A Large-Scale Distributed Environment for Peer-to-Peer Services *

June 2010
Technical Report tr-ri-10-317

Joachim Gehweiler Friedhelm Meyer auf der Heide Ulf-Peter Schroeder

Heinz Nixdorf Institute and Computer Science Department
Paderborn University, 33095 Paderborn, Germany
{joge, fmadh, ups}@uni-paderborn.de

Abstract

We present a large-scale distributed environment for Peer-to-Peer (P2P) software. It enables developers to run and debug their P2P software written in Java using the JXTA framework with minimal effort: Via a web frontend to the central server they can easily setup a JXTA P2P network on a large number of computers at different physical locations and upload their code. By only one mouse click they can trigger an automated configuration, start / stop / reset their test scenario, and view debug output. For this purpose the central server holds the complete configuration of the environment, and automatically controls, configures, and updates all computers of the environment. Except for the server, no dedicated hardware is required; rather, our environment can be installed on existing Windows and Linux computers, where it runs with lowered priority in order not to disturb other users / processes. There is no prescribed limit on the number of computers being part of the environment; its current setup for the EU FP6-IST project AEOLUS, for example, consists of more than 100 computers at 13 different locations in Europe. By virtualization the number of peers can be increased by a factor of up to 10. Further computers, which, e.g., may act as a frontend to the P2P software or as a gateway to special hardware such as sensor networks, can easily be integrated. Finally, our environment is secured against both external attacks over the Internet and internal attacks via hijacked accounts or malfunctioning software.

Keywords Large-Scale Distributed Environment, Peer-to-Peer Services, Java, JXTA

1. Introduction

We present a large-scale distributed environment, which enables developers to run and debug P2P software with minimal effort. It was originally developed for testing the overlay computing platform developed within the EU FP6-IST project "Algorithmic Principles for Building Efficient Overlay Computers" (AEOLUS). Its current setup within the AEOLUS project at `http://aeolus.cs.uni-paderborn.de/` consists of more than 100 computers (and 1000 virtual nodes) located at 13 different European universities and research institutes. However, its architecture is kept so general and simple that it is suitable for running and debugging any (P2P) services implemented in Java using the JXTA framework [1, 7] with minimal effort. Developers can request a login for the existing installation or easily setup an own installation. Via a web frontend they can upload their code and (possibly parameterized) configuration files, which are then automatically deployed to a (large) number of computers (at possibly different physical locations), which are configured as a JXTA P2P network. Via the web interface the develop-

* Partially supported by the EU within FP6-IST-2004-15964 (AEOLUS).

ers can easily control their software and view its output for debugging purposes. Also, they can easily connect further computers to the environment, which, e.g., act as a frontend to their software, as an interface to other software, are equipped with special hardware, or work as a bridge to other hardware such as wireless sensor networks, for example.

Our approach differs from PlanetLab [2, 5] in that it provides an environment for P2P software written in Java using the JXTA framework, whereas PlanetLab simply provides a network of virtual machines to the developer, i.e., PlanetLab is a more general tool, allowing to run and debug almost any kind of distributed software; but, at the same time, such a general approach involves quite some work to setup and configure everything properly. For the purpose of developing P2P software in Java using the well-established JXTA library we provide a comfortable ready-to-use tool, i.e., our approach significantly simplifies and speeds up the developers' work. Furthermore, our environment can be installed on existing Windows and Linux computers, where it runs with lowered priority in order not to disturb other users / processes, whereas PlanetLab must be installed on additional hardware dedicated solely for this purpose.

2. Architecture

The architecture of our system is depicted in Fig. 1. Its main components are:

- the server which includes a web server, management software, and a database with the system configuration;
- edge peers: these are essentially the computing nodes donated to the system by several different partners;
- rendezvous peers: these nodes support special functionalities of JXTA, which enable the edge peers to locate each other and communicate;
- user nodes: through these nodes, developers access the system. Beside using it through the web interface and JXTA shell, specialized edge peers can be employed as user frontends or interfaces to other software outside the system.

