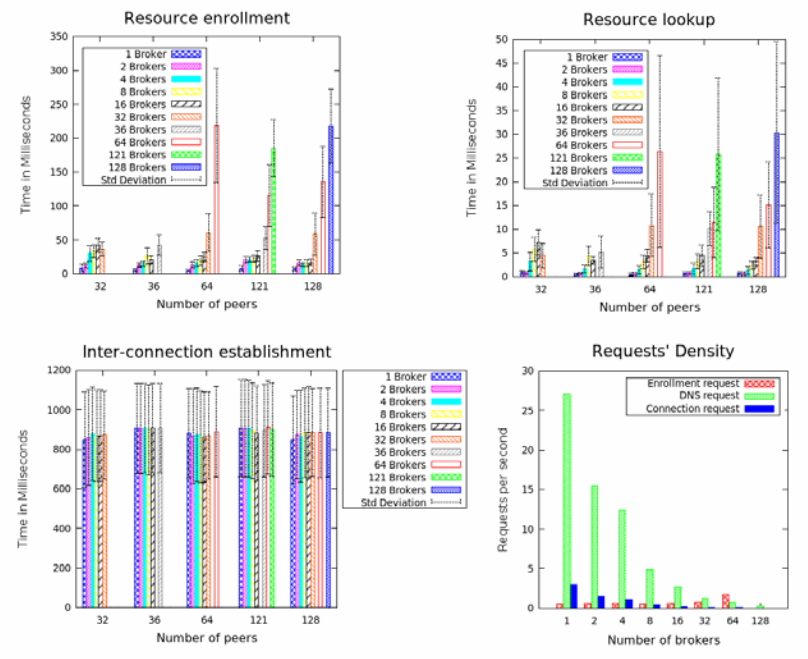

Fig. 2. NPB experiments results

We leveraged the Grid5000 [33] platform for the first two stages. The Grid5000 project offers an experimental test-bed for grid experiments. This platform is made up with computing resources coming from nine sites geographically distributed in France. Each site has its own hardware and network specifications, and clusters are inter-connected through a 10Gb/s link or at least 1 Gb/s, when 10Gb/s is not available. Depending on the size of the performed experiment, resources can belong to only one site, to a maximum of eight different sites. The third experiment deals with a heterogeneous environment we will describe later.

4.1 NAS Benchmarks

For the first group of experiments, we ran the NAS [34] benchmarks on top of PVC. The NAS Parallel Benchmarks (NPB) are used to evaluate the performance of parallel supercomputers. They provide a set of kernel and pseudo-applications to measure different aspects of parallel environments. We used the MPI-based implementation of the NPB. The CG, EP, LU, MG benchmarks were run for the following PVC virtual cluster sizes: 32, 64 and 128 nodes, while the BT and SP benchmarks for 36, 64 and 121 nodes. For a fixed virtual cluster size, each benchmark were run with different sets of IBSP nodes serving PVC: we increased the number of IBSP nodes starting from a single instance managing all the PVC nodes, to the match of the virtual cluster size (i.e. PVC peers) following the power of 2 (2, 4, 8, ... 256). An equitable number of PVC peers were assigned to each IBSP node. Note that the same experiments with a random assignment of PVC peers among the IBSP nodes gave

similar results. In all the NPB experiments, we used the *Library interposition technique* on the PVC peers. This technique overloads certain standard system calls, by means of the Dynamic Library Linking feature. The different tested functionalities showed similar behaviors independently on which NPB it was run. Figure 2 shows the average time and standard deviation of each functionality for every configuration of IBSP nodes and PVC peers. The performance evolution of the resource enrollment and lookup operations for any virtual cluster size, in relation with the number of IBSP nodes, follows a similar trend. The larger overhead of the resource enrollment functionality is explained by the fact that the resource enrollment includes a resource lookup, in order to check if the chosen ID it is not already in use. The address generation itself and the creation of a new entry in the DHT also increase the execution time of this functionality. With only a single IBSP, the overhead is very limited, since these two operations remain local. When the number of IBSP nodes increases, the overhead due to coordination and network communication becomes higher: an IBSP node must interact with the nodes in its neighborhood and wait for their answers to complete the operation. From a cluster size of 64 nodes, the resource enrollment and lookup operations undergo a consistent decrease, even if they remain in a very low scale. This is explained by the influence of the Grid5000 inter-site communication arising when resources belong to more than one single site. It also explains a higher standard deviation. Finally, when the number of IBSP nodes is equivalent to the number of PVC peers, the information reaches its maximum distribution over the DHT. Experiments with more IBSP nodes than PVC peers would not provide any additional information, since the worst test case comprises each IBSP node managing a single PVC peer as maximum. These results might appear in contrast with the intuitive deduction that

