

# A Peer-to-Peer Architecture for Automatic Software Package Installation on Heterogeneous Clusters

**Diego Kreutz**

Universidade Federal do Pampa, CTA  
Alegrete, Brazil, 97540-000  
kreutz@inf.ufsm.br

and

**Marcelo Veiga Neves, Elton Nicoletti Mathias**

Universidade Federal do Rio Grande do Sul, II  
Porto Alegre, Brazil, 91501-970  
{mvneves, enmathias}@inf.ufrgs.br

and

**Tiago Scheid, Andrea Schwertner Charão**

Universidade Federal de Santa Maria, LSC  
Santa Maria, Brazil, 97105-900  
{scheid, andrea}@inf.ufsm.br

## **Abstract**

This article presents a hybrid peer-to-peer architecture for flexible automatic installation of software packages on Linux-based computer clusters. In this architecture, software updates are performed without intervention of a centralized server and can be carried out during idle cycles of the potentially heterogeneous cluster nodes. To cope with hardware and software heterogeneity, cluster nodes are organized into groups based on their similarities. Flexibility is achieved by means of two installation protocols which use different communication schemes and harness native package managers from Linux distributions to perform local installations. In order to validate our approach, an automatic installer named Clumpt has been developed based on this architecture. Experimental results using Clumpt show that the peer-to-peer architecture can provide scalability and robustness to the software installation process.

**Keywords:** Peer-to-Peer, clusters, software heterogeneity, robustness, scalability.

## 1 Introduction

Software installation is a recurrent subject in the cluster computing area. As cluster systems grow in size and complexity, installing, configuring and updating cluster software become a challenging task which have been tackled in different ways by a number of cluster management tools. This is the case of systems such as FAI [11], SystemImager [8], LCFG [1] and SCMS [16], which are targeted to automatic software installation on Linux based clusters. Existing automatic installers usually adopt a client-server architecture to perform either a package-based installation (where software packages are transferred from server to client and installed using some distribution-dependent tool) or a disk image-based installation (where complete or partial system images are stored on the server and copied onto the clients). Due to the server-centered model, these solutions are highly susceptible to bottlenecks and failures. Also, using these tools may interfere on the performance of user applications because no resource usage information is considered.

As an alternative to the traditional client-server model, peer-to-peer (P2P) architectures became very popular during the last few years. This approach may improve scalability and robustness through decentralization. There are numerous applications of peer-to-peer networks for distributed computing, collaboration or file sharing [2], but to our knowledge, such approach have not yet been deployed in cluster installation tools.

In this paper, we explore essential characteristics of peer-to-peer models [13], namely decentralization and dynamism, as the basis for a software architecture targeted to package-based software installation on computer clusters. The main goals of this architecture are robustness, achieved through its inherent self-organization property; scalability, in face of the decentralized approach, and low overhead, achieved through idle time usage. The paper is organized as follows: section 2 presents an overview of the peer-to-peer architecture as well as its main design goals. The architectural components and their overall operation are described in section 3, while section 4 presents an automatic installer which implements our peer-to-peer architecture. section 5 describes an experimental evaluation of such installer and section 6 discuss some related work.

## 2 Architecture Overview

The architecture is based on a hybrid P2P model [15], where each cluster node is a member of the peer-to-peer network. In our hybrid approach, peers store and retrieve configuration information from a distributed (and potentially replicated) database. During software installation and updates, peers interact without intervention of a centralized server. Each peer stores information of previous software installations as well as software packages installed locally. This allows each peer to act as both a client and a server which is able to multicast installation notifications and respond to package requests.

Figure 1 presents an overall view of the peer-to-peer architecture. As can be seen from the illustration, peers are organized into groups. The goal of such organization is to cope with heterogeneous cluster environments that are increasingly popular [5, 3, 6, 7]. Every cluster node belongs to at least one group (DefaultGroup) and can be part of as many groups as necessary, as defined by the system administrator. Peer groups can be used to arrange nodes according to hardware similarity, network organization and/or software configuration requirements.

