Peer-to-Peer for Computational Grids: Mixing Clusters and Desktop Machines

Denis Caromel and Alexandre di Costanzo and Clément Mathieu

INRIA Sophia - I3S - CNRS - Université de Nice Sophia Antipolis
INRIA, 2004 Rt. des Lucioles, BP 93
F-06902 Sophia Antipolis Cedex, France

Abstract

This paper presents a Peer-to-Peer (P2P) infrastructure that supports a large scale grid. The P2P infrastructure is implemented in Java and federates Java Virtual Machines (JVMs) for computation. The management of shared JVMs is decentralized, self-organized, and configurable.

The P2P infrastructure was deployed as a permanent desktop grid, with which we have achieved a computation record by solving the NQueens problem for 25 queens. Thereafter, we have mixed this desktop grid with five highly heterogeneous clusters from the Grid'5000 platform. We analyze the behavior of this thousand CPU grid with two communicating applications: NQueens and Flow-Shop.

Key words: Grid, Unstructured Peer-to-Peer, Communicating application

1 Introduction

These last years, computing grids have been widely deployed around the world to provide high performance computing tools to research and industrial fields. Those grids are generally composed of dedicated clusters. In parallel, an approach for using and sharing resources called Peer-to-Peer (P2P) networks has also been deployed. In P2P networks, we can discern two categories: Edge Computing or Global Computing, such as SETI@home [1], which takes advantage of machines at the edges of the Internet; and P2P files sharing, such as Gnutella [2], which permits Internet users to share their files without central servers.

Email address: `First.Last@sophia.inria.fr` (Denis Caromel and Alexandre di Costanzo and Clément Mathieu).

There are many definitions of a P2P network: decentralized and non-hierarchical node organization [3], or taking advantage of resources available at the edges of the Internet [4]. In this paper, a P2P network follows the definition of a "Pure Peer-to-Peer Network", according to [5], meaning that it focuses on sharing resources, decentralization, and peer failures.

Grid users have usually access to one or two clusters and have to share their computation time with others; they are not able to run computations that would take months to complete because they are not allowed to use all the resources exclusively for their experiments. At the same time, these researchers work in labs or institutions, which have a large number of desktop machines. Those desktops are usually under-utilized and are only available to a single user. They also are highly volatile (*e.g.* shutdown, reboot, failure). Organizing such desktop machines as a P2P network for computations or other kinds of resource sharing is now increasingly popular.

However, existing models and infrastructures for P2P computing are limited as they support only independent worker tasks, usually without communications between tasks. However P2P computing seems well adapted to applications with low communication/computation ratio, such as parallel search algorithms. We therefore propose in this paper a P2P infrastructure of computational nodes for distributed communicating applications.

The proposed P2P infrastructure is an unstructured P2P network, such as Gnutella [2]. In contrast to others P2P approaches for computing, which are usually hierarchical or master-salve, our approach is original in the way that an unstructured P2P network commonly used for file sharing can be also used for computing.

The P2P infrastructure has three main characteristics. First, the infrastructure is decentralized and completely *self-organized.* Second, it is flexible, thanks to parameters for adapting the infrastructure to the location where it is deployed. Finally, the infrastructure is portable since it is built on top of Java Virtual Machines (JVMs). Thus, the infrastructure provides an overlay network for sharing JVMs.

The infrastructure allows applications to transparently and easily obtain computational resources from grids composed of both clusters and desktops. The application deployment burden is eased by a seamless link between applications and the infrastructure. This link allows applications to be communicating, and to manage the resources' volatility. The infrastructure also provides large scale grids for computations that would take months to achieve on clusters.

The contributions of this paper are:

- an unstructured P2P overlay network for sharing computational resources;
- building grids by mixing desktops and clusters;
- deploying communicating applications; and,
- achieving computations that take months on clusters.

In Section 2, we present some related work. Next, Section 3 presents our P2P infrastructure. Then, Section 4 presents experiments with the P2P infrastructure, which has allowed us to deploy a permanent desktop grid on 260 desktops. In Section 5, we show how the P2P infrastructure has also permitted us to benchmark a communicating application with Grid'5000 and our desktop grid. We also deployed an application on 1007 CPUs. Finally, we discuss and evaluate the P2P infrastructure.

2 Related Work

Unstructured P2P networks, such as Gnutella [2] and KaZaA [6], do not organize the overlay network of peers, each peer maintains a list of connections to other peers, called neighbors or acquaintances. Due to the lack of structure, there is no information about the location of resources, therefore peers broadcast queries through the network, through a method called "flooding". In order to limit the cost of flooding many mechanisms have been proposed: Dynamic Querying [7], which dynamically adjust the TTL of queries; or by dynamically adapting the search algorithms [8].

Distributed Hash Table (DHT), such as Chord [3] or Pastry [9], organize shared resources (principally data) in order to satisfy requests very efficiently. These kinds of systems are structured P2P overlay networks because they try to organize the P2P network topology and/or the data distribution, such that looking for a resource requires $O(logn)$ steps, whereas in comparison unstructured P2P networks require $O(n)$ steps.