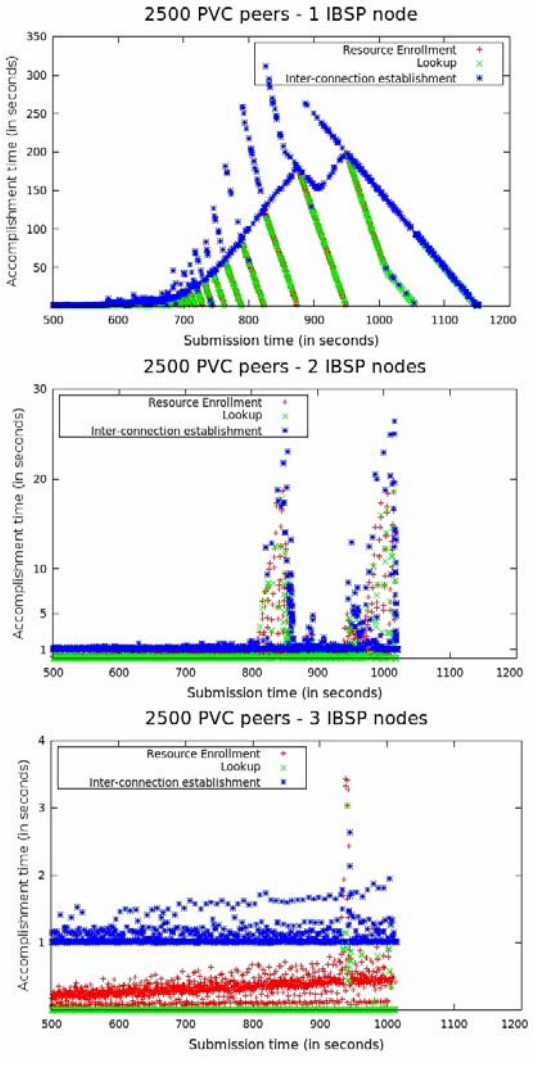

Fig. 3. Scalability experiments results

more brokers should be faster than a single one, because they are supposed to perform tasks in parallel. It may be true over a certain threshold causing a performance decrease because of the concurrent management of multiple peers. However, we did not reach this scalability threshold, even if the scale of real collected peers was not negligible. It means that the overhead due the connection broker-to-peer is lower than the overhead due to communication broker-to-broker. We did not get over that scale because of practical burdens hampering to set-up larger experiments on Grid5000. It is possible to say that a single instance of IBSP is enough up to manage 128 PVC peers. More instances can be used for any reason (e.g. administrative policies, reliability, high availability, etc.) with the shown overhead. In the inter-connection establishment graphic, the measures show a constant overhead in all the cases. Considering the scale, the behavior reflects the influence of the other measures plus a constant due to the PVC peers interposition and connectivity techniques, which are not under control of IBSP and take an active role during this process. Obviously, the overhead due to the Grid5000 inter-cluster communication has also an influence. Figure 2 shows also the requests density associated to a virtual cluster of 128 peers, the largest in the NPB experiments. The stress associated to the resources enrollment remains constant, since PVC peers were registered sequentially at the same rate, in all cases. For the other requests, we can easily notice a decrease in the stress the IBSP nodes are subjected to, when their number increases, since the workload is shared among them. It could suggest another reason to adopt multiple IBSP instances in order to reduce the load on the machines hosting IBSP.