The database stores peer groups configuration and software installation history. Its main purpose is to provide a global, consistent view of the whole cluster software state. Peers retrieve information from the database during initialization or when recovering from failures. Software packages are not stored in the database, but rather distributed among networked peers, as mentioned before. With this approach, we aim to avoid a central point of failure during maintenance operations and prevent network bottlenecks that can arise when employing a single package repository server. Indeed, centralized solutions frequently used in clusters, as for example a centralized NFS (Network File System) server, can lead to network congestion and decrease performance and scalability of software installation processes, as well as interfere with message-passing applications running on the cluster.

The database can be replicated or distributed across the cluster administrative domain, allowing peers to access it through different network paths. Database replication may improve robustness, while data distribution may increase performance and availability, avoiding bottlenecks during peer initialization and failure recovery.

Another important characteristic of our architecture is that it takes advantage of idle periods to perform

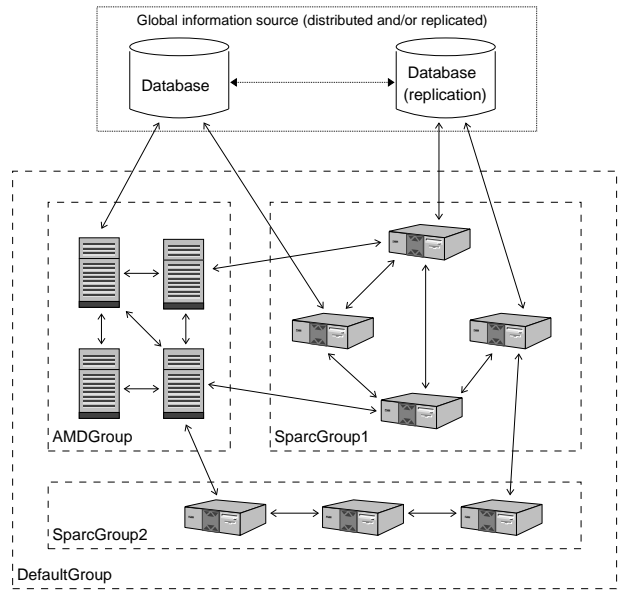


Figure 1: Architecture overview.

package installation on cluster nodes. Each peer is responsible for monitoring local system load metrics, in order to determine whether it is able to carry out installation protocols and operations. By taking load metrics dynamically into account, we aim to keep installation overhead as low as possible, while providing system administrators with a dynamic and flexible solution for cluster software updates.

### 3 Components and Operation

This section describes the main architectural elements, namely peer components and the global database, as well as the overall operation of the peer-to-peer architecture.

#### 3.1 Peer Components

Figure 2 presents the organization of each node belonging to the peer-to-peer network. Each component is described below in more detail.

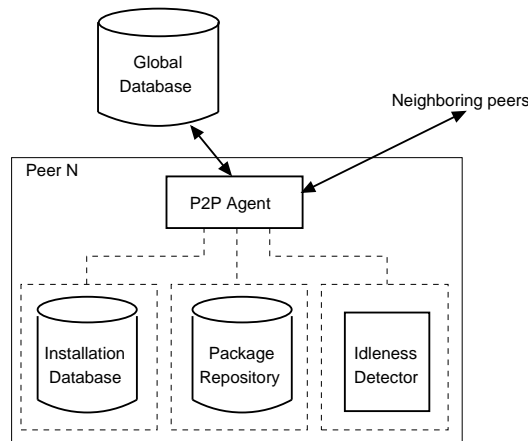


Figure 2: Peer components.

**Package Repository** This component stores software packages used for local installations. Such packages may be sent to other peers upon request. Packages remain stored after installation and may be in any format supported by the operating system. In other words, a copy of the received packages is left in the local repository. This copy is then used for seeding the packages to neighbour peers.

