

Analysis of Peer-to-Peer Protocols Performance for Establishing a Decentralized Desktop Grid Middleware

Heithem Abbes^{1,2} and Jean-Christophe Dubacq¹

¹ LIPN — UMR CNRS 7030 — Institut Galilée — Université Paris-Nord
99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, FRANCE

Tel: +33(0)1 49 40 35 78 Fax: +33(0)1 48 26 07 12

² École Supérieure des Sciences et Techniques de Tunis, Unité de recherche UTIC
5, Av. Taha Hussein, B.P. 56, Bab Mnara, Tunis, TUNISIA

Tel: (+216) 71 496 066 Fax: (+216) 71 391 166

{heithem.abbes, jean-christophe}@lipn.univ-paris13.fr

Abstract. The Desktop Grid technology consists mainly in exploiting personal computer, geographically dispersed, to deliver massive compute power to investigate complex and demanding problems in a variety of different scientific fields. However, as resources number increases, the need for scalability and decentralization becomes more and more essential. Since such properties are exhibited by Peer-to-Peer systems, we aim at using them to create a decentralized desktop grid middleware. Nevertheless, in order to judge the efficiency of such P2P library, an experimental performance evaluation of the provided functionalities is unavoidable. Very few analysis of this kind have been reported, as most evaluations are limited to complexity analysis and to simulations. Such experimental analysis are important, especially when using P2P tools in grid computing context, when applications may have precise efficiency requirement. In this paper, we focus on three libraries: Bonjour, Avahi and Pastry, which provide generic API intended to serve as basis for specialized P2P applications. We perform a performance evaluation of the scalability and their capacity to register and browse an important number of services over 300 hosts in Grid'5000 for recent versions of Pastry, Avahi and Bonjour. We provide detailed analysis explaining the behavior of each library related to two criteria: the elapsed time for registration services and the needed time to discover services. Our aim is to choose the most adequate protocol for creating a decentralized middleware for desktop grid.

1 Introduction

Grid Computing aim at providing a powerful infrastructure based on the aggregation of large numbers of resources spanning multiple organizations. This concept is rapidly emerging as the dominant paradigm for distributed problem solving for a wide range of application domains. The middleware of grid computing is, however, so complex that it needs a lot of effort to maintain. Therefore

it is natural, that single persons do not offer their resources but all resources are maintained by institutions, where professional system administrators take care of the environment and ensure the availability of the Grid. Examples of such Grid infrastructures are the TeraGrid [1], EGEE [2] and Open Science Grid (OSG) [3]. Complementary to grid computing, Desktop Grids leverage Internet connected computers to support large computations. In this kind of system, a single person can more easily add his personal computer (PC) to the grid. Desktop grid have been successfully used to address large applications with significant computational requirements, including global climate prediction (Climatprediction.net), protein structure prediction (Predictor@Home), search for extraterrestrial intelligence (SETI@Home), gravitational wave detection (Einstein@Home), and cosmic rays study (XtremWeb). While the successes of the above applications do demonstrate the potential of desktop grid, existing systems are often centralized and suffer from being not able to scale due to centralized control. To bypass this, we would profit from existing decentralized P2P systems in order to organize the management of the desktop grid. Nonetheless, we face the following issue: (i) which kind of systems to choose? (ii) more exactly, what is the maximum number of PC that can be managed by a system? (iii) If one machine join the desktop grid, how many time needed to register this machine? By evaluating three "famous" P2P systems with real experimentations on the testbed GRID'5000[4], we provide quantitative answers to these questions.

Moreover, in this work, we assume that we have a high level middleware able to virtualize the network (we have no more problems with firewall and NAT) and we are able to run Bonjour, Avahi and Pastry on top of such middleware. Instant Grid / Private Virtual Cluster [5, 6] is one of the candidate for network virtualization. Its main requirements are: 1) simple network configuration 2) no degradation of resource security 3) no need to re-implement existing distributed applications. Under these assumptions, it is reasonable to check if Bonjour [7, 8], Avahi [9] and Pastry [10, 11] can scale up. The core idea behind these protocols is to build self-organized overlay networks when nodes join a virtual organization.