BOINC [10] (Berkely Open Infrastructure for Network Computing) is a global computing platform. Global computing platforms use spare CPU cycles from machines, which run at the edges of the Internet. The original and most famous BOINC application is **SETI@home** [1]. **XtremWeb** [11] is also a global computing platform. Those platforms principally run master-slave applications. Workers download tasks from a server and when tasks are completed, results are sent back to the server. Neither BOINC nor XtremWeb enable tasks to communicate between each other. In addition, users have to provide different compiled versions of their tasks to be able to use workers, running on different architectures, different operating systems, *etc.*

Condor [12] is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Condor can be used to build Grid-style computing environments that cross administrative boundaries. Condor allows multiple Condor compute installations to work together. It also incorporates many of the emerging Grid-based computing methodologies and protocols. For instance, Condor-G [13] is fully interoperable with resources managed by Globus. Condor uses a mechanism called *matchmaking* for managing resources between owners and users. This mechanism is mainly based on a matchmaking algorithm, which compares two descriptors called *classified advertisements* (ClassAds); one describes the task and one describes the compute node, *i.e.* there are two kinds of ClassAds: Jobs ClassAd and Machines ClassAd. Users have to provide this Job ClassAd with the query description of which resources are needed. Condor is well suited for deploying batch jobs, which need to run on specific resources, such as on a given architecture types.

OurGrid [14] is a complete solution for running Bag-of-Tasks applications on computational grids. These applications are parallel applications whose tasks are independent. Furthermore, OurGrid is a cooperative grid in which labs donate their idle computational resources in exchange for accessing other labs idle resources when needed. It federates both desktops and dedicated clusters. The architecture of this middleware is composed by three main components: *MyGrid*, a scheduler, which is the central point of the grid and provides all the necessary support to describe, execute, and monitor applications; *Peers* have as their main role to organize and provide machines that belong to the same administrative domain, *i.e.* they are machine providers for the scheduler; and *User Agents* run on each grid machine and provide access to functionality of the machines (User Agents are registered in Peers). As with Condor, OurGrid users have to provide a Job descriptor file, which queries resources and describes all tasks of the application to deploy.

JXTA Project [15] is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner. JXTA creates a virtual network where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs or are relying on different network transports. JXTA is a low-level specifications/systems for P2P communications. The communication protocol is based on an XML data structure. Indeed all communications are handled by the JXTA P2P network. JXTA can be seen as a network layer, like TCP/IP sockets, on which developers build their applications.

Unlike DHTs, our infrastructure is an unstructured P2P network. Although DHTs avoid flooding, this approach cannot be used here. The main goal of our infrastructure is to provide computational nodes (JVMs) to applications, where applications ask for a desired number of nodes and then the infrastructure broadcasts the request to find any available nodes to satisfy the request. Here queries do not target a specified resource, such as data identified by a hash, but instead try to find a number of available peers. In addition, DHTs earned some criticisms for their high maintenance cost with high churn [8], where the system has to discover failures, re-organize lost data and pointers, and then manage data re-organization when the failed peers return to the network.

Like unstructured approaches we propose that each peer maintains a list of acquaintances and also some modifications to the basic flooding protocol to make the system scalable. Unlike all these systems we do not focus on data sharing, instead we propose some modifications in order to adapt the flooding to find available nodes (JVMs) for computation.

Unlike global computing platforms (BOINC/SETI@home and XtremWeb), Condor, and OurGrid, we do not provide any job schedulers. Applications connect to our P2P infrastructure and request nodes, nodes are returned in a best-effort way by the infrastructure. Unlike the others, applications dynamically interact with the infrastructure to obtain new nodes. The infrastructure works as a dynamic pool of resources. Once applications get nodes, there are no restrictions on how they are used. This property allows applications to communicate easily in arbitrary ways. Application communications are not handled by our infrastructure, unlike other P2P networks. Most other P2P infrastructures require the overlay network to be used for all communications. This limitation is highlighted by JXTA, which has very poor communication performance [16]. With our approach, applications can freely use different communications transport layers.

3 A Self-Organized and Flexible Peer-to-Peer Infrastructure

3.1 First Contact: Bootstrapping

A well-known problem of P2P networks is the bootstrapping problem, also called the first contact problem. This problem can be solved by many different network protocols, such as JINI [17]. It can be used for discovering services in a dynamic computing environment. This protocol seems to be perfectly adapted to solve the bootstrapping problem. However, there is a serious drawback for using this protocol: JINI needs to be deployed on a network with IP multicast communications allowed. That means JINI cannot be widely distributed.

Therefore, a different solution for the bootstrapping problem was chosen, inspired from super-peer networks [18]. A fresh peer has a list of "registry" addresses. These are peers that have a high potential to be available; they are in a certain way the P2P network core. The fresh peer tries to contact each registry with this list. When a registry is responding, it is added to the fresh peer list of known peers (acquaintances). When the peer has connected to at least one registry, it is a member of the P2P Network.

3.2 Discovering Acquaintances

The main problem of the infrastructure is the high volatility of peers because those peers are desktop machines and clusters nodes, possibly available for a short time.

Therefore, the infrastructure aims at maintaining an overlay network of JVMs alive; this is called *self-organizing*. When it is impossible to have external entities, such as centralized servers, which maintain peer databases, all peers must be capable of staying in the infrastructure by their own means. The strategy used for achieving self-organization consists of maintaining, for each peer, a list of acquaintances.

The infrastructure uses a specific parameter called *Number of Acquaintances* (NOA): the minimum number of known acquaintances for each peer. Peers update their acquaintance list every *Time to Update* (TTU), checking their own acquaintance list to remove unavailable peers, *i.e.* they send heartbeat messages to them. When the number in the list is less than NOA, a peer will try to discover new acquaintances. To discover new acquaintances, peers send exploring messages through the infrastructure by flooding the P2P network.

The exploring message is sent every TTU until the length of the list is greater than the NOA value. This message is sent with a unique identifier, with a reference to the sender, and with the *Time To Live* (TTL) in number of hops. The TTL and the unique identifier limit the network flooding.

When a peer receives an exploring message, it has to:

(1) check the unique identifier: if it is an old message, drop it and do nothing;
(2) store the unique identifier;
(3) if the requester is not already in its acquaintance list: use a function to determine if the local peer has to answer. This function is for the moment a random function, in a future work we would like to improve the network organization.
(4) then if the TTL decremented is greater than 0, broadcast the message to all its acquaintances.