The server stores the configuration of the whole system in its database; the configuration does not only include setup parameters of all the edge peers, but also the software and all testing parameters of all registered developers. This ensures that the system is able to automatically recover the configuration of edge and rendezvous peers after crashes, reboots, or even reinstallations of the operation system. Additionally, it eases developers' lives as they only have to upload and configure their software once through the web interface of the server; the server then automatically deploys the software to edge peers, configures, and controls it, and collects the output (stdout + stderr) on demand. The communication between the server and the edge / rendezvous peers is secured by SSH (for Linux peers) resp. a user-defined protocol (for Windows peers). Peers, which are part of the same testing scenario, communicate using JXTA; beside the peers assigned to the developer's test case, he / she can also add one or more own edge peers—either a JXTA shell or specialized edge peers which work as a user frontend or interface to other software outside the system.

The rendezvous peers are a subset automatically selected out of the edge peers by the server. The edge peers are computers running Windows or Linux, donated by different project partners; these computers need not be exclusively dedicated to our system as the software to test is executed on them with lowered priority in order not to disturb the owner of the computer in his activities.

An Example for specialized edge peers is given by gateways to special hardware. In the AEOLUS project, e.g., 289 wireless sensors of heterogeneous types located at 11 different sites have been connected to the system via gateway edge peers.

3. Requirements

For the server a Linux machine is required. In the particular case of the current setup within AEOLUS, a multiprocessor machine with a gigabit backbone connection and a softraid system running the Ubuntu Server Edition is used, hosted by the University of Paderborn.

In order to setup your own installation, the main required software components (which are usually part of every modern Linux distribution) are the following: ssh, cron, bash, perl, gcc, libc, Java (Sun JDK version 6.0 or later), MySQL database, Apache web server, Apache PHP plugin. Instructions on how to properly configure these software components are usually shipped together with them and can additionally be found on the distributors' particular web pages. Spe-