The remainder of this paper is organized as follows. Section 2 motivates the choice of the three systems Bonjour, Avahi and Pastry and presents an overview of them. Section 3 describes our experimental setup to evaluate the performance of the three systems. Section 4 and section 5 provide results from our experiments in Grid'5000. Section 6 discusses related works. Section 7 summarizes our contributions and gives future works.

2 ZeroConf vs DHT

At the moment, the trend would be to use DHT, because of the scalability property. In fact, the originality of our work is to evaluate DHT versus publish/subscribe system. We use the Pastry library to implement a DHT concept and the two implementations of ZeroConf protocol for publish/subscribe systems. The choice of Avahi and Bonjour is justified by the fact that they are two implementations of the ZeroConf protocol (Zero Configuration Networking)

which already proved its reliability in the field of the local area networks and which can be extensible on wide area networks (by using the Unicast sending protocol coupled with the DNS protocol). Among the protocols based on a DHT (Distributed Hash Table) such as CAN [12] and CHORD [13], we chose Pastry because, on the one hand, it offers the possibilities of replications and, on the other hand, because there exists an open-source implementation [11].

2.1 Bonjour

Bonjour is an implementation by Apple of the ZeroConf protocol. The goal is to obtain a functional IP network without dependency of an infrastructure comprising DHCP or DNS servers. Bonjour is structured around three functionalities: it allows the dynamic allocation of IP addresses without DHCP, it ensures the resolution of names and IP addresses without DNS and carries out the research of the services without directory server. At a technical level, Bonjour uses Link-Local addresses. When DHCP fails or is not available, link-local addressing lets a computer make up an IP address (of IPv4 type) for itself. In IPv4, link-local address is selected by means of a pseudo-random generator in a defined range of addresses (169.254.1.0-255). The checking of address uniqueness is done using three requests which are diffused on the link-local. If IP address is already used (or requested) by another machine, then the machine tries another address provided by the generator. When the machine finds a free address, it diffuses in broadcast two ARP advertisements with the source IP address containing the selected one. In fact, if at any time the machine obtains an address by DHCP then it uses this address and leaves the process of self-configuration on the link-local. Like link-local addressing, when DNS servers are unavailable or unreachable, the machines can still refer the ones with the others by name by using the protocol mDNS (multicast DNS). Bonjour uses DNS-SD protocol (DNS Service Discovery) to discover the services published in a local area network. Since DNS-SD is built on top of DNS, it works not only with mDNS but also with traditional DNS for discovering remote services.

2.2 Avahi

Avahi is a system which facilitates service discovery on a local network. It allows the programs to publish and discover services and hosts running on local network without any specific configuration. Avahi is an implementation of DNS Service Discovery and mDNS specifications for ZeroConf protocol. Avahi is mainly based on mDNS implementation for Linux. It uses D-Bus (an asynchronous library for communication between processes) for communication between user applications and system.

2.3 Pastry

In this section we briefly sketch Pastry [10]. A Pastry system is a self-organizing overlay network of nodes, where each node routes client requests and interacts

with local instances of one or more applications. Any computer that is connected to the Internet and runs the Pastry node software can act as a Pastry node, subject only to application-specific security policies. Each node in the Pastry P2P overlay network is assigned a 128-bit node identifier (nodeId). The nodeId is used to indicate a node's position in a circular nodeId space, which ranges from 0 to $2^{128} - 1$. The nodeId is assigned randomly when a node joins the system. It is assumed that nodeIds are generated such that the resulting set of nodeIds is uniformly distributed in the 128-bit nodeId space. For instance, nodeIds could be generated by computing a cryptographic hash of the node's IP address. As a result of this random assignment of nodeIds, with high probability, nodes with adjacent nodeIds are diverse in geography. Assuming a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $\log_2^b N$ steps under normal operation (b is a configuration parameter with typical value 4). Despite concurrent node failures, eventual delivery is guaranteed unless nodes with adjacent nodeIds fail simultaneously (c is a configuration parameter with a typical value of 16 or 32).

