Going Large-scale in P2P Experiments Using the JXTA Distributed Framework

Gabriel Antoniu, Luc Bougé, Mathieu Jan, Sébastien Monnet

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Going Large-scale in P2P Experiments Using the JXTA Distributed Framework

Gabriel Antoniu, Luc Bougé, Mathieu Jan, Sébastien Monnet

N°5151

March 2004

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5151--FR+ENG

Going Large-scale in P2P Experiments Using the JXTA Distributed Framework

Gabriel Antoniu, Luc Bougé, Mathieu Jan, Sébastien Monnet

Thème 1 — Réseaux et systèmes
Projet Paris

Rapport de recherche n˚5151 — March 2004 — 12 pages

Abstract: The interesting properties of P2P systems (high availability despite node volatility, support for heterogeneous architectures, high scalability, etc.) make them attractive for distributed computing. However, conducting *large-scale experiments* with these systems arise as a major challenge. Simulation allows to model only partially the behavior of P2P prototypes. Experiments on real testbeds encounter serious difficulty with *large-scale deployment and control* of peers. This paper shows that using an optimized version of the *JXTA Distributed Framework* (JDF) allows to easily deploy, configure and control P2P experiments. We illustrate these features with sample tests performed with our JUXMEM JXTA-based grid data sharing service, for various large-scale configurations.

Key-words: Peer-to-peer, large scale, experiments, JXTA

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Vers la grande échelle dans les expériences P2P: une approche basée sur *JXTA Distributed Framework*

Résumé : Les propriétés intéressantes des systèmes pair-à-pair (haute disponibilité malgré la volatilité des nœuds, support des architectures hétérogènes, passage à l'échelle, etc.) les rendent séduisants pour les systèmes distribués. Toutefois, mener à bien des *expériences à grande échelle* pour ces systèmes apparaît comme un défi majeur. La simulation ne permet de modéliser que partiellement le comportement des prototypes pair-à-pair. Les expériences menées sur des plates-formes physiques font, elles, face à de sérieuses difficultés pour le *déploiement et le contrôle à grande échelle* de pairs. Ce papier montre que l'utilisation d'une version optimisée de *JXTA Distributed Framework* (JDF) permet de facilement déployer, configurer et contrôler des expériences pair-à-pair. Nous illustrons ces fonctionnalitées par des tests réalisés à différentes grandes échelles sur JUXMEM, notre service de partage de données pour la grille basé sur JXTA.

Mots-clé : Pair-à-pair, grande échelle, expériences, JXTA

1 How to test P2P systems at a large-scale?

The scientific distributed systems community has recently shown a growing interest in the Peer-to-Peer (*aka* P2P) model [11]. This interest is motivated by the properties exhibited by P2P systems such as the high availability despite node volatility, the support of heterogeneous architectures and, most importantly, a high scalability. For example, the *KaZaA* network has shown to scale up to 4,500,000 users, an unreachable scale for distributed systems based on the traditional client-server model.

However, the experimental validation phase remains a major challenge for designers and implementers of P2P systems. Validating such highly-scalable systems requires large-scale experimentations, which is extremely difficult. Consider for instance popular P2P software, like *Gnutella* or *KaZaA*: workloads of these systems are not fully analyzed and modeled because the behavior of such systems cannot be precisely reproduced and tested [7]. Recently, P2P systems like *CFS* [6], *PAST* [17], *Ivy* [12] and *OceanStore* [9] based on smarter localization and routing schemes have been developed. However, most of the experiments published for these systems exhibit results obtained via simulation or on very few (typically 4) *physical* nodes, and rarely go up to a few tens of nodes [13]. Even when larger scales are reached via emulation [14], no experimental methodology is discussed for automatic deployment and volatility control. For instance, to simulate node failures, peers are stopped manually using the `kill` signal! There is a definite need for infrastructures allowing to test P2P systems at a large-scale. Several approaches have been considered so far.

Simulation. Simulation allows to define a model for a P2P system and then study its behavior through experiments with different parameters. Simulations are often executed on a single sequential machine. The main advantage of simulation is the reproducibility of the results. However, existing simulators, like *Network Simulator* [3], or *SimGrid* [5], need significant adaptations in order to meet the needs of a particular P2P system. This holds even for specific P2P simulators, like *ng-simulator*, used by *NeuroGrid* [8]. Also, the simulated prototype often needs to be written having in mind the simulator used, which may result in deviations from reality. Last but not least, simulators model simplified versions of real environments. Further validation by alternative techniques such as emulation or experiments in real environments is still necessary.

