A scalable file distribution and operating system installation toolkit for clusters

Philippe Augerat[1], Wilfrid Billot[2], Simon Derr[2], Cyrille Martin[3]
[1]INPG, ID Laboratory, Grenoble, France
[2]INRIA, ID Laboratory, Grenoble, France
[3]Bull/INRIA, France

Abstract:

This paper describes a toolkit for optimizing file distribution on a large size infrastructure such as a cluster or a grid of clusters. It provides both the software for operating system installation and file broadcasting based on multicast, chain and several tree-structured algorithms.
It has been used on a 225-node cluster allowing a complete installation of the cluster in just a few minutes and efficient file staging for user's applications. It has also been used to quickly install Linux, Windows 2000 and dual boot systems.

1. Introduction

Large infrastructures such as big clusters, grids of clusters or computer rooms at universities have to be managed in a very automatic, dynamic and efficient way. System administrators need scalable tools to install, administer and monitor the infrastructure while users need scalable tools to program, manage and monitor applications and data.

On one hand scalability means obtaining more power with more computers, on the other hand it means obtaining the same response time regardless of the number of computers. The Ka project deals mainly with the latter problem and aims at designing management tools that scale to thousands of PCs. The main purpose of the Ka project is file distribution in the context of operating system installation (very large file distribution) and day-to-day data broadcasting (small to medium-size file distribution).

The broadcast problem has been studied considerably in the framework of message passing libraries and collective communication operations [15], [16], [17], [18]. Several algorithms based mainly on multicast and unicast trees have been suggested in the literature. Yet the lack of a generic model for "self-made" supercomputers such as grids or network of workstations makes it harder to decide which broadcast algorithm is the best. Our idea, then, was to write, compare and make all these algorithms available through a general library for file distribution.
Numerous tools exist for deploying a set of PC in clusters, intranet networks or University computer rooms. Yet, we think that the Ka management tool is more generic and scalable than most existing software. Moreover, it is completely open source and quite easy to maintain.

The library has been used to install a 225-node cluster. It is also a basis for the design of scalable parallel commands for cluster management and file staging.

2. Operating system installation

The installation of hundreds of PCs excludes the idea of performing manual tasks. The data to be transferred (several gigabytes) also imposes an optimal use of the network. We present now a three-step automatic installation process that performs the complete installation of hundreds of client PCs within fifteen minutes. The first step automatically launches a small operating system on each client. Then a previously installed PC is used to broadcast a file system image to all the client PCs. Finally, a post installation process provides each computer with its own name and characteristics.

a) Initial boot process

A blank PC can boot on its network card (PXE protocol) to obtain information on its name and the operating system that it will use next. The PXE protocol is used to contact DHCP and TFTP servers, which provide the client computer with a boot program, which in turn loads a "higher level" operating system. This installation scheme is used classically to install workstations in a computer room in a university, for instance with the BpBatch software [6]. .

Since BpBatch is not open source we re-implemented a limited set of BpBatch features (read files through TFTP, load and run a Linux kernel, start a computer on its hard disk) in our boot program, thus providing an Open Source cloning system. This program names Ka-boot.

The boot program plus the minimal operating system weigh less than two megabytes. For two hundred PCs, the duration of the initial boot process will be a few seconds. A multicast file transfer protocol could be used to strengthen this part of the installation for thousands of PCs.

Figure 1 summarizes the "client-server" exchanges during the first installation stage including the "PXE" and "Ka-boot" stages.