Finally, NOA, TTU, and TTL are all configurable by the administrator, who has deployed the P2P infrastructure. Each peer can have its own value of those parameters and the values can be dynamically updated.

3.3 Asking Resources

For the infrastructure and the application, all resources are similar. The infrastructure is *best-effort*, applications ask for computational nodes, JVMs, but the infrastructure does not guarantee that applications requests can be satisfied (not enough free nodes, *etc.*). Usually applications request nodes from the infrastructure and then the infrastructure returns node references back to the application. All requests from applications are in competition to obtain available resources.

In order to satisfy node queries faster, we distinguish three cases.

First, the application needs *only one node*, then the query node message uses a random walk algorithm, which means that the next hop is randomly chosen. The message is forwarded peer by peer until a peer has a free node to share or until TTL reach zero. While no free node has been found the application has to re-send the message at each TTU increasing eventually to the TTL.

Second, the application needs *n nodes* at once, for that case the resource query mechanism used is similar to the Gnutella [2] communication system, which is based on the Breadth-First Search algorithm (BFS). Messages are forwarded to each acquaintance, and if the message has already been received or if its TTL reach 0, it is dropped. The message is broadcast by the requester every TTU until the total number of requested nodes is reached or until a global timeout occurs. Also, we have added a kind of transactional commit. When a peer is free, it sends a reference on its node to the requester. Before forwarding the message the current peer waits for an acknowledgment from the requester because the request could have already been fulfilled. After an expired timeout or a non-acknowledgment, the peer does not forward the message. Otherwise, the message is forwarded until the end of TTL or until the number of requested nodes reaches zero. The acknowledgment message from the requester is indeed the total number of nodes still needed by the requester. We can distinguish three states for a peer: free, busy, or booked. This mechanism is specified in Message Protocol 1.

Third, the application may ask for all available nodes, the message protocol is close to the previous one but does not need to wait for an acknowledgment and the message is broadcast every TTU until the application end.

Message Protocol 1 Asking for n nodes at once from the P2P infrastructure. This protocol shows the response by a peer when it receives an **n nodes** request:

```
Require: A remote reference on the node requester $Node\_Requester$,
  the request $TTL$,
  the request $UniqueID$, and
  the requested number of nodes $n$
Ensure: At most $n$ free nodes for computation
  if $Node\_Requester$ is $Myself$ or $allPreviousRequests$ contains $UniqueID$ then
    drop the request
  else
    Add $UniqueID$ in $allPreviousRequests$
    if I have a free node then
      Send the node to $Node\_Requester$ {$Node\_Requester$ receives the node and decides whether or not to send back an ACK to the peer, which has given the node}
      while not $timeout$ reached do
        wait for an ACK from $Node\_Requester$ {ACK is the number of still needed nodes by the requester, $ACK = 0$ means NACK}
        if ACK received and $TTL > 0$ and $ACK > 1$ then
          Broadcast the message to all acquaintances with $TTL = TTL - 1$ and $n = ACK$
        end if
      end while
    end if
  end if
```

3.4 Peer and Node Failures

The infrastructure itself is stable, according to the definition of "Pure P2P networks" [5], as each peer manages its own list of acquaintances by the heartbeat mechanism (see section 3.2). Therefore the infrastructure can maintain a network of peers until all peers are down, *i.e.* a peer failure is not a problem for the infrastructure.

The issue is at the application level. The infrastructure broadcasts the node request of the application through itself (see section 3.3); when a peer has an available node, it returns directly (point-to-point) to the application a reference to the node. Once the application has received the reference there is no guarantee that the node is still up or if the node will be up for all the time that the application needs it. Therefore it is the application's responsibility to manage node failures.

However the P2P infrastructure is implemented with the ProActive grid middleware (see section 3.5), which proposes a mechanism for fault-tolerance. With this particular use case we have started to work on a mechanism to deploy non-functional services, such as fault-tolerance or load balancing, on a grid [19]. This mechanism allows users to deploy theirs applications on the P2P infrastructure and to have fault-tolerance automatically applied.

Furthermore, the infrastructure does not work as a job/task scheduler, it is just a node provider. Therefore the application has to manage all node failures and its own failures.

3.5 Implementation in ProActive

This P2P infrastructure is implemented with the ProActive grid middleware [20]. It is a 100% Java library, which aims to achieve seamless programming for parallel, and distributed computing. ProActive provides a way to remotely access objects and handles few communication protocols, such as RMI, SSH/RMI, HTTP, and Ibis [21].

The P2P infrastructure is implemented with the ProActive library. Thus, the shared resources are not JVMs but ProActive Nodes [22].

Since the P2P infrastructure is implemented on top of the ProActive library each peer can use a different communication protocol. For example, a peer, on a desktop machine can accept RMI communications but can use RMI/SSH to communicate with another peer within a cluster.

Finally, the P2P infrastructure is fully integrated in ProActive thereby allowing the infrastructure to seamlessly use all ProActive features, such as security.

4 Desktop Experiments

4.1 Environment of Experiments

In order to run our experiments, the *INRIA Sophia P2P Desktop Grid* (InriaP2PGrid) has been deployed on about 260 desktop machines of the INRIA Sophia Antipolis lab; this grid is now a permanent grid managed by our P2P infrastructure. All these desktop machines are running various GNU/Linux distributions or Microsoft Windows XP as operating systems, on Intel CPUs, from Pentium 2 to dual-Pentium 4. As to not interfere with daily work, the JVMs, Sun 1.4.2 or Sun 1.4.1, are started with the lowest system priority.