Emulation. Emulation allows to configure a distributed system in order to reproduce the behavior of another distributed system. Tools like *dummynet* [15] or *NIST Net* [4] allow to configure various characteristics of a network, such as the latency, the rate of loss, the number of hops between nodes, and sometimes the number of nodes (e.g. *ModelNet* [18] and

Emulab/*Netbed* [19]). This allows to emulate networks with various sizes and topologies. However, all the heterogeneity of a real environment (in terms of the physical architecture and software resources) cannot be reproduced. More importantly, the deployment of a given P2P prototype is left to the user: it is a often overlooked, but it actually remains a major limiting factor.

Experiments on real testbeds. Real testbeds such as *GridLab* [1] or *PlanetLab* [16] (usually called grids) are large-scale, heterogeneous distributed environments. They are made of several interconnected sites, with various resources ranging from sensors to supercomputers, including clusters of PC. Such environments may prove helpful for realistic testing of P2P systems. On such platforms, experiments are not reproducible in general. Here again, deployment and configuration control generally has to be managed by the user, which is even more difficult than in the case of emulation, because of the much larger *physical* scale.

To sum up, actually *deploying and controlling* a P2P system over large-scale platforms arises as a central challenge in conducting realistic P2P experiments. The contribution of this paper is to introduce the JDF tool, and demonstrate its use in this area.

2 Case study: towards large-scale experiments in JUXMEM

We are currently building a prototype for a data-sharing service for the grid, called JUXMEM [2] (for *Juxtaposed Memory*). It is designed as a compromise between DSM systems and P2P systems: it provides *location transparency* as well as *data persistence* in a *dynamic environment*. JUXMEM is based on the Sun Microsystems JXTA open platform [22]. JXTA provides basic mechanisms for P2P interaction, which facilitate the design and implementation of custom P2P services. As seen in Section 1, deploying such a prototype on a grid is a challenging problem. The JXTA platform provides a tool called *JXTA Distributed Framework* (JDF) [21], allowing to deploy a JXTA service over a large grid, to configure and control each of its peers. This paper describes how we enhanced the JDF tool to run large-experiments for JUXMEM.

2.1 The JUXMEM Project: a JXTA-based grid data-sharing service

JXTA is an open-source framework, which specifies a set of language- and platform-independent XML-based protocols. It provides a rich set of building blocks for the management of P2P systems. The basic entity is the regular peer (called *edge peer*). Edge peers communicate within the JXTA virtual network through specialized *rendezvous peers*. Peers

can be members of one or several *peer groups*. A peer group consists of peers that share a common set of interests, e.g., peers that share access to some resources.

We used JXTA to build JUXMEM, a prototype for a data-sharing service for a grid consisting of a federation of distributed clusters. Its conceptual architecture takes advantage of this feature: it is *hierarchical*. JUXMEM consists of a network of peer groups, called *cluster* groups, each of which generally corresponds to a physical cluster. All the *cluster* groups are enclosed by the *juxmem* group, which includes all the peers members of the service. Each *cluster* group consists of a set of peers which provide memory for data storage (*providers*). A *manager* monitors the providers in a given *cluster* group. Any node can use the service to allocate, read or write to data as a *client*. All providers which host copies of the same data block make up a *data* group, uniquely identified by an ID. Clients only need to specify this ID to read/write a data block: the platform transparently locates it. JUXMEM can tolerate peer volatility: each data block is replicated across a certain number of providers, according to a redundancy degree specified at allocation time. The number of replicas is dynamically monitored and maintained through dynamic replica creation when necessary.

2.2 The JDF Tool

The purpose of JDF is to facilitate automated testing of JXTA-based systems by providing a generic framework allowing to easily define custom tests, deploy all the required resources and run the tests with different configurations of the JXTA platform.

JDF is based on a regular Java Virtual Machine (JVM) (Version 1.3.1 or higher), a Bourne shell and `ssh` or `rsh`. File transfers and remote control are handled using either `ssh`/`scp` or `rsh`/`rcp`. JDF assumes that all the nodes are visible from the control node. JDF is run through a regular shell script which launches a distributed test. This script executes a series of elementary steps: install all the needed files, configure a JXTA network, run the test, collect generated log files, analyze the overall results and remove intermediate files. Additional options are available, such as killing all remaining JXTA processes, which is useful if the test failed, for instance when setting up new tests. Finally, JDF allows to run a sequence of such distributed tests.