**Installation Database** This element registers information on every software installation assigned and performed on the local system. This comprises package identification and location in the repository, local package manager options for performing installation, and target peer groups. As different peers may belong to distinct groups, they may have different installation entries in this database. Moreover, as peers may be out of the network for a certain period (for hardware maintenance, for example), members of a given group may also have different entries at a given time. However, as an installation proceeds for this group, all corresponding entries converge to a consistent state, where all peers have completed installation and updated their local databases.

**Idleness Detector** This component is responsible for gathering local load metrics (CPU, memory and network usage, among others) and determining idle periods that can be used to perform software updates. A peer is considered idle when their load metrics remain below a given threshold for a certain amount of time. Intrusiveness is a major issue for this component. To address this issue while ensuring flexibility, there are global configuration options to setup the frequency of metric measures as well as thresholds and time intervals for determining idle state.

**P2P Agent** This agent coordinates all peer components presented above. It performs queries and updates on the global database and on the local package repository and installation database. It is also responsible for P2P interactions among cluster nodes. To this end, this component implements a collection of protocols that define how peers operate within the architecture. section 3.3 describes these protocols as well as the overall operation of each peer.

### 3.2 Database

As mentioned before, the database stores information about groups, machines and installed packages history. Database modifications happen only when the system is configured and every time a new software package is installed. Queries are executed every time a peer is initialized, when it has locally processed a new installation package and when it recovers from failures.

### 3.3 Peer Operation

Every peer starts by performing some initial queries to neighboring peers and to the global database, as part of an initialization protocol. After initialization, each peer is able to receive and propagate installation notifications and to retrieve and actually install software packages. To this end, two protocols are provided: a load-aware installation protocol and an immediate installation protocol. The former is only carried out when a peer is considered idle, reducing intrusiveness on running applications. The further is useful in cases where software installation have a high priority and need to be performed as soon as possible. A group update protocol is also part of the architecture, as a means of dynamically manage peer groups. All these protocols are coordinated by the P2P agent. The following paragraphs present these protocols in more detail.

**Initialization Protocol** Every peer starts by retrieving basic information from the global database: cluster node addresses and their group organization, as well as its installation history (previous package installations completed on the peer). As soon as such information is received, the peer is able to join the P2P network. It then synchronizes its local installation database with neighboring peers belonging to the same groups, in order to identify new packages to be installed. This allows machines that have been off for a certain period to reach a consistent software installation state within its peer groups.

The installation history is stored locally and globally. In doing so, the system is able to provide some kind of availability mechanism. Further improvements will include checksumming local and global data copies. This is going to enable node's local data usage, just proceeding a checksumming with global stored data for consistence and integrity verification.

**Load-aware Installation Protocol** This protocol is used to carry out package installations while the target nodes are considered idle. Any peer owning a software package for installation may initiate this protocol. It begins by notifying neighboring peers belonging to a given group about the new package to install. The target notification peers are randomly chosen from a subset of the target peer group (the subset size is a global configuration option). When a peer receives a notification message, it starts a communication channel to request the new package from its neighboring peers, starting from the notifying peer. When a peer receives a package request, it searches its package repository for a local match. If the package is found, it is sent to the requesting peer, along with all related information stored on the local installation database. When the package is locally available to the requesting peer, it performs actual package installation and updates both local and global databases. It also propagates the notification to other randomly chosen peers remaining within the same group. It is worth noting that package transfer and installation, which are most resource consuming operations, only proceed when the peers are in idle state.

**Immediate Installation Protocol** This protocol provides an alternative to the load-aware installation protocol. Within this protocol, package transfers and installations are carried as soon as possible without taking system load into account. Both installation notifications and package transfers are carried out using a distributed hierarchical communication scheme, initiated from a father node owning the package for installation. Local and global database updates are performed as described previously for the load-aware protocol.