Regarding the adoption of multiple IBSP instances and the mentioned scalability issue, we ran a further experiment with a simulated configuration. We implemented a simulated PVC peer which calls sequentially the enrollment, the lookup and the Inter-connection establishment operations disregarding the NAS benchmarks. For each run, five Grid5000 machines were used to launch simultaneously 500 instances of simulated PVC peer per machine, at a rate of one instance every two seconds. Each IBSP node ran in a different machine. Figure 3 reports the interesting results we obtained. The graphics show for each run, the submission and accomplishment time of each operation request. It looks evident the performance decrease when a single broker has to manage the whole set of 2,500 simulated peers. The overlap of functionalities requests, the overhead associated to the ID generation process, the simultaneous access to the DHT, and the stress in the network management are added thus provoking

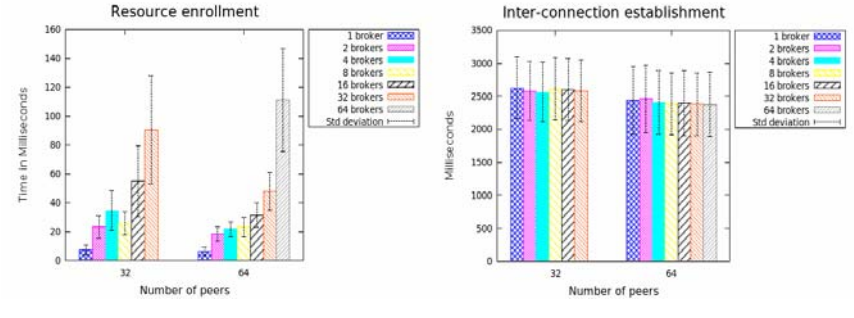

Fig. 4. CONDOR/BLAST experiment results

	Total Execution Time				
	cg	ep	lu	mg	condor
1 peer	64,6 s	3,9 s	99,4 s	6,9 s	7 s
2 peers	67,4 s	3,2 s	98,9 s	7,1 s	7 s
4 peers	32,1 s	3,9 s	97,8 s	4,9 s	6,9 s
8 peers	68,3 s	4,4 s	97,4 s	6,9 s	7 s
16 peers	70,6 s	4,4 s	95,3 s	6,4 s	6,7 s
32 peers	66,8 s	4,1 s	99,7 s	6,8 s	6,6 s
64 peers	32,2 s	3,9 s	81,6 s	3,3 s	6,9 s
128 peers	24,5 s	5,4 s	68 s	4,5 s	-

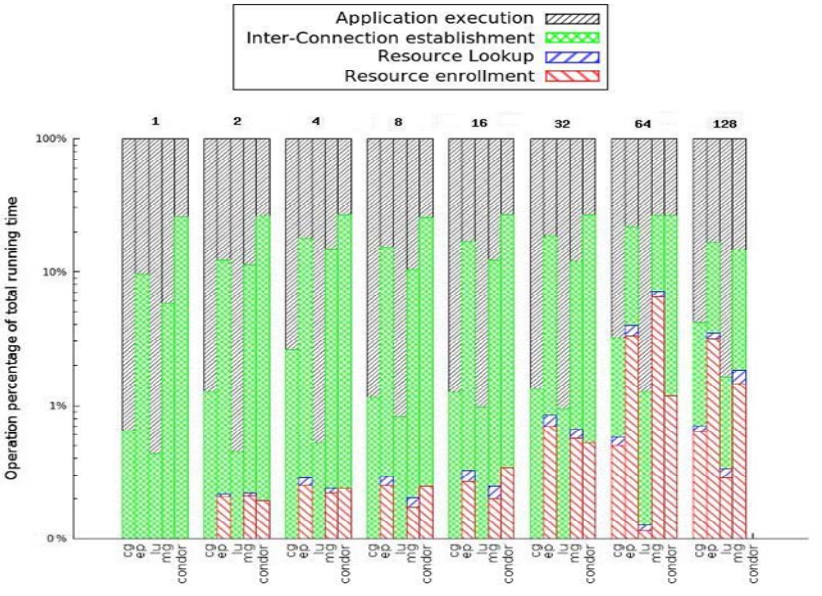

Fig. 5. IBSP/PVC Impact

a noticeable amount of peaks and a general increase of the measured accomplishment times of each functionality. As shown, the adoption of one instance more reduces greatly the general increase. Some peaks still remain when the structure is stressed by the accumulation of many concurrent requests. Finally, the third IBSP instance reduces significantly even the number of peaks arising in the most stressful moments. Obviously, these results are less meaningful than the ones obtained in a real execution environment running consolidated benchmarks. However, they provide a reasonable cue advising the use of multiple instances of IBSP for pure performances reasons, as it seems when the size of PVC virtual cluster is one order of magnitude larger than the largest we reached in the real environment.