For the purpose of routing, nodeIds and keys are thought of as a sequence of digits with base $2b$. Pastry routes messages to the node whose nodeId is numerically closest to the given key. This is accomplished as follows. In each routing step, a node normally forwards the message to a node whose nodeId shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the present node's id. If no such node is known, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the present node's id.

Free-Pastry is an open-source implementation of Pastry intended for deployment in the internet. We have used in our experimental tests the release 2.0.

3 Description of the experimental setup

The experimental platform is Grid'5000, a highly reconfigurable and controllable grid system, which gathers 9 sites geographically distributed in France. All sites are connected by RENATER network (10 Gb/s). Our tests are applied only in Orsay site, where nodes are connected by a network of 1 Gb/s. We almost used the totality of the available machines in this site (more than 300 machines). All machines have AMD Opterons processors and networks cards of 1 Gb/s.

We represent the nodes by services to build a virtual network on Grid'5000 platform. Indeed, on each machine we register a service, thus, if the service is running then the machine is connected on the network, if not (we deactivate or we remove the service) the machine is disconnected.

Our objective is to study the scalability and the response time of the tools described above. In fact, we look for the maximum number of nodes which can be supported by these tools and the response time necessary to discover a new node which has been just connected on the network (depending on the grid state). The same benchmarks criteria are applied for the three systems.

3.1 Specific environment on Grid'5000

Grid'5000 offers an infrastructure with standard images. To run our experimental tests, we personalized an image to support Avahi, Bonjour and Free-Pastry. Thus, we created a specific Linux kernel containing the necessary packages to execute our codes. Thereafter, by using OAR [14] and Kadeploy [15] tools, we reserve and we deploy the specific image on all reserved machines according to the traditional procedure for the Grid'5000 users.

3.2 Sequential registrations

In this test, the first step is to reserve N nodes on Grid'5000. The number N represents the maximum nodes that can be used for the experiment. Each node requests a registration for a given service at a given time. Initially, all nodes have the needed codes to request a service but are inactive. Let δ be the activation time. We activate sequentially all the requests (and we receive back an acknowledgement). Indeed, the k^{th} request will be activated at time $k \times \delta$. We increase δ to analyze the behavior of the system when the delay between events becomes larger. Obviously, at the beginning of the test, the registration number is small, thus the registration time will be fast. We increase N until the saturation value (i.e. the registration service no longer responds for a new registration). We aim at analyzing the scalability of the system without overloading the network: in this test, only one multicast appears at a given time.

3.3 Simultaneous registrations

In the first test, the registrations are done sequentially. This leads to a limited number of communications to exchange information. In this experiment, we stress the scalability of the system and its capacity to manage communications between registered nodes. Therefore, we request N (the number of reserved nodes) simultaneous registrations and we compute the time to complete the registration step. If we obtain a "reasonable" response time, we increase N until the saturation value. In others words, we are looking for the maximum registered nodes that the system handles when the network is overloaded by several multicast packets at the same time.

3.4 Browsing services

The other important metric is the time needed to browse a given service. Indeed, in previous tests, we compute the registration time. We need also to compute the discovering time which is the elapsed time between the registration end of a unique service and the date at which a browser node has discovered this service. The browsing program listens any new event, i.e. a new registration or deleting services. With the two setup mentioned before, we can analyze the performance of service discovery.