**Group Update Protocol** This protocol is aimed for automatic and dynamic management of peer groups. Group configuration is initially obtained from the global database, as part of the initialization protocol performed by each peer. As peers may be off during a certain amount of time, the group update protocol is used to propagate information on peers which leave or enter the peer-to-peer network. When a peer detects another peer as being out of service, it notifies all peers belonging to the same group, so they can update their local peer group list. Once a peer is back to the network, it notifies all neighboring peers about it. This protocol avoids delaying installation protocols over the whole group when some of its nodes are out of the network. It is worth noting that this protocol itself does not change the peer group configuration as defined by the system administrator.

## 4 Architecture Implementation: Clumpt Automatic Installer

In order to validate the peer-to-peer architecture presented in the previous sections, we developed an automatic installer named Clumpt. This tool comprises three related programs: `clumptd`, which runs as a daemon process on each peer, `clumptconf`, which is targeted to peer group administration, and `clumpt`, which is the main package installation interface. Clumpt provides two installation modes: the default installation mode, which implements the load-aware installation protocol, and the urgent installation mode, which uses the immediate installation protocol. section 4.1 presents some implementation decisions for this automatic installer, while section 4.2 give an overview of its user interface.

### 4.1 Peer Components and Database Implementation

All peer components are encapsulated within the `clumptd` program. The package repository uses the local file system infrastructure, while the installation database is implemented as a collection of tables stored on local memory. The idleness detector uses the `liblproc` library [14] for collecting load metrics. This library can obtain metric measures from cluster monitoring tools like Ganglia[12], SCMS[16], PCP[9] and Parmon[4], which may already be running on the cluster. All installation protocols use TCP for data transferring.

The global database is stored in an OpenLDAP [17] server. This server has been chosen because it is widely used as authentication mechanism on network servers and clusters. OpenLDAP also allows transparent replication and distribution of the database, which can improve scalability and robustness characteristics.

### 4.2 User Interface

Figure 3 shows the main use cases for `clumpt` and `clumptconf` programs. The following paragraphs present an overview of these programs and their implementation.

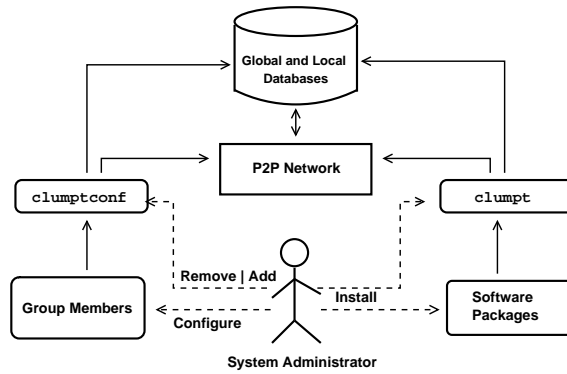


Figure 3: User interaction with Clumplt.

**clumpltconf program** This program is used to manage peer groups through the group update protocol, as a means of manually removing (or adding) nodes from the peer-to-peer network. It can also be used to dynamically manage peer groups through read and write operations onto the group configuration file, which are then propagated through the group update protocol. The group configuration file is text-based file which could also be manually edited, although in this case the group update protocol would not automatically take place.

Figure 4 presents a configuration file example with two groups. The first group, LSC, comprises two machines named `host1` and `host2`. The second group, named PAPLE, is composed of twelve machines, which are represented by an IP range (from `10.0.0.1` to `10.0.0.12`). This program uses the Confuse library [10] to manipulate group and machine representation formats.

```

group{
    name = "LSC"
    description = "Room 214 - CT/UFSM"
    machines = {
        host01,
        host02
    }
}
group{
    name = "PAPLE"
    description = "PAPLE Cluster"
    machines = {10.0.0.1-12}
}
  
```

Figure 4: Configuration file example.