Also all desktop machines by default work during the night (8:00pm to 8:00am) and during weekend (Friday 8:00pm to Monday 8:00am), this group is called *INRIA-ALL*, and a sub-group of these machines always work, this sub-group is called *INRIA-2424*, this group counts 53 machines. Machines of INRIA-2424 are selected in regard to their CPU power, thus they are the fastest one. Also, the users could interrupt the computation if our experiments bother them. The INRIA-2424 peers are used as registries (all registries use themselves as registries); and at fixed moments the rest of INRIA-ALL machines join the P2P infrastructure by contacting those registries. Figure 1 shows the InriaP2PGrid structure.

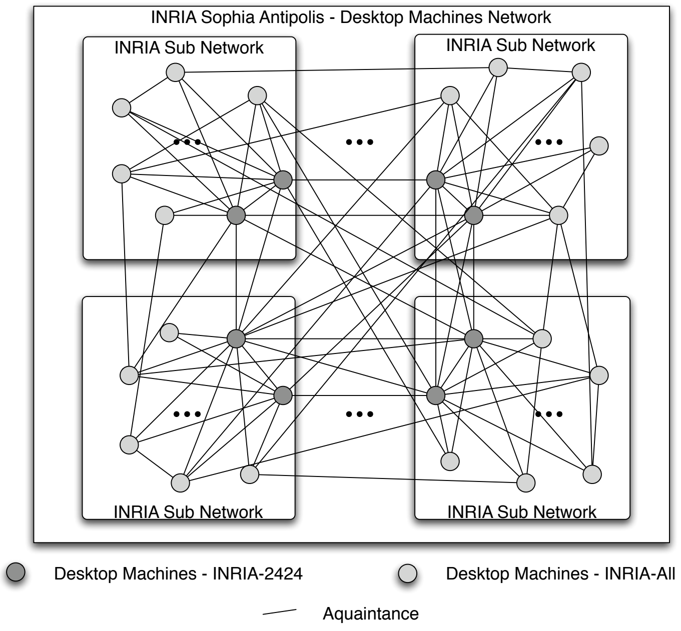
INRIA Sophia Antipolis - Desktop Machines Network
INRIA Sub Network
INRIA Sub Network
INRIA Sub Network
INRIA Sub Network
Desktop Machines - INRIA-2424
Desktop Machines - INRIA-All
Aquaintance

Fig. 1. Desktop experiments: *INRIA Sophia P2P Desktop Grid* structure

The repartition of CPU frequencies of all desktop machines are summarized in Figure 2.

Finally, the values of the P2P infrastructure parameters used by the InriaP2PGrid are:

- $NOA = 30$ *acquaintances*: each sub-network contains on average 20 machines, so a peer discovers some acquaintances outside of its sub-network.
- $TTU = 10$ *minutes*: INRIA-2424 machines are highly volatiles, we have observed on this group that on average 40 machines are available out of 53. Every 10 minutes, one peer out of 30 becomes unavailable (host down, JVMs killed by users *etc.*). It usually joins back the infrastructure some time later.
- $TTL = 3$ *hops*: it is the diameter of the network, as shown by Figure 1.

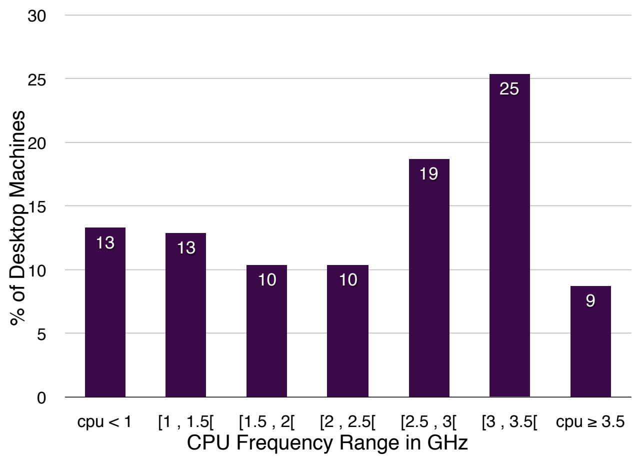

Fig. 2. Desktop Grid: CPU frequencies of the desktop machines

4.2 NQueens: Computation Record

With the InriaP2PGrid managed by our P2P infrastructure, we are the first, as referenced by [23], to solve the NQueens problem with 25 queens. All the results of the NQueens experiment are summarized in Table 1. The experiment took six months for solving this problem instance. The result was confirmed later by Pr. Yuh-Pyng Shieh from the National Taiwan University.

The NQueens problem consists in placing n queens on a $n \times n$ chessboard so that no two queens are on the same vertical, diagonal, or horizontal line (*i.e.* attack each other). We aim to find all solutions with a given n. The chosen approach to solve the NQueens problem was to divide the global set of permutations in a set of independent tasks. Then a master-slave model was applied to distribute these tasks to the workers, which were dynamically deployed on the InriaP2PGrid.

Table 1
Desktop experiments: NQueens experiment summary with n=25

Total of Solution Found	$2,207,893,435,808,352$ ($\approx 2 quadrillions$)
Total # of Tasks	$12,125,199$
Total Computation Time	4444 *hours* 54 *minutes* 52 *seconds* (≈ **185 days**)
Average Time of One Task Computation	≈ 2 *minutes and* 18 *seconds*
Equivalent Single CPU Cumulated Time	464344 *hours* 35 *minutes* 33 *seconds* (≈ **53 years**)
Total # of Desktop Machines	260 (max of 220 working concurrently)

Moreover, Figure 3 shows the number of peers by days, which participated to the NQueens computation. This graph does not report all the experiment period time, only three months. During all the time of this experiment, about six months, the infrastructure counted 260 different desktop machines and a top of 220 machines working at the same time. As shown in Fig. 3 the infrastructure was totally down 3 times, owing to global lab power cuts. Fig.