4 Performance of registration services

4.1 Registration of Bonjour services

Bonjour starts by making a DHCP request to obtain an address. In case of absence of DHCP, it performs one (up to 3) ARP request(s); however when the kernel is booted, an IP is already attributed, so we can not take into account the elapsed time in this first phase. Bonjour should notify all machines that the node has the service by updating all ARP caches. There could be an additional management cost related to replacements in the cache but the default size of ARP cache is 1040 entries which is higher than the number of services used in the worst case of our experiments. Consequently, we can not reasonably charge the management of ARP caches in the degradation of performances. Bonjour proceeds by checking the uniqueness of service. An initial ARP advertisement is made with one second waiting (Probe wait = 1s). If there has been not reply in the second (it is well the case, because all the services were selected with a single name), then the name is considered as unique.

In figure 1, the y -axis shows the percentage of services correctly registered at time x . The sequential registration shows a better times than simultaneous one. Indeed, the measurements taken on 308 machines give registrations times ranging between 1015 and 1030 ms. Whereas, in the simultaneous version, elapsed time varies between 1017 ms and 2307 ms. Comparing with the sequential case, we find again the same constant (1015 ms) for simultaneous registration in the first registration, but it becomes longer to reach the bus in turn and that costs about 10 to 30 ms, which generates an increase cumulated in the registration time from 1017 ms to 2307 ms.

4.2 Registration of Avahi services

On each node of the Avahi network, there is a running avahi-daemon. This daemon implements the two protocols mDNS/DNS-SD of Zeroconf. To register one service, Avahi publishes the service by a multicast send to all nodes, using D-Bus as transport protocol, and updates the cache of each node. These protocols showed a great effectiveness in services registration. Indeed, figure 1 shows that Avahi gives almost same elapsed times in sequential and in simultaneous tests. Elapsed time varies between 760 and 1110 ms which are better than ones given by Bonjour.

4.3 Registration of Pastry services

Contrarily to Avahi and Bonjour, Pastry shows a great difference between sequential tests and simultaneous ones. Indeed, figure 2 shows that in simultaneous registration, until 160th service, elapsed time varies between 600 and 1000 ms. Beyond that, registration time increases from one registration to another to reach 320 000 ms. In addition, figure 2 shows that the sequential registration gives better times. We mention that Pastry gives reduced times in comparison

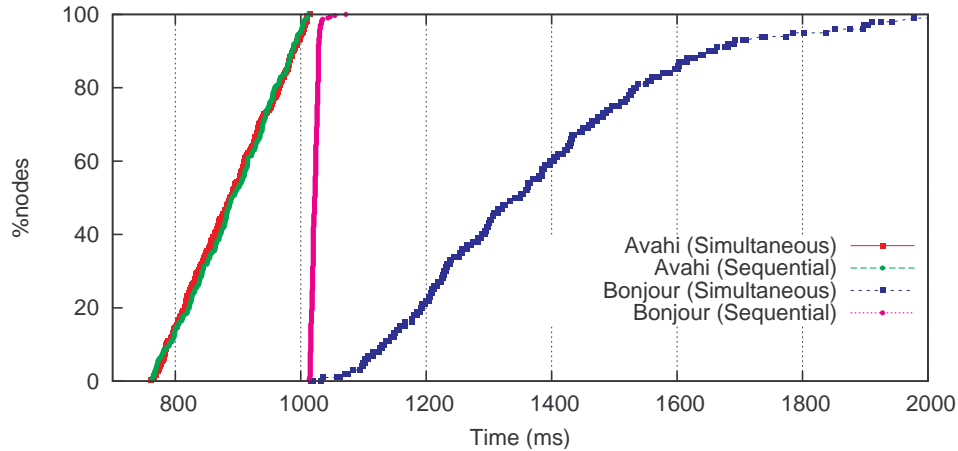


Fig. 1. Elapsed time for registrations of Bonjour and Avahi services

to Avahi and Bonjour (30% of services, among 307, are registered in the interval 450-550 ms). Indeed, when we register a service, we require connection to the same bootstrap machine (to avoid creating several rings). Thus, in simultaneous case, the bootstrap can not answer all simultaneous requests, what causes a fast growth of times since 160th service. Moreover, Pastry updates the leaf sets to maintain the coherence of the system, and may be for this reason the simultaneous version times reach 320 s. Whereas in the sequential version, the bootstrap receives only one request each minute; the update of the leaf sets and routing tables is, thus, recovered so that we got reduced times (2 s maximum). Our measurements show that it is necessary to create a second bootstrap to initialize a new ring when the number of simultaneous registrations overtake 160 services.