A distributed test is specified by the following elements. 1) The set of Java classes describing the behavior of each peer. These Java classes must *extend* the framework provided by JDF, in order to easily start JXTA, stop JXTA and save the results into files. These files are collected by JDF on each node and sent back to the control node to be analyzed by an additional Java class specified by the user. 2) A XML file describing the requested JXTA-based network, such as edge peers, rendezvous peers, relay peers and their associated Java classes, and how they are linked together. This file also allows to specify the number of

peers launched on each node. 3) A file containing the list of nodes where to deploy and run the previously described JXTA network, as well as the path of the JVM used on each node.

An important issue when running tests on grid infrastructures regards the necessary interaction between the deployment tool and the resource allocator available on the testbed. Some grid environments (e.g. Globus) may rely on batch systems (like PBS [23], etc.) in order to dynamically allocate nodes to jobs scheduled on a given cluster. JDF requires as an input a static network file which lists the physical nodes involved. On platforms using dynamic resource allocators, such a list cannot be provided at the time of the job submission. We therefore realized an enhanced version of JDF which allows the job to be submitted without such a file; it then interacts with the batch system and dynamically creates the file at the time when the nodes are assigned to the job, before the job is launched.

3 Sample experiments in JUXMEM

The resources provided by grid computing seem a viable solution for analyzing the huge masses of data produced by large projects such as genomes sequencing. The GriPPS (*Grid Protein Pattern Scanning*) project aims at developing and adapting bioinformatics algorithms so that they can exploit such architectures. When executed on grid environments (e.g. DIET [20]), these adapted applications can benefit from a grid data-sharing service such as JUXMEM. In such applications, data that need to be analyzed are stored in large centralized data banks. Therefore, the transparent data localization and transfer provided by JUXMEM allows biologists to efficiently search for interesting patterns in newly sequenced proteins, for instance. The persistent storage also allows to store parts of these large data on the different sites of a grid that need them, avoiding unnecessary transfers of the data banks. In order to perform realistic experiments for these applications, we need to test our JUXMEM infrastructure on configurations of the order of hundreds to thousands of nodes. This is where deployment tools like JDF can prove to be very helpful.

3.1 Deploying JUXMEM on various grid configurations

The most basic experiments require the deployment of the JUXMEM service on different network configurations. We use the JDF network file to describe the different (application-specific) peer profiles involved by specifying if they correspond to rendezvous peers, how they are interconnected, etc. To facilitate testing on various large-scale configurations, we modified JDF by introducing the ability of specifying the number of instances for each peer profile. For instance, let us assume we deploy a small JUXMEM network of 2 interconnected cluster groups, on 2 different clusters, each of which consisting of 10 providers. An extract

```
<profile name="clusterManager1" instances="1">
<profile name="clusterManager2" instances="1">
...
<profile name="providers1" instances="10">
<profile name="providers2" instances="10">
```

```
<profile name="clusterManager1" instances="1">
...
<profile name="clusterManager10" instances="1">
...
<profile name="providers1" instances="100">
...
<profile name="providers10" instances="100">
```

Figure 1: A small JUXMEM network (left) and a large one (right).

of the corresponding JDF network file is shown on the left side of Figure 1. It defines two profiles for the cluster managers (1 instance each) and two provider profiles (10 instances each). The provider profile references a manager profile to which it is connected, hence the need for 2 different profiles for both type of peers.

Now, imagine we target a more complex JUXMEM network, e.g. consisting of 10 cluster groups with 100 providers each. The new network file is shown on the right side of Figure 1. To do this, we need: 1) to increase the number of profiles for both kinds of JUXMEM peers; 2) increase the number of instances for each provider profile. This is how we can use JDF to deploy the same JUXMEM test on various network sizes in a very simple way. Another way to increase the scale of a network (not illustrated here) is to increase the number of peers hosted on each physical node.

3.2 Experimenting with various configurations

For our experiments, we use the Distributed ASCI Supercomputer 2 (DAS-2) located in The Netherlands, which consists of 5 clusters for a total number of 200 Dual 1-GHz Pentium-III nodes. DAS-2 nodes are managed via the PBS scheduler and are available only through `ssh`/`scp` commands. To test the adequacy of the JDF tool, we measured the time needed to deploy, configure and update a JUXMEM service on a variable number of nodes using an optimized version of JDF. The JUXMEM configuration for this experiment consists in 1 cluster group made up of different numbers of providers; each physical node hosts a single peer.