3 shows some troughs where the infrastructure provided less peers for the computation, these troughs result from some network hardware failures.

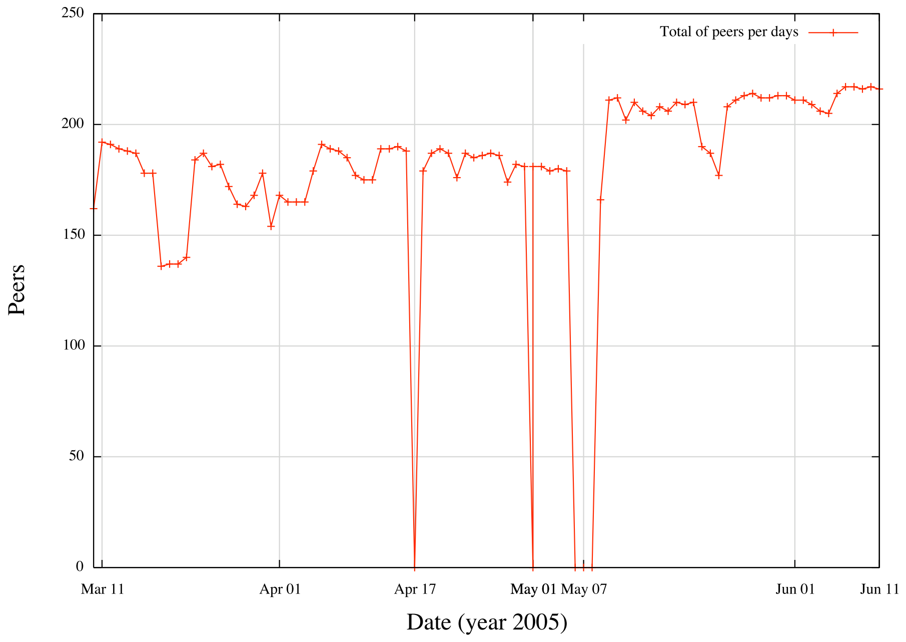

Fig. 3. Desktop experiments: number of peers per day, which have participated in the NQueens computation.

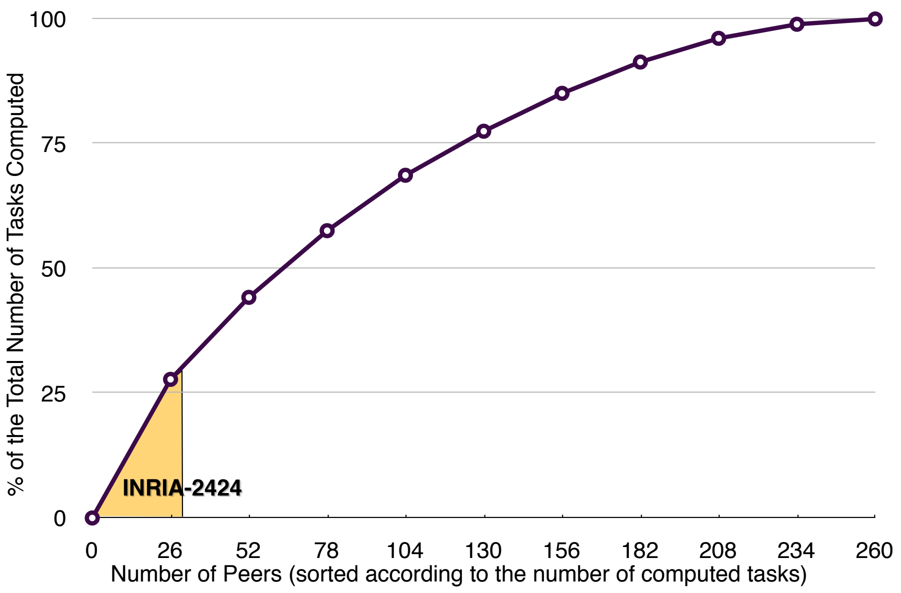

Fig. 4. Desktop Grid: Percentage of tasks computed by all peers

Figure 4 shows the percentage of tasks computed by the workers. To plot this graph we first sort all machines according to the number of tasks computed. Then we calculated the percentage of tasks computed by those workers. We observe that 10% of all workers (26 workers) have computed 27.78% of total tasks and that 20% has computed 44.21% of total tasks. We also observed that the first 28 workers ($\approx$ 11%) of this graph are all members of the group *INRIA-2424*. It is a normal observation because these machines are the most powerful and work 24 hours a day. However, it is only a half of this group, the group counting 53 machines. The JVMs of the second part ran on machines over-loaded because used by usual users or the JVM was often killed by users, hardware failures, ran on more unstable systems, *etc.*

The INRIA-2424 are selected in regard to their CPU power, thus it is normal that they computed a large number of tasks. Also, the INRIA-ALL is composed of a large number of less powerful machines. Figure 2 shows that about 46% of machines have CPU speed less that 2.5GHz, *i.e.* machines at least 2 years old. These machines have computed 34% of the total number of tasks.

All this experimentation and figures show that it is hard to forecast which machines we have to choose for improving the total computation time.

5 Mixing Desktops and Clusters Experiments

5.1 Environment of Experiments

In addition to the InriaP2PGrid, described in section 4.1, we have access to a large scale nationwide infrastructure for grid research, *Grid'5000* (G5K) [24]. The G5K project aims at building a highly reconfigurable, controllable and monitorable experimental grid platform gathering 9 sites geographically distributed in France featuring a total of about 2000 CPUs.