4.4 Synthesis

The comparison of the three libraries (Bonjour, Avahi and Pastry) from the viewpoint of simultaneous registration times of approximately 300 services on the platform Grid'5000 (one service by machine) shows that Avahi is the best since it spends less time (last registered service needs 1000 ms). Bonjour requires 1300 ms moreover to register the last service. Pastry gives times close to those given by Avahi until 160 registrations, beyond this, it spends times definitely larger (until 320 000 ms) that those of Avahi and Bonjour. When we sequentially register one service on each machine (we register about 300 services), we can mention that there is not a great difference between the three libraries. In fact, Bonjour and Avahi give similar results. Pastry spends almost same time to register 60% of services, needs less times to register first 30% but more times than Avahi and Bonjour for the rest (10%).

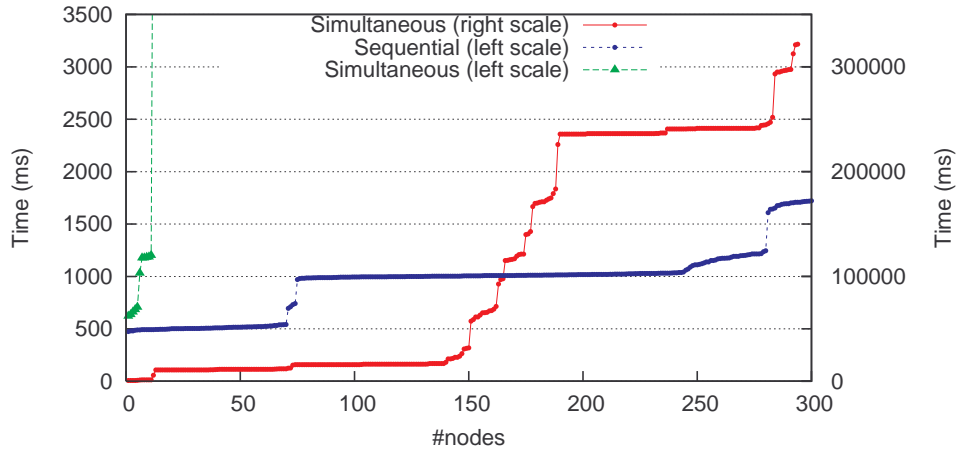


Fig. 2. Elapsed time for registrations of Pastry services

5 Performance of discovery services

The second metric is to measure the necessary time to browse a registered service. Then for each system (Bonjour, Avahi and Pastry) we measure the elapsed time between the registration end and the discovery time. We repeat the same benchmarks for both simultaneous and sequential registration. For that, we dedicate one machine which runs the browser to discover services.

5.1 Discovery behavior of Bonjour

Bonjour proves a good performance in discovering services. In fact, it is able to discover 307 services registered on 307 machines (one service on one machine). Furthermore, the discovery time does not exceed 1 second. That leads us to affirm that the implementation by Apple of DNS-SD functions is good and gives satisfactory results.