4.2 Condor/BLAST

We set up a different PVC configuration to test the IBSP functionalities in another applicative scenario. We configured PVC peers to use the *Kernel Module interposition technique* instead of the Library interposition, used to run the NPB benchmarks.

Even if the communication protocol does not change, the different approach to manage the network communications brings PVC to behave like a different application compared to the previous case. We ran the job scheduler Condor on top of PVC. The jobs submitted to Condor belong to BLAST (Basic Local Alignment Search Tool) application [35]. This program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. In the tests, different queries are performed against the Genome Survey Sequence (GSS) database, being managed each one of them as an independent job. Condor schedules these jobs, and assigns them to the suitable nodes. Condor has been set up without any special configuration and with a single resource pool, where the participating nodes are registered. We ran this test on virtual clusters composed respectively by 32 and 64 PVC nodes. The configuration of the PVC virtual cluster size follows the same pattern we used to run the NAS benchmarks. The number of BLAST queries submitted for each test is equal to the number of participating nodes.

Figure 4 reports the graphs of the average time obtained for the enrollment and the inter-connection establishment. The measures show trends similar to the previously seen ones. The different absolute values are a reflection of the different interposition technique used by PVC. Figure 5 provides a single view of the time percentage related to each IBSP operation compared to the whole execution time of each application we ran. We used the data coming from the experiments with the largest virtual cluster sizes, 128 nodes for NPB benchmarks and 64 nodes for Condor/BLAST. In the lowest total execution time, the IBSP overhead does not exceed the 20%. The Condor/Blast test over PVC configured with the Kernel Module interposition technique has the largest impact.

4.3 Heterogeneous Environment

The aim of the third type of test is to prove the capability of IBSP to manage PVC peers belonging to different domains. We collected PVC peers running over resources coming from: Amazon EC2 (5 PVC peers and 1 IBSP node), DSLlab [36] (1 PVC peer and 1 IBSP node), an experimental platform providing a set of nodes spread over

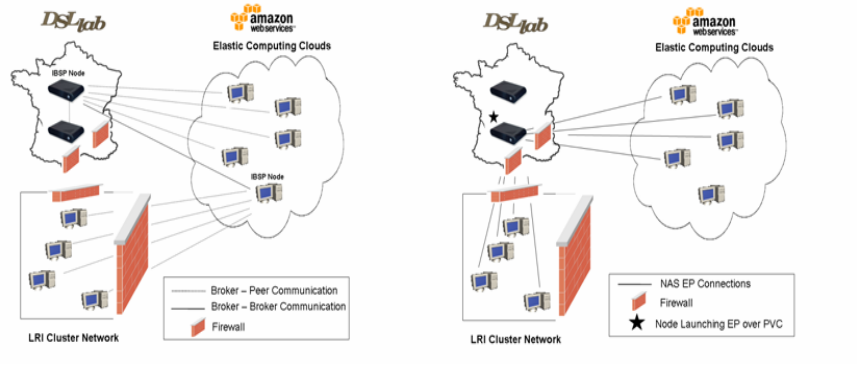

Fig. 6. Heterogeneous environment configuration

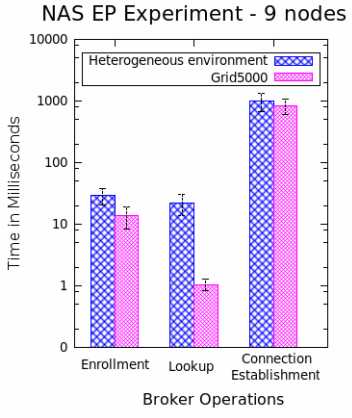

Fig. 7. Heterogeneous experiment results

the ADSL, and a private network (4 PVC peers) belonging to the Laboratoire de Recherche en Informatique (LRI), located at Orsay, France. We ran the EP NAS benchmark on top of PVC, using all the available resources (i.e. 9 PVC peers), with the same configuration used in the experiments shown in section 4.1.