G5K is composed of a large number of machines, which have different kinds of CPUs (dual-AMD Opteron 64 bits, dual-PowerPC G5 64 bits, dual-Intel Itanium 2 64 bits, and dual-Intel Xeon 64 bits), of operating systems (Debian, Fedora Core 3 & 4, MacOs X, *etc.*), of supported JVMs (Sun 1.5 64 bits and 32 bits, and Apple 1.4.2), and of network connexion (Gigabit Ethernet, and Fast Ethernet).

Figure 5 shows the grid used for our experiments. This grid is a mix of InriaP2PGrid and G5K clusters. The left of the figure shows the INRIA Sophia P2P Desktop Grid wherein INRIA-2424 peers are used as registries (all registries use themselves as registries); and at fixed moments the rest of INRIA-ALL machines join the P2P infrastructure by contacting those registries. In addition, the right of the figure shows the G5K platform, clusters of G5K are connected to the P2P infrastructure by a few INRIA-2424 peers, and each of these peers handles a G5K site. These peers do not share their local JVMs but share JVMs deployed on clusters. G5K and INRIA networks are both closed network with only few SSH access point, and G5K is a NAT, thus communications between INRIA and G5K are tunneled via SSH.

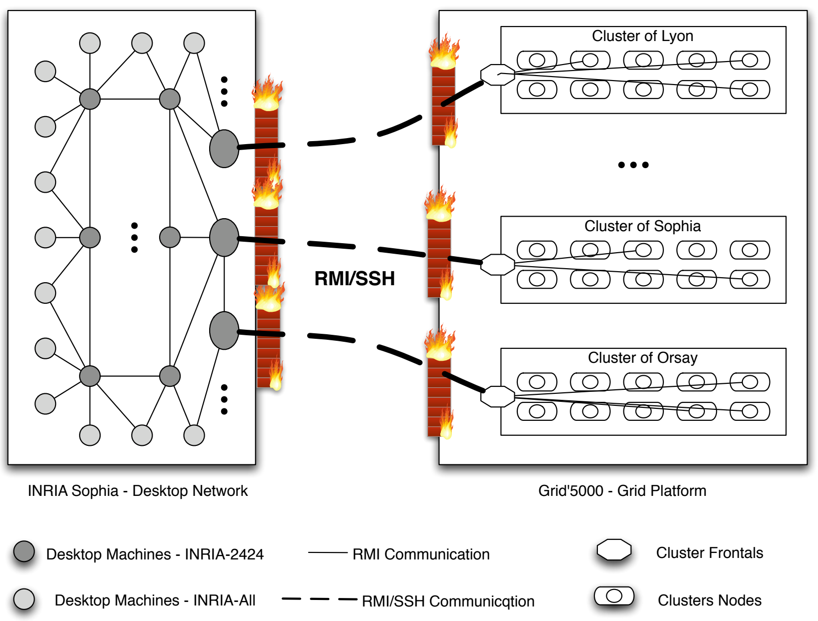

Fig. 5. Mixing Desktop and Clusters: environment of experiment structure

5.2 NQueens: Large Scale Grid

We took the same application as previously, the NQueens (see section 4.2), and ran it on a grid that it is a mix of machines from *INRIA Sophia P2P Desktop Grid* and from clusters of G5K.

Experiments run with $n = 22$. Figure 6 shows all results. Counting up all the machines, we reached 1007 CPUs. The execution time decreases with a large number of CPUs on a heterogeneous grid, mix of desktop and clusters.

We show that an embarrassingly parallel application, such as NQueens, benefits of the large number of CPUs provided by our P2P infrastructure, even if this grid is highly heterogeneous. For that kind of application, mixing desktop and cluster machines is beneficial.

5.3 Flow-Shop: Communicating Application

To illustrate that our P2P infrastructure can be used to deploy communicating applications, we consider a *master-slaves* application for solving Flow-Shop

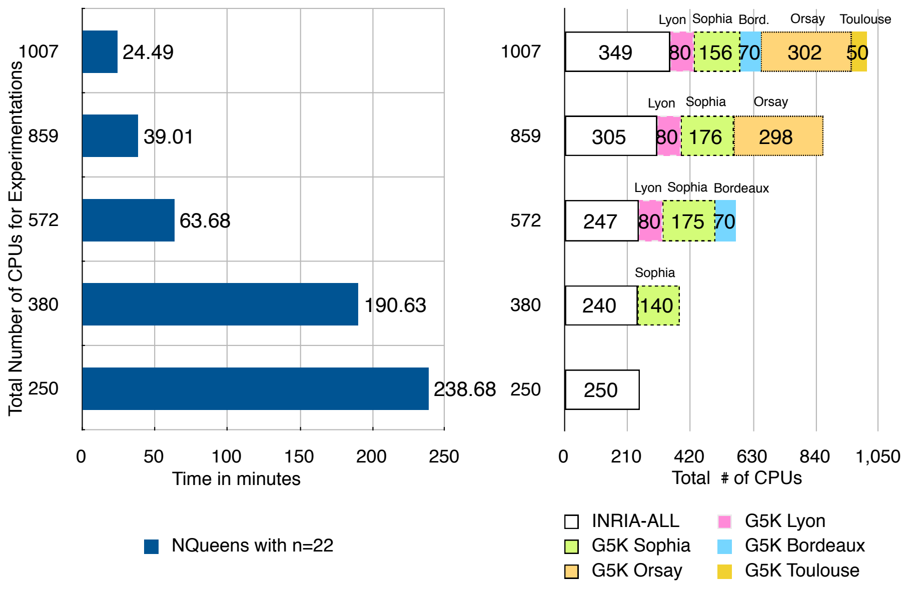

Fig. 6. Mixing Clusters and Desktops: NQueens with $n = 22$ benchmark results

problems. A Flow-Shop problem aims to find the optimal schedule of a set of jobs on a set of machines in order to minimize the total execution time; this problem can be solved by exploring a solution tree. The whole solution tree is explored in parallel, and while exploring the tree, the current best solution is shared within the application. As opposed to the NQueens problem, we have communications between workers and they discard tree branches, which are slower than the best current solution.