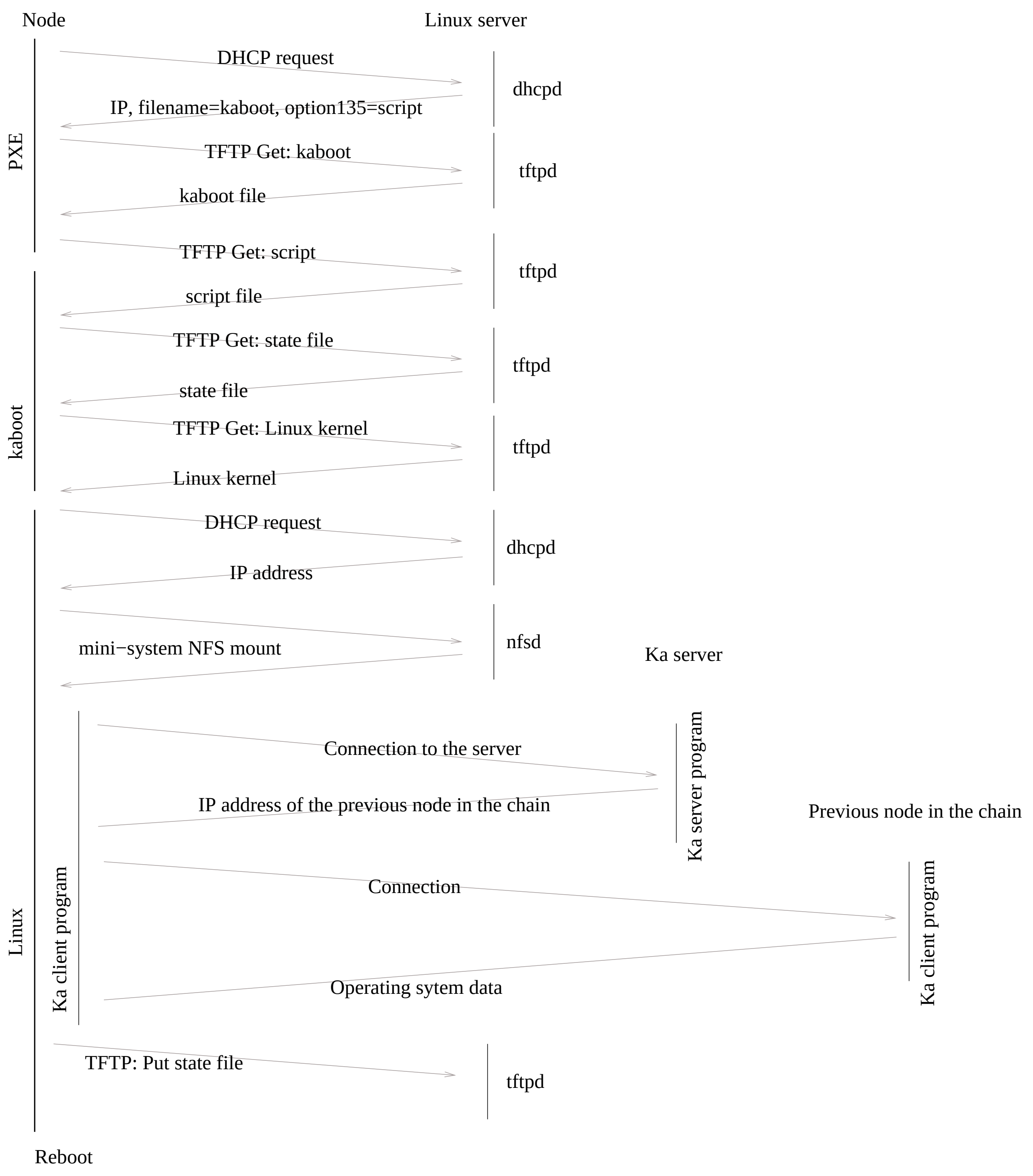

Figure 1

b) File system cloning

Consider first the case of a sequential client-server installation. If many PCs need to be installed at the same time, then the server sends the same complete OS image many times and this operation becomes a bottleneck. Several strategies can be used to remove this bottleneck. Common techniques to broadcast large files on a local network are IP multicast and optimized unicast tree broadcasting. We will compare these techniques at the end of this paper. In this section, we consider the context of a switched network where we found that the most suitable strategy is a “pipelined TCP chain”.
The resulting application is named Ka and is available under the GPL license.

The cloning process with Ka occurs in the following order: a small diskless Linux operating system is run on each client system. This system uses a Linux kernel loaded through TFTP by the boot program and a root filesystem mounted by NFS through the network. Then it creates and formats the hard disk partitions, which will be used to install the operating system and launches the Ka client. The Ka client will connect to a Ka server located on the computer to be cloned. When all the machines to be installed are connected to the Ka server, the chain algorithm starts.
The Ka server coordinates the creation of a chain of TCP connections between clients. Once this chain is created, it is used to broadcast the data (Figure 2). Data is read on the server’s hard disk, sent to the first client, written on its hard disk and at the same time forwarded to the second client, etc. Once all the data have been written on the hard

disk, each computer can reboot and use the system that it has just installed. In order to avoid restarting the whole installation procedure, a status file located on the TFTP server and updated by the installation program tells the computer to perform its post-installation phase.

The broadcasted data can be a tar image of the root filesystem for a Linux system, or a raw partition image for an MS-Windows system.