Figure 6 shows the IBSP-PVC topology, and the connections established among PVC peers when the NAS EP benchmark is launched. Special attention is needed when the PVC-IBSP resources belong to different administration domains. PVC peers belonging to the same domain cannot connect to an IBSP node which is also running in the same domain: the IBSP node would register their private real IPs instead of the public real IPs associated to their administration domain. This would cause a conflict when an external PVC peer attempts to establish a connection because the IBSP node would provide the private addresses of the requested peers. This problem is solved by enrolling PVC peers to IBSP nodes running in different domains. This deployment operates on a real user environment instead of an isolated one as Grid5000. In fact, this experiment is able to collect PVC peers belonging from commercial providers to single Internet-connected machines (DSLlab), and goes a step further from isolated test-bed environments. We ran the same experiment on the Grid5000 platform to be able to compare the results coming from this heterogeneous environment. Figure 7 shows the measures we obtained. The resource enrollment and resource lookup operations present a longer execution time than their equivalents on Grid5000 because of the higher network latency due to the communication through the public Internet. There is not a substantial change in the Inter-connection procedure. In fact, the lightweight IBSP-PVC communication protocol is not affected by the lower bandwidth of the public Internet.

5 Conclusion and Future Works

In this paper, we have presented a distributed software architecture targeted to provide a brokering service seized to various needs of P2P-based virtual cluster platforms. Our work takes its origin from a classification of brokering needs arising in different platforms. The drawn categorization is the base of the Information Brokering Service Provider (IBSP), a SOA-compliant Resource Broker for virtual clusters. It is built on top of a flexible architecture which allows to be plugged-in in order to serve different platforms.

We tested the early IBSP reference implementation serving the Private Virtual Cluster (PVC) platform. As shown in section 4, the overhead added by IBSP to the considered applications and benchmarks is acceptable when compared to the total

execution time. We performed our tests on an isolated test-bed as Grid5000 as well as on a real scenario involving resources distributed over Internet. The main contribution of the experiments on Grid5000 is a precise evaluation of IBSP capabilities without any disturbs possibly arising in real environments. The experiments showed that a single IBSP instance could be enough for a large number of PVC peers. However, as shown in the last experiment, there are real cases which require multiple instances because of practical burdens not permitting the optimal deployment. In fact, IBSP is built with a flexible architecture which allows administrators to tune its topology. Our evaluation quantifies the overhead involved by this possibility, and provides precise cues on the performance impact which can be brought by a particular topology choice. The encouraging results we obtained are the starting point for definition of new plug-ins in order to promote the IBSP adoption for the deployment of P2P-based virtual clusters. As immediate future works, we planned the development of the IBSP support for the emerging HIPCAL [37] cluster virtualization platform.

Acknowledgments. Experiments presented in this paper were carried out using the Grid'5000 experimental test-bed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

1. Foster, I., Iamnitchi, A.: On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735. Springer, Heidelberg (2003)
2. Ruth, P., McGachey, P., Xu, D.: VioCluster: Virtualization for Dynamic Computational Domains. In: Proceedings of the IEEE International Conference on Cluster Computing, Cluster 2005, Boston, MA, USA, September 26-30 (2005)
3. Ganguly, A., Wolinksky, D.I., Boykin, P.O., Figueiredo, R.: Improving Peer Connectivity in Wide-area Overlays of Virtual Workstations. In: Proceedings of the 7th International Symposium on High Performance Distributed Computing, HPDC 2008, Boston, MA, USA, June 23-27 (2008)
4. Rezmerita, A., Morlier, T., Néri, V., Cappello, F.: Private Virtual Cluster: Infrastructure and Protocol for Instant Grids. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 393–404. Springer, Heidelberg (2006)
5. Cluster On Demand (COD), `http://www.cs.duke.edu/nicl/cod/`
6. Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., Tsugawa, M.: Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In: Proceedings of the International Conference on Cloud Computing and its Application, CCA 2008, Chicago, IL, USA, October 22-23 (2008)
7. Amazon Elastic Compute Cloud (EC2), `http://aws.amazon.com/ec2/`
8. Nimbus, `http://workspace.globus.org/`
9. Eucalyptus, `http://eucalyptus.cs.ucsb.edu/`
10. Krafzig, D., Banke, K., Slam, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice-Hall, Englewood Cliffs (2004)