The solution tree of the problem is divided by a manager into a set of tasks. The manager manages task allocation to the workers. The manager handles dynamic acquisition of new workers and also handles worker failures by reallocating failed tasks. Workers perform communications between each other to synchronize the best current solution.

This approach has a low communication/computation ratio, even if at the beginning of the computation the application has to deploy workers on all CPUs and workers are also intensively broadcasting new better solutions. The intensive phase takes 20 minutes for the run of one hour on 628 CPUs. With this run we were able to measure the communication size on 116 CPUs of the G5K Sophia cluster. We measured 143 MB of network traffic inside the cluster for this first 20 minutes. The bandwidth used is about 120 KB/s. After, there are only sporadic communications until the best solution is found.

Figure 7 shows all results of Flow-Shop computations with an instance of *17 jobs/17 machines*. An analysis of Figure 7 shows that computation time decreases with the number of used CPUs. However, the increase in execution

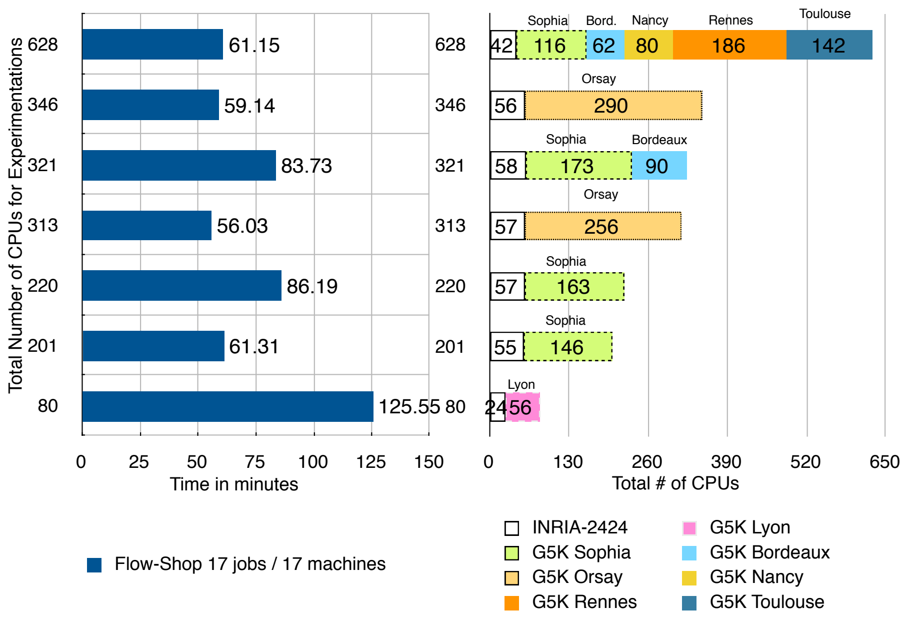

Fig. 7. Mixing Clusters and Desktops: Flow-Shop benchmark results

time between 201 and 220 comes from a communication bottle-neck between workers of INRIA-2424, which are desktop machines, and workers of G5K Sophia, which are clusters. Communications between *INRIA P2P Desktop Grid* and G5K are tunneled in *SSH*. This bottle-neck can also be observed on the run with 321 CPUs on three sites. The lower impact of bottle-neck with 313 CPUs can be explain by the distribution of the tasks and by the fact that there is only one cluster. Then the last experimentation with 628 CPUs has an execution time close to the bench with 346 CPUs, we explain that by a long phase of deployment, 11 minutes, opposed to only 6 minutes for 346 CPUs.

In addition, all these benchmarks were realized during working days, so usual users have a higher priority to execute their process. Unfortunately, we were unable to deploy the Flow-Shop problem on more CPUs, because those bottle-neck create a too large number of connections on peers which are in charge to tunneling communication between G5K and desktops.

To conclude, we have been able to deploy a communicating application using a P2P infrastructure on different sites which provided 628 CPUs. Those CPUs were composed of heterogeneous architectures, and came from desktop machines and clusters.

6 Discussion and Future Work

The infrastructure itself is stable, but its resources are unstable, thus applications have to manage the node volatility; that means that applications have to be fault-tolerant. The infrastructure provides available nodes to the applications, even if some resources are available only a few hours a day.

The main goal of this work is to propose an infrastructure, which provides JVMs for computations that would take months to achieve on clusters; these JVMs run on different kinds of resources: clusters and desktops. This infrastructure has to be dynamic for two reasons: first, desktops are used by their owners and second clusters cannot usually be monopolized for an extended period by the same application.

The NQueens computation record shows that it is possible to use the P2P infrastructure to run an application, which needs months of computations to achieve on a grid of desktop machines. All others global computing platforms, such as XtremWeb or BOINC/SETI@home also do this, but support only master-slave applications without communication between slaves and require the deployment of applications on servers dedicated to the problem. Nevertheless, these systems support multi-application deployments because they have scheduler for resource allocation. This is currently an issue with our approach, which uses flooding, to do multi-application deployments. For instance, an application can monopolize all nodes for a while and others applications cannot get nodes to use. We have started to work on a job scheduler, which uses the infrastructure as a node provider. The scheduler will be the only one application deployed on the infrastructure and the scheduler will schedule some applications as jobs on the infrastructure.

As opposed to global computing platforms, using our infrastructure does not provide support for application fault-tolerance. The main function of the infrastructure is to provide nodes to applications; applications can have different architectures, such as master-slaves, bag-of-tasks, SPMD, *etc.* Therefore, the infrastructure is not able to manage application faults, and thus it is the application responsibility up to manage faults.