5.2 Discovery behavior of Avahi

As we have already mentioned, Avahi uses avahi-daemon and executes a request via D-Bus to publish a service by a multicast package. The machine which launches the discovery program (a browser program which remains listening to services) loses 60% of simultaneously registered services. Moreover, the discovery time increases beyond 49 registrations to reach 900 s in the registration of the 73th service. Beyond that, the discovery program spends around 220 s to discover a registered service (see figure 3). Contrarily to the simultaneous registration, when we register the services in a sequential manner (for instance each minute),

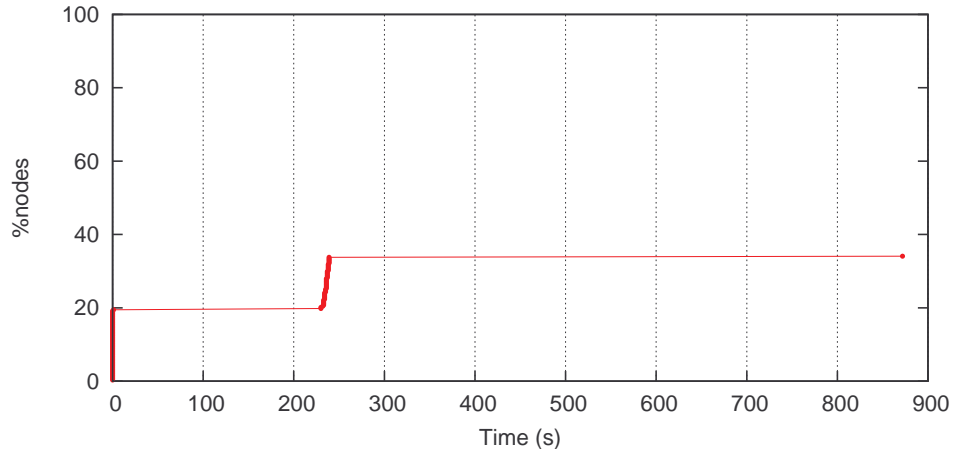


Fig. 3. Elapsed time for browsing simultaneously registered Avahi services

the Browser is able to discover more services (303 among 307 registered services). In addition, the discovery time is better (maximum 4 s) for 204 services except that 9 services need intervals of time of 2835-8686 s. The browser, which has the avahi-daemon running on it, is unable to receive, at a given moment, all multi-cast sends emitted by the nodes which registered the services at the same time (simultaneous case). This can explain the loss of services in the simultaneous version. Whereas, in sequential version the browser needs more time to discover the four lost services.

5.3 Discovery behavior of Pastry

In both types of sequential and simultaneous registrations, pastry provides good discovery times (1 s maximum). Meanwhile, the number of discovered services for simultaneous registrations, the browser discovers 270 among 293 services, whereas for sequential registration, the browser discovers 275 of the 292 services what corresponds to a light improvement in comparison to the previous case. That can leads us to affirm that the Browser fails to recover all services publications notifications when registrations exceed 270.

6 Related work

Experiments and analysis of P2P networks have been conducted over the Grid'5000 platform for the generic JXTA P2P framework [16]. In this article, the goal of the performed benchmarks is similar to our goal. It concerns to answer common and unanswered questions: how many rendezvous peers are supported by JXTA

in a given group and what is the expected time to discover resources in such groups?

Two main protocols of JXTA have been evaluated in [16]: 1) the peer-view protocol used to organize super peers, known as rendezvous peers, in a JXTA overlay and 2) the discovery protocol, that relies on the peer-view protocol, used to find resources inside a JXTA network. All sites of Grid'5000 were used and a mix of hundreds of rendezvous and normal peers, called edge peers, have been deployed on at most 580 nodes. Results show that with default values for parameters of the peer-view protocol, the goal of the algorithm is not achieved, even with just 45 rendezvous peers. However, parameter tuning makes it possible to reach larger configurations in terms of number of rendezvous peers. For the discovery protocol, authors show that discovery time is rather smaller, provided that all rendezvous peers satisfy a given property. These results give developers a better view of the scalability of JXTA protocols.

Our results³ augmented with those of [16] clearly demonstrate that for open source projects as well as for industrial software with production quality, there is a strong need to test and evaluate the properties of the distributed system in real large platform such as Grid'5000.