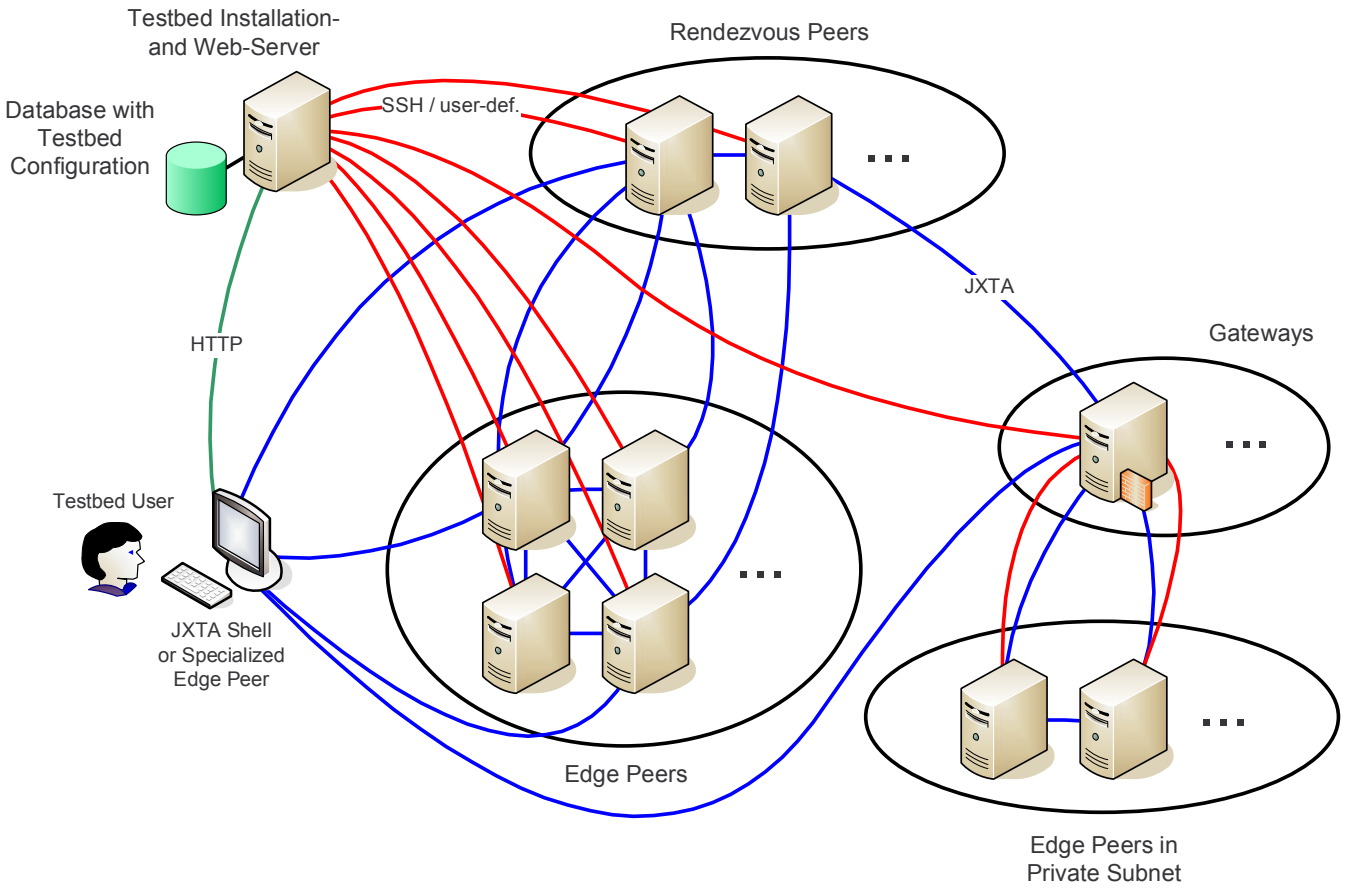

Figure 1. The System / Network Architecture.

cific instructions on how to install the server are included in the distribution.

Once the server is setup, edge peers can be added to the system. These can be either Windows or Linux computers. In order to comply with the strict security regulations of different networks, the requirements for integrating computers into the system are kept as low as possible. In particular, the requirements for Windows nodes are as follows: Windows 2000/XP/2003/Vista/7, a public IP address, open TCP port 2022 for system management and ports 9700-9799 for JXTA (these default values can be changed in case these ports are already occupied by other software). The installation procedure using the Windows installer is very simple: during the setup you are asked to enter your login and password, and you may change the default values for the TCP ports; the rest is done automatically.

For Linux nodes, the requirements are as follows: Kernel 2.6.x, ssh, bash, perl, tar, gzip (typically present on every distribution), a public IP address (or a gateway), open TCP port 22 (SSH) for system management and ports 9700-9799 for JXTA. Since an automated installer is not yet available, the installation procedure involves the manual setup of a dedicated ordinary user account (without root privileges); you need to provide the IP address, ports (when deviating from the default values), and the login to the testbed administrator and install an SSH key for automated login.

We remark that there is no need to install Java since our software comes with its own Java installation in order to assure that all edge peers are running the same Java version and have all required libraries installed.

4. Security Aspects

The system is secured against different types of attacks. On the one hand, to prevent outside attackers from gaining unauthorized access, a multilevel security model is applied. First, all management / installation / configuration traffic with the Linux nodes is tunneled using SSH with DSA keys (or RSA keys if DSA is not supported) of a reasonable strength; second, our software completely runs in user space, i.e., even if a con-

nection would be hijacked, nothing outside this dedicated user account could be damaged. For the Windows nodes, all connections to the management port from another IP address than the server are dropped; second, a fixed set of reactive commands is defined, i.e., a command can only be triggered from outside without any parameters and the local process then collects the required options itself (thus, no invalid options can be used).