As expected, Figure 2 shows that the deployment time using JDF is linear with the number of nodes, e.g. it takes 19.6 s for 16 nodes, 38.6 s for 32 nodes. This is due to a loop executed on the control node, which runs `scp`/`ssh` commands in order to deploy JUXMEM and all its dependencies on each node. These high times are explained by the size of the compressed archive that needs to be deployed, which includes all libraries for JXTA, JDF and JUXMEM (4 MB). Of course, this step is required only once per node, so its cost can be "shared" by a sequence of experiments. When modifying a feature of a P2P service an option of JDF can be used to update on each node only modified files. Consequently, the

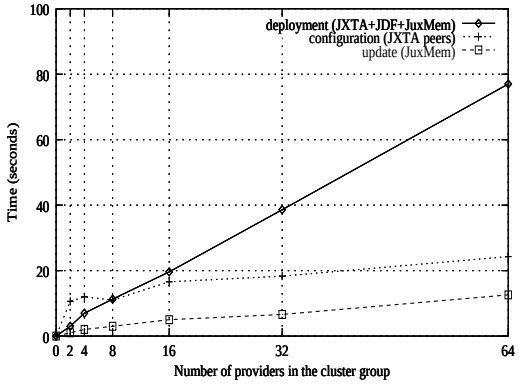

Figure 2: Time needed to deploy, configure, and update the JUXMEM service on various network sizes

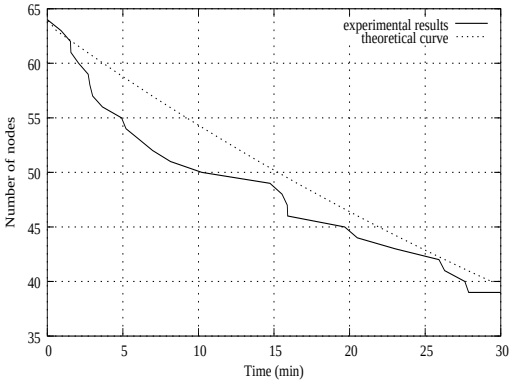

Figure 3: Number of remaining nodes in an experiment with a MTBF set to 1 minute

time needed for instance to transmit modifications on JUXMEM's consistency protocols is significantly lower, since the size of JUXMEM's `jar` file is around 100 KB.

Note also that, in order to reach larger scales, a grid architecture can be emulated by hosting several peers per node. According to our measurements, this has no significant impact on the deployment time, since the only important factor for this step is the number of *physical* nodes.

As seen in section 2.2, using a notion of profile, JDF describes in a concise way what types of JXTA peers are requested, how they are interconnected, etc. The configuration step consists in 2 phases: 1) based on this description, generate on each node and for each peer the JXTA configuration file; 2) update on each peer this previously created file with its list of rendezvous peers. Figure 2 shows that the cost of the configuration step slowly and linearly increases with the number of nodes.

Finally, once the configuration is done, JUXMEM is started by invoking a JVM for each peer of the configuration.

3.3 Experimenting with various volatility conditions

Volatility is an important feature of the P2P model. Being able to control it precisely allows to run tests with various volatility conditions. For instance, the MTBF (Mean Time Between Failures) of each cluster may be a significant test parameter. Some peers may have a high probability of failure, while others may be almost stable. A schedule of failures can be preliminarily computed using (empirical) statistical laws and then used in tests by JDF to

simulate faulty peers. The results collected by JDF then allow to check which peer went down, and when. We are developing a set of tools that generate a configuration file with various volatility-related parameters (e.g. the global MTBF of a JUXMEM network) which are given as an input to JDF, in order to control peer uptimes.

Figure 3 shows the results of a sample experiment on 64 nodes with a configuration file generated to provide a MTBF of 1 minute: the uptime for each node follows an exponential distribution with a rate parameter of $\frac{1}{64}$. In a laps of 30 minutes, 25 peers are killed. The difference between the theoretical and the experimental figures is mainly due to the statistical flows (but also to the fact that all the nodes do not exactly start at the same time and that all nodes clocks are not perfectly-well synchronized).

One of the JUXMEM's goals is to serve as an experimental platform for various failure detection techniques (e.g. ping or heartbeats) and various replication policies (e.g. active or passive). We plan to use JDF in order to tune parameters like the delays between heartbeats, or the replication degree. Short delays favor system reactivity and reliability; yet, they stress the network and non-faulty, but slow nodes may be incorrectly detected as faulty. On the other hand, the longer the delays between heartbeats, the higher the needed replication degree.

The failure detection policy and the replication policy have to be adapted to the system state and to the application behavior. We plan to implement dynamic adaptive mechanisms into JUXMEM. Thanks to JDF, we are able to specify test conditions while varying only a subset of the parameters at a time. It is then possible to identify classes of applications and system states (number of faults, network load, etc.), and adapt fault tolerance mechanisms (failure detection and replication strategies) according to them.