The infrastructure is integrated in a grid library, ProActive, that provides the link between applications and the infrastructure. This link provides a high-level of abstraction of the resource's usage. That integration with the library allows running all kinds of distributed applications, such as master-slaves, communicating, *etc.* Also, having an implementation in Java allows portability and use of all kinds of resources, operating systems and machine architectures. Nevertheless, Java is not perfect in terms of running performance, but it is portable. ProActive now provides some tools for wrapping and distributing native MPI executables, such as FORTRAN and C.

We propose several parameterised protocols (see section 3.3). We have currently fixed parameters value by experimentation for our testbed, but we believe that it will be interesting to have an evaluation of their impact on the infrastructure.

Finally, in this kind of grid environment an open infrastructure with no management tools, there are significant security issues. For the moment, the infrastructure conceptually does not provide security, but the implementation with ProActive provides a security mechanism based on certificates. This mechanism can constrain a node to run only applications with a known certificate.

Lastly, our experimentation show that it could have a bottleneck when many nodes behind a firewall try to communicate with the outside. The network topology (NAT, firewalls, *etc.*) could affect performances of applications, notably with highly communicating applications. With the mixed environment experimentation shows that it is well adapted for embarrassingly parallel applications and also could be used for applications with a low communication/-computation ratio.

7 Conclusion

The proposed Peer-to-Peer infrastructure is an unstructured overlay network for sharing JVMs. This infrastructure provides a new way for building grids, which are a mix of desktop machines and of clusters. Such grids can be used for deploying communicating applications and for running computations that would take months to achieve on clusters.

We show with experiments that our infrastructure allows the deployment of applications on a large scale grid and also it is stable enough to run applications for months. Furthermore, the infrastructure allows running various kinds of distributed applications, including communicating applications on a mix of clusters from Grid'5000 and desktops. We also ran a master-slave application on the infrastructure on a such grid, totaling 1007 CPUs.

Today, grids such as Grid'5000 are over-loaded with users, it is hard to find slots for deploying jobs on a large number of nodes and for a long time (days or months). It is sometimes possible with dedicated production clusters, rarely for large grids. We believe that the P2P infrastructure could be easily used to manage grids for that kind of use. The infrastructure dynamically provides nodes from clusters and desktops, upon availability. An application could be run on desktops and sometimes profit from nodes of clusters by intertwining small jobs between bigger jobs from usual cluster users.

References

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, Seti@home: an experiment in public-resource computing, Commun. ACM 45 (11).

[2] Gnutella, http://www.gnutella.com.

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Trans. Netw. 11 (1) (2003) 17–32.

[4] A. Oram, Peer-to-Peer : Harnessing the Power of Disruptive Technologies, O'Reilly & Associates, Sebastopol, CA, 2001.

[5] R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications., in: Peer-to-Peer Computing, 2001.

[6] KaZaA, http://www.kazaa.com.

[7] D. Q. Protocol, http://www.the-gdf.org/index.php?title=Dynamic_Querying.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making gnutella-like p2p systems scalable, in: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, ACM Press, New York, NY, USA, 2003, pp. 407–418.

[9] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001, pp. 329–350.

[10] D. P. Anderson, Boinc: A system for public-resource computing and storage., in: GRID, 2004, pp. 4–10.

[11] V. N. Gilles Fedak, Cécile Germain, F. Cappello, Xtremweb : A generic global computing system, in: I. Press (Ed.), CCGRID2001, workshop on Global Computing on Personal Devices, 2001.

[12] Condor, http://www.cs.wisc.edu/condor/.

[13] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-g: A computation management agent for multi-institutional grids, in: HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, Washington, DC, USA, 2001.

[14] N. Andrade, L. Costa, G. Germoglio, W. Cirne, Peer-to-peer grid computing with the ourgrid community, in: Proceedings of the SBRC 2005 - IV Salão de Ferramentas, 2005.

[15] I. Sun Microsystems, Project jxta: An open, innovative collaboration, http://www.jxta.org/project/www/docs/ (April 2001).

[16] E. Halepovic, R. Deters, Jxta performance study, in: PACRIM. 2003 IEEE Pacific Rim Conference on Communications, Computers and signal, Vol. 1, 2003, pp. .pp. 149– 154.

[17] J. Waldo, K. Arnold, The Jini Specifications, Addison-Wesley, 2000.

[18] B. Yang, H. Garcia-Molina, Designing a super-peer network, icde 00 (2003) 49.

[19] D. Caromel, C. Delbe, A. di Costanzo, Peer-to-peer and fault-tolerance: Towards deployment based technical services (January 2006).
URL `http://hal.inria.fr/inria-00001238/en/`

[20] L. Baduel, F. Baude, D. Caromel, A. Contes, F. Huet, M. Morel, R. Quilici, Grid Computing: Software Environments and Tools, Springer Verlag, 2005, Ch. Programming, Composing, Deploying, for the Grid.

[21] R. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. F. H. Hofman, C. J. H. Jacobs, T. Kielmann, H. E. Bal, Ibis: a flexible and efficient java-based grid programming environment, Concurrency - Practice and Experience 17 (7-8) (2005) 1079–1107.

[22] F. Baude, D. Caromel, L. Mestre, F. Huet, J. Vayssière, Interactive and descriptor-based deployment of object-oriented grid applications, in: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, Edinburgh, Scotland, 2002, pp. 93–102.

[23] N. J. Sloane, Sloane a000170, http://www.research.att.com/.

[24] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, O. Richard, Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform, in: Grid'2005 Workshop, IEEE/ACM, Seattle, USA, 2005, to appear.