On the other hand, the system is also secured against internal attacks. In case, for example, a user does not properly protect his password, an attacker can neither hack other user accounts nor hijack any computers of the system. This is due to the security policy of the Java Sandbox which is configured in such a way that the uploaded code can only access files in the local folder and only open network sockets on ports above 1024 (thus, computers "infected" by malicious uploaded code can only produce a high CPU and network load in the worst case).

5. Testing a P2P Service

As already mentioned in the introduction, P2P software to be tested has to be written in Java using the JXTA library. In order to both provide a uniform interface to our system and simplify developer's lives, our API already includes basic implementations of JXTA P2P services and service discoveries.

To execute and test a particular piece of software, the user typically performs the following steps:

1. Allocate a number of computing nodes;
2. configure them as edge peers of a JXTA P2P network;
3. upload his / her code to test;
4. run / debug the software.

To ease this procedure as much as possible, most of these tasks are done automatically. The server holds the complete configuration of all test setups of all users and automatically installs and configures the computing nodes. Moreover, it automatically recovers computers which were temporarily unavailable or even suffered a loss of data.

Computing nodes are allocated via the web interface of the server, and afterwards the software to test (and all its configuration files) are uploaded. The server then automatically delivers all components of the software to the allocated computing nodes, configures them properly, and runs the software. For debugging purposes, the output of all peers can be viewed via the web interface.

In order to allow scalability tests with a huge number of computing entities, our system is able to simulate up to 10 virtual nodes per physical computing node; the number of requested virtual nodes has to be specified during the allocation step.

In order not to interfere with other users when reconfiguring or restarting the P2P network (or when your software to test appears to be buggy), a separate JXTA network is setup for each test case.

As not all P2P applications consist of totally homogeneous peers, the code and configuration files to be uploaded to the edge peers can be individually configured on a peer basis. Furthermore, there is a set of scripted constants supported, e.g., `__IP__` which is automatically replaced by the edge peer's IP address in all configuration files.

Some applications require peers outside our system, e.g., as a user frontend or interface to other software outside our system. Our API provides a framework to implement a specialized edge peer for such a case. To connect these peers to a JXTA network in our system, the location and port of the rendezvous peer is displayed in the web interface.

As an example for a use case of our system we refer to the Paderborn University BSP-based Web Computing (PUB-Web) library [3, 4], which is a P2P parallel volunteer computing system. During the development of novel distributed load balancing strategies, e.g. [6], our environment can be used to support the developers in testing, debugging, and evaluating their algorithms: After uploading the code and the configuration files of the PUB-Web library, they can easily configure a P2P network, start / reset the PUB-Web network, and view the debug output through the web interface or our environment. In order to actually run a parallel program, a specialized edge peer is required because the developers need access to the graphical user interface at one of the peers in the PUB-Web network. As a very simple example think of a parallel program rendering the Mandelbrot image: A user behind a PUB-Web peer simulated on a specialized edge peer enters the parameters of the Mandelbrot image to draw (cf. Fig. 2). The PUB-Web software then requests a number of computing nodes from a PUB-Web supernode discovered through a lookup in our JXTA network. Then the PUB-