7 Conclusion and future work

With the growth of grids size, it is feasible to use P2P systems known for their scalability and the management of high volatility. In this context, we studied in this paper three protocols for resource discovery which are Bonjour, Avahi and Pastry. The three protocols showed high performances except that Avahi failed to discover all services in simultaneous version and Pastry spends a considerably long time to register all services simultaneously. We are going to continue working on the three protocols. Indeed, Bonjour is very powerful in registration and discovery in both sequential and simultaneous versions. Pastry also proved its performance in sequential registration and also in simultaneous case provided when we does not exceed 160 registrations at a given moment, what does not represent really a weakness point if we do not work with, let us say 10 million nodes. Avahi has the advantage of being free with accessible source code, which makes its study much easier than Bonjour. As technical point of view, the ZeroConf API does not offer full functionalities to build grid middleware, whereas Pastry offers an open source API developed with Java containing the preliminary functions necessary to the development of this middleware. Our final objective is to build a Desktop Grid middleware based on one of these protocols.

As mentioned before, the initial question was: "how to decentralize the services offered by Desktop Grids?". Our idea is to capitalize on existing systems (for instance on XtremWeb [17]) rather than inventing a completely new one. We are currently implementing a prototype according to the following vision: a user requests for a computation. He provides a tasks graph and codes implementing

³ For more results see the LIPN internal report on this URL: <https://hal.ccsd.cnrs.fr/docs/00/15/93/88/PDF/acdj.pdf>

his distributed application. The idea is to deploy locally a master node (according to the XtremWeb terminology) and to request for participants. Negotiations to select them should now take place. The Publish/Subscribe infrastructure is used to solve the problem. For instance, each node multicasts, periodically, its state (idle, slave, master) and also information about its local load or its use cost, in order to provide metrics for choosing the participants. Under these assumptions, the master node can select a subset of slave nodes according to a strategy that could balance the “power” of the node and the “price” to use it. When a master node finishes, it becomes free and returns to the sleeping state or, if it needs to start a new job (because the user has many works to do) it is ready to accept it. When a slave node finishes its tasks (as a slave), it has the possibility to become a master or a slave node again but not necessarily for the same master. In this way, we ensure an automatic load balancing schema and the whole system becomes less centralized. Again, the key idea is to rely on existing Desktop Grid middleware but also to control and coordinate multiple instances through Publish/Subscribe systems.

Acknowledgement

Experiments presented in this paper were carried out using the Grid’5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners. We would like also to thank Mathieu Jan for his fruitful comment on an early version of this paper.

References

1. : Teragrid. <http://www.teragrid.org/>
2. : Enabling grids for escience. <http://public.eu-egee.org/>
3. : Osg. <http://www.opensciencegrid.org/>
4. : Grid’5000. URL: <http://www.grid5000.fr>
5. 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, Parallel Processing, 12th International Euro-Par Conference. Volume 4128 of Lecture Notes in Computer Science., Springer (August 2006) 393-404
6. : Instant grid. URL: <http://www.lri.fr/~rezmerit/Instant%20Grid.html>
7. Steinberg, D., Cheshire, S.: Zero Configuration Networking: The Definitive Guide. first edn. O’Reilly Media, Inc. (December 2005)
8. : ZeroConfiguration. URL: <http://www.zeroconf.org>
9. : Avahi. URL: <http://www.avahi.org>
10. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware ’01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, London, UK, Springer-Verlag (2001) 329-350
11. : FreePastry. URL: <http://www.freepastry.org>

12. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2001) 161–172
13. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2001) 149–160
14. : OAR. URL: <http://gforge.inria.fr/projects/oar/>
15. : KADeploy. URL: <http://gforge.inria.fr/projects/kadeploy/>
16. Antoniu, G., Cudennec, L., Duigou, M., Jan, M.: Performance scalability of the JXTA P2P framework. In: Proc. 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, CA, USA (March 2007) 108–131
17. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V., Lodygen-sky, O.: Computing on large scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. In: Future Generation Computer Systems. Volume 21 of issue 3. (2005) 417–437