4 Conclusion

Validating P2P systems at a large-scale is currently a major challenge. Simulation is nowadays most-widely used, since it leads to reproducible results. However, the significance of these results is limited, because simulators rely on a simplified model of reality. More complex validation approaches based on emulation and execution on "real" large-scale testbeds (e.g. grids) do not have this drawback. However, they leave the deployment at the user's charge, which is a major obstacle in practice.

This paper shows that the *JXTA Distributed Framework* (JDF) provides an adequate basis to overcome this difficulty. We have illustrated how we customized it in order to configure and control the deployment of our JXTA-based data sharing service at a various scales going up to 64 physical nodes. We have also enhanced JDF in order to be able to

control the volatility conditions during large-scale tests. Some preliminary performance measurements for the basic operations are given.

Further enhancements can be considered for JDF. A hierarchical, tree-like scheme for the `ssh`/`rsh` commands could be used to balance the load of copying files from the control node to other nodes, in the lines of [10]. We plan to integrate a synchronization mechanism for peers to support more complex distributed tests. Another extension will consist in integrating more sophisticated analysis features into JDF. The final goal is to have a rich generic tool allowing to deploy, configure, control and analyze large-scale distributed experiments on a *federation of clusters* from a single control node. Such a tool could prove very helpful for the validation of JXTA-based P2P services, but the approach can be easily generalized to other large-scale P2P environments.

Acknowledgments

The authors thank Thilo Kielmann's group from the Vrije Universiteit for making available to us the DAS-2 cluster.

References

[1] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann, André Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf, and Ian Taylor. Enabling applications on the grid: A GridLab overview. *International Journal of High Performance Computing Applications*, 17(4):449–466, 2003.

[2] Gabriel Antoniu, Luc Bougé, and Mathieu Jan. JuxMem: Weaving together the P2P and DSM paradigms to enable a Grid Data-sharing Service. Research Report RR-5082, INRIA, IRISA, Rennes, France, January 2004. To appear in the Kluwer Journal of Supercomputing.

[3] A Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC. The ns manual (formerly ns notes and documentation). `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf`, 2003.

[4] Mark Carson and Darrin Santay. NIST Net - a Linux-based network emulation tool. 2004. To appear in special issue of Computer Communication Review.

[5] Henry Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, pages 430–441, Brisbane, Australia, 2001.

[6] Frank Dabek, Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 202–215, Chateau Lake Louise, Banff, Alberta, Canada, October 2001.

[7] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 314–329, Bolton Landing, NY, October 2003. ACM Press.

[8] Sam Joseph. P2P metadata search layers. In *Second International Workshop on Agents and Peer-to-Peer Computing (AP2PC'2003)*, number 2872 in Lecture Notes in Computer Science, Bologna, Italy, July 2003. Springer-Verlag.

[9] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. OceanStore: An architecture for global-scale persistent storage. In *9th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS 2000)*, number 2218 in Lecture Notes in Computer Science, pages 190–201, Cambridge, MA, November 2000. Springer-Verlag.

[10] C. Martin and O. Richard. Parallel launcher for clusters of PC. In London Imperial College Press, editor, *Parallel Computing (ParCo 2001)*, pages 473–480, Naples, Italy, September 2001. World Scientific.

[11] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, March 2002. available at `http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf`.

[12] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pages 31–44, Boston, MA, December 2002.

[13] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: the OceanStore prototype. In *2nd USENIX Conference on File and Storage Technologies (FAST '03)*, California, CA, USA, March 2003.

[14] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. Technical Report CSD-03-1299, UC Berkeley, December 2003. Available at `http://oceanstore.cs.berkeley.edu/publications/papers/`.

[15] Luigi Rizzo. Dummynet and forward error correction. In *1998 USENIX Annual Technical Conference*, New Orleans, LA, 1998. FREENIX track.

[16] Timothy Roscoe. PlanetLab: an open community testbed for planetary-scale services. `http://www.planet-lab.org/pubs/2003-04-24-IntelITPlanetLab.pdf`, April 2003.

[17] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 188–201, Chateau Lake Louise, Banff, Alberta, Canada, October 2001.

[18] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. In *5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 271–284, Boston, MA, 2002.

[19] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, pages 255–270, Boston, MA, 2002.

[20] The DIET project: Distributed interactive engineering toolbox. `http://graal.ens-lyon.fr/~diet/`, 2001.

[21] JXTA Distributed Framework. `http://jdf.jxta.org/`, 2003.

[22] The JXTA project. `http://www.jxta.org/`, 2001.

[23] The Portable Batch System. `http://www.openpbs.org/`, 1990.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifi que,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399