**clumplt program** This program is used to submit package installations to the peer-to-peer network. Its command-line user interface is similar to most package managers for Linux distributions (`emerge`, `apt-get`, `rpm`, `pkgtool`, etc.). The main command-line arguments are the package location, package manager options and target peer groups, as well as the installation mode to use. For each new package installation, `clumplt` generates a unique sequential installation identifier and registers package information on the global database. It then starts the appropriate installation protocol by activating the local P2P agent, which coordinate the next steps of the installation process. Actual package installation is performed through the native package manager. The overall installation process begins with a few nodes. They can be manually chosen by system administrator or randomly selected by `clumplt` program. In case of random selection, users may specify how many initial nodes will receive the package. `clumplt` program will afterwards send the package data to each chosen peer.

## 5 Evaluation

This section presents an experimental evaluation of Clumplt using two cluster platforms. The first one was used to evaluate the overall performance and scalability of both installation modes provided by Clumplt. The second platform was employed for robustness and intrusiveness evaluation.

### 5.1 Performance and Scalability Evaluation

The first test platform was a 60-node cluster consisting of identical Dell PowerEdge 1750 machines interconnected through a Gigabit Ethernet network. Each node had two Intel Xeon processors running at 2.4 GHz with 2 GB RAM memory. We used both default and urgent installation modes to perform various software installations over this cluster, varying the package sizes and the number of nodes of the target installation group. We also perform the same package installations using a centralized NFS server. When performing installations using the default mode, we turned off the idleness detector on each peer and configured every node to be always available (idle) to carry out package installations. This was done in order to focus on communication performance of the installation protocols. The intrusiveness of the idle detection mechanism is assessed in the next section.

Figure 5 presents the elapsed time for transmitting a 30MB package (a Linux kernel package) over different peer group sizes ranging from 1 to 60 nodes. Figure 6 shows transmission times for different package sizes (from 50MB to 1000MB) over a single group comprising the whole 60-node cluster.

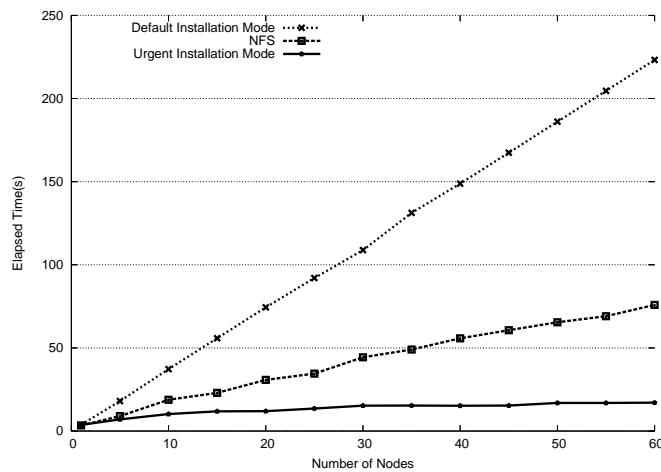


Figure 5: Transmission time for a 30MB package varying the number of nodes.

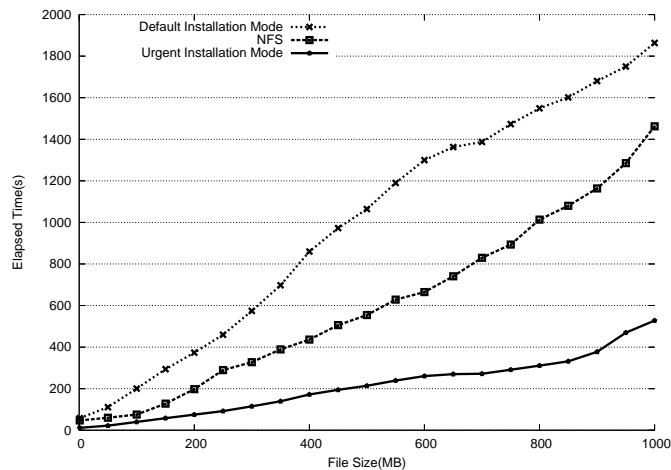


Figure 6: Transmission time for 60 nodes varying the package size.