11. Jiang, X., Xu, D.: VIOLIN: Virtual internetworking on overlay infrastructure. In: Cao, J., Yang, L.T., Guo, M., Lau, F. (eds.) ISPA 2004. LNCS, vol. 3358, pp. 937–946. Springer, Heidelberg (2004)
12. XEN, http://www.xen.org/
13. Irwin, D., Chase, J., Grit, L., Yumerefendi, A., Becker, D., Yocum, K.: Sharing Networked Resources with Brokered Leases. In: Proceedings of the annual USENIX Technical Conference, Boston, MA, USA, 30 May – 3 June (2006)
14. Sundararaj, A., Gupta, A., Dinda, P.: Dynamic Topology Adaptation of Virtual Networks of Virtual Machines. In: Proceedings of the Seventh Workshop on Languages, LCR 2004, Houston, TX, USA, October 22-23 (2004)
15. Tsugawa, M., Fortes, J.: A Virtual Network (ViNe) Architecture for Grid Computing. In: Proceedings of the 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, Rodhes Island, Greece, April 25-29 (2006)
16. UDDI Specification, http://www.uddi.org/pubs/uddi_v3.htm
17. Sun, C., Lin, Y., Kemme, B.: Comparison of UDDI registry replication strategies. In: Proceedings of the International Conference on Web Services, San Diego, CA, USA, July 6-9 (2004)
18. UBR Shutdown FAQ, http://uddi.microsoft.com/about/FAQshutdown.htm
19. Banerjee, S., Basu, S., Garg, S., Lee, S.J., Mullan, P., Sharma, P.: Scalable Grid Service Discovery Based on UDDI. In: Proceedings of the 3rd International Workshop on Middleware for Grid Computing, Grenoble, France, November 28-29 (2005)
20. Pallickara, S., Fox, G.: NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672. Springer, Heidelberg (2003)
21. JXTA, http://www.sun.com/software/jxta/
22. H2O, http://dcl.mathcs.emory.edu/h2o/
23. Wun, A., Petrovi, M., Jacobsen, H.: A System for Semantic Data Fusion in Sensor Networks. In: Proceedings of the inaugural International Conference on Distributed Event-Based Systems, DEBS 2007, Toronto, Canada, June 20-22, 2007, pp. 20–22 (2007)
24. JAIN SLEE, http://jainslee.org
25. Freeman, T., Keahey, K.: Flying low: Simple leases with workspace pilot. In: Luque, E., Margalef, T., Benítez, D. (eds.) Euro-Par 2008. LNCS, vol. 5168, pp. 499–509. Springer, Heidelberg (2008)
26. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus Open-source Cloud-computing System. In: Proceeding of the International Conference on Cloud Computing and its Application, CCA 2008, Chicago, IL, USA, October 22-23 (2008)
27. Enomaly, http://www.enomaly.com/
28. Dabek, F., Brunskill, E., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I., Balakrishnan, H.: Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. In: Proceedings of the 8th Workshop on Hot Topics in Operating Systems, HotOS-VIII, Elmau/Oberbayern, Germany, May 20-23 (2001)
29. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, p. 329. Springer, Heidelberg (2001)
30. FreePastry DHT, http://freepastry.org

31. Bamboo DHT, http://bamboo-dht.org/
32. Kato, D., Kamiya, T.: Evaluating DHT Implementations in Complex Environments by Network Emulator. In: Proceedings of the 6th International Workshop on Peer To Peer Systems, IPTPS 2007, Bellewue, WA, USA, February 26-27 (2007)
33. Cappello, F., et al.: Grid'5000: a large scale, reconfigurable, controllable and monitorable Grid platform. In: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, GRID 2005, Seattle, WA, USA, November 13-14 (2005)
34. NPB, http://www.nas.nasa.gov/Resources/Software/npb.html
35. BLAST, http://blast.ncbi.nlm.nih.gov/Blast.cgi
36. DSLlab, https://dsllab.lri.fr:4322
37. HIPCAL, http://hipcal.lri.fr/