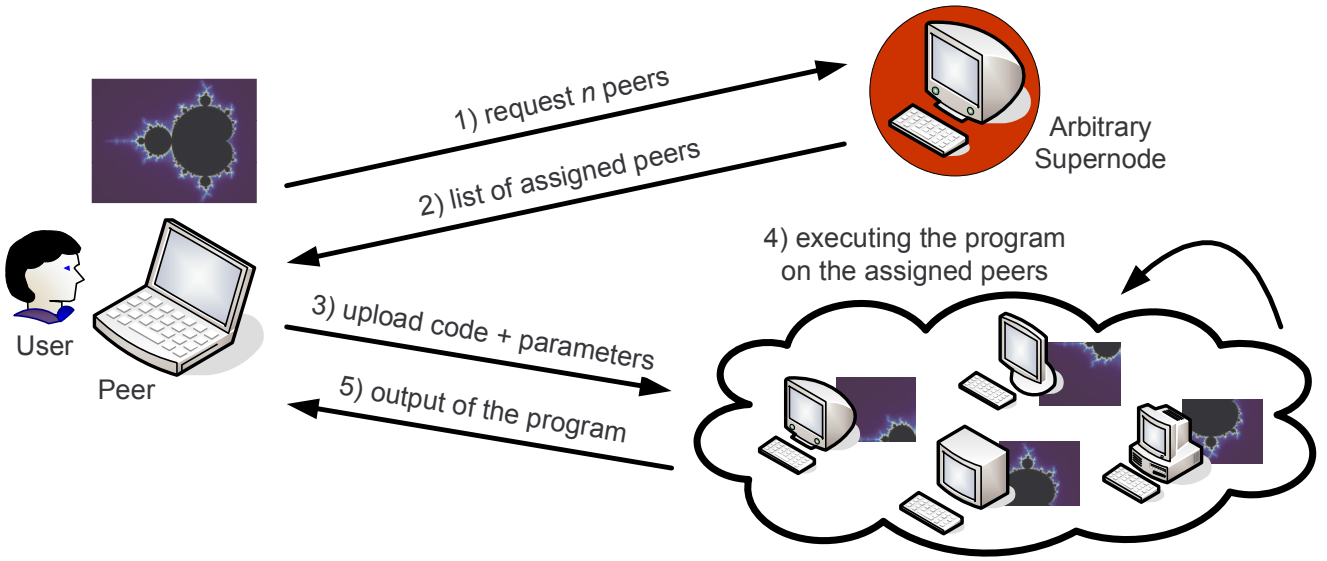

Figure 2. Example for a Use Case.

Web software on the specialized edge peer uploads the parallel Mandelbrot code and the parameters to the assigned subset of our JXTA network, where the Mandelbrot code is executed, and finally composes the image out of the output sent back from the peers.

Setting up such a testing scenario in a distributed environment without our system would require a lot of manual (remote) configuration work. But via the web interface of our environment it is just one click to upload code or to start / stop / reset the testing scenario, which significantly improves the workflow of a developer.

6. Conclusion

We have presented a large-scale distributed environment for P2P software written in Java using the JXTA framework. Via a web frontend to a central server, which holds the complete configuration and automatically controls, configures, and updates the computers of the environment, developers can test their software on a big number of computers at different physical locations with minimal effort.

Future improvements of the environment include not only a fully automated setup for the Linux case; in order to support dynamic IP addresses, the network interface could be monitored by our software (currently the IP address is only determined on the computers in our system during the startup of our software). Also, employing an auditing technique is conceivable as an additional security feature (traceability).

References

[1] The JXTA community. Website: `https://jxta.dev.java.net/`.

[2] PlanetLab. Website: `http://www.planet-lab.org/`.

[3] The Paderborn University BSP-based Web Computing (PUB-Web) library. Website: `http://pubweb.cs.uni-paderborn.de/`.

[4] O. Bonorden, J. Gehweiler, and F. Meyer auf der Heide. A web computing environment for parallel algorithms in Java. In *Proceeedings of International Conference on Parallel Processing and Applied Mathematics (PPAM)*, Poznan, Poland, September 2005.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003. ISSN 0146-4833.

[6] J. Gehweiler and G. Schomaker. Distributed load balancing in heterogeneous peer-to-peer networks for web computing libraries. In *Proceeedings of 10th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 51–58, Torremolinos, Malaga, Spain, October 2006.

[7] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5:88–95, 2001. ISSN 1089-7801.