By analyzing figure 5, one observes that the urgent installation mode performed better than NFS and the default installation mode for all group sizes (number of nodes) considered. The former also scaled better than the further installation schemes, which is evidenced by a smoother growth of the transmission times for the urgent installation mode, when increasing the number of nodes. The behavior of all installation approaches is sustained when the package size increases, as can be observed in figure 6. However, the default installation mode may be preferred due to its capability of taking node idleness into account.

## 5.2 Intrusiveness and Robustness Evaluation

The second experiment was carried out over 5 machines with a 2.4 GHz Intel Pentium IV processor and 512MB of RAM memory. These machines were connected by a Fast Ethernet network.

In order to evaluate the overhead of running the `clumpton` daemon on these machines, we began by monitoring CPU, memory and network usage for this process alone, without submitting any software installation to the peer-to-peer network. For this test, we configured the system to perform database synchronization every 60 seconds. The whole test was carried out during a 12 hours period. As a result, we obtained an average of 0.00000013 % of CPU time usage, 0.0028 % of memory usage and 0.0000039 MBits/s of network bandwidth usage.

In order to evaluate the robustness of the software installation process, we forced a failure by turning off one of the nodes for 20 minutes<sup>1</sup>. Over this time, we used the `clumpton` program to submit 3 new package installations, totaling 48MB of data. When we turned the node back on, it started the initialization protocol and took around 29 seconds to synchronize its local database and perform the 3 pending installations. On a second test, one node was turned off after receiving a 30MB package for installation. In this case, it took around 15 seconds to complete the interrupted installation when it was turned back on.

## 6 Related Work

There are a number of tools aimed to automatic software installation on clusters or networked computers. Here we discuss three popular automatic installers:

- FAI (Fully Automatic Installer) [11]: this is a package-based installation management tool that can be used for installing cluster software from scratch. FAI has a centralized architecture relying on a main package server. The main drawback of this tool is that it can only be used with Linux systems based on Debian distribution.
- SystemImager [8]: this disk image-based installer is targeted to software installation and update on a group of machines sharing the same hardware of software characteristics. As this tool works with a whole system disk image, it is not restricted to a single operating system distribution. A single server stores different system images, so it is possible to use this tool for installing heterogeneous systems. However, this approach may require large storage spaces.
- LCFG (Local ConFiGuration system) [1]: this is a package-based tool targeted to automatic software installation, configuration and update of large Unix-based systems. A central server stores software in RPM format, as well as configuration information on installed clients. The server notifies client nodes of every configuration changes, so affected clients can retrieve packages and perform local software updates.

An important difference between Clumpton and these tools is that the former has a decentralized architecture. This approach is employed to improve robustness and scalability, and for supporting heterogeneous clusters.

Just as FAI and LCFG, Clumpton is a package-based tool. In comparison with image-based installation, a package-based approach allows easier software installation and update, for there is no need to generate complete system images every time a new software is required. There is also no need for a central image server, which represents a single point of failure and may lead to network bottlenecks.

Clumpton, as well as SystemImager, can be used with different Linux distributions. Besides that, it allows using package management tools which are part of each distribution. This approach relieves Clumpton from the burden of managing package dependencies.

---

<sup>1</sup>We considered only process and system general failures, with may lead to near future updates and/or full recovery (reinstallation of all packages)

## 7 Conclusion

Automatic software installers are valuable tools for managing large-scale cluster systems. A number of such tools already exist and they are usually based on a client-server architecture to distribute software packages or disk images over several cluster nodes for installation. In this paper we presented a software architecture which explores a peer-to-peer approach for automatic software installation on computer clusters. Such architecture provides a solution to cope with heterogeneous clusters and to take into account system load metrics to perform software installations on idle nodes during cluster utilization.

We developed an automatic installer named Clumpt which is based on our peer-to-peer architecture. Experimental results indicate that Clumpt can combine scalability, robustness and low overhead. Such characteristics distinguish Clumpt from existing software installers.

Our peer-to-peer architecture may also apply to software installation on general purpose computer networks. We are currently working on making Clumpt available to third-party system administrators, as a means of improving our experimental evaluation. As future work, we intend to investigate the impact of a pure P2P model as an alternative to our hybrid peer-to-peer approach. Further investigations may also include performance analysis on larger scenarios, like clusters composed by thousands of nodes.

## References

- [1] ANDERSON, P., AND SCOBIE, A. LCFG: The Next Generation. In *UKUUG Winter Conference (2002)*, UKUUG. <http://www.lcfg.org/doc/ukuug2002.pdf>. Last access: may 2006.
- [2] ANDROUTSELLIS-THEOTOKIS, S., AND SPINELLIS, D. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.* 36, 4 (2004), 335–371.
- [3] BOHN, C. A., AND LAMONT, G. B. Load balancing for heterogeneous clusters of PCs. *Future Generation Comp. Syst* 18, 3 (2002), 389–400.
- [4] BUYYA, R. PARMON: a portable and scalable monitoring system for clusters. *Software Practice and Experience* 30, 7 (jun 2000), 723–739.
- [5] CÉRIN, C. Improving parallel execution time of sorting on heterogeneous clusters. In *16th Symposium on Computer Architecture and High Performance Computing (2004)*.
- [6] DEVINE, K. D., BOMAN, E. G., HEAPHY, R. T., HENDRICKSON, B. A., TERESCO, J. D., FAIK, J., FLAHERTY, J. E., AND GERVASIO, L. G. New challenges in dynamic load balancing. *Appl. Numer. Math.* 52, 2-3 (2005), 133–152.
- [7] FAIK, J., TERESCO, J. D., DEVINE, K. D., FLAHERTY, J. E., AND GERVASIO, L. G. A model for resource-aware load balancing on heterogeneous clusters. Tech. Rep. CS-05-01, Williams College Department of Computer Science, 2005.
- [8] FINLEY, B. E. *SystemImager v3.2.0 Manual*. <http://www.systemimager.org/doc/systemimager-manual.pdf>. Last access: may 2006.
- [9] GOODWIN, M. Performance co-pilot web page, 2005. <http://oss.sgi.com/projects/pcp/>. Last access: may 2006.
- [10] HEDENFALK, M. libconfuse web page, 2003. <http://www.nongnu.org/confuse/>. Last access: may 2006.
- [11] LANGE, T. *Fully Automatic Installation (FAI)*. <http://www.informatik.uni-koeln.de/fai>, 1999. Last access: may 2006.
- [12] MASSIE, M., CHUN, B., AND CULLER, D. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing* 30, 7 (2004).
- [13] MILOJICIC, D. S., KALOGERAKI, V., LUKOSE, R., NAGARAJA, K., PRUYNE, J., RICHARD, B., ROLLINS, S., AND XU, Z. Peer-to-peer computing. Tech. Rep. HPL-2002-57R1, Hewlett Packard Laboratories, July 14 2003.

- [14] NEVES, M. V. liblproc, 2005. <http://freshmeat.net/projects/liblproc/>. Last access: may 2006.
- [15] SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing* (2001), pp. 101–102.
- [16] UTHAYOPAS, P., AND RUNGSAWANG, A. SCMS: An extensible cluster management tool for Beowulf cluster. In *Proceedings of Supercomputing'99* (1999), ACM SIGARCH and IEEE.
- [17] ZEILENGA, K. *OpenLDAP Software 2.3 Administrator's Guide*. <http://www.openldap.org/doc/admin23/>. Last access: may 2006.