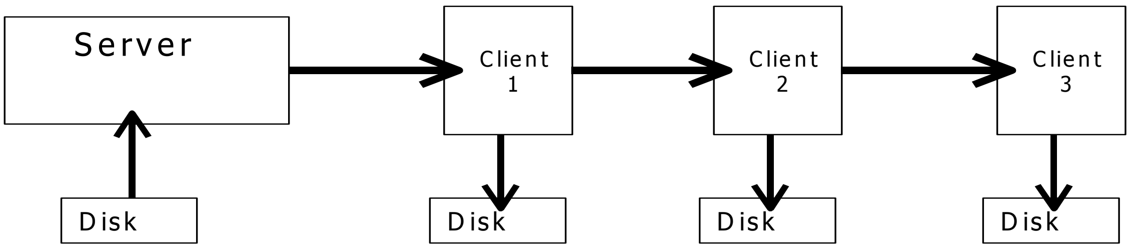

Figure 2: installation chain

c) Post-installation

The post-installation stage is the set of operations executed on a node during its first boot of the newly installed system. These include writing a fresh boot sector on the hard drive by running the lilo command and writing the hostname or the IP address of the node on the configuration files that need it. On Windows machines this post-installation stage includes also joining an NT domain, or generating a new Security ID (SID).

d) Tests

The test bed for our work is a 225-node cluster (Figure 3) with “mainstream” components.
Each node is a PC equipped with a single Pentium III processor with 256 Mb of memory and 15 gigabytes of disk. The 225 computers are connected to an Ethernet gigabit backbone.

The network bandwidth ranges from 100 megabits/s (the computer network interface) up to 1 gigabit/s (the switch interface). The switches interconnect (and thus the hierarchy of the communications) are easily customizable (ring, double ring, tree, star...). The aggregate bandwidth for one switch is 3.8 Gigabits.

The installation of a complete Linux system (3 gigabytes of data) on 225 nodes takes less than fifteen minutes. The installation of a complete Windows 2000 system (2 gigabytes of data on a 3 gigabytes partition) on 100 took less than ten minutes (not including the post-installation process, that takes a few seconds for Linux but may last a few minutes on Windows systems).

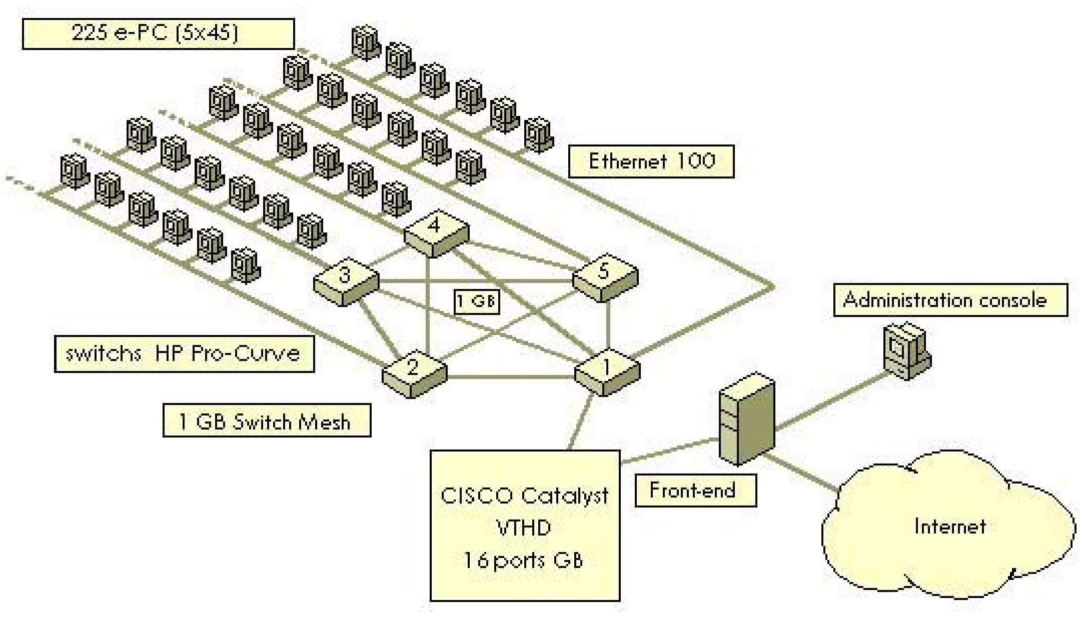

Figure 3: Structure of I-Cluster (INRIA/Lab. Id-imag)

3. *File distribution*

a) Efficient process launching

The one-to-all copy of a file is a common need for the users and administrators. It can be used to upgrade a cluster by upgrading only one node manually and then broadcasting the changes. File staging is another example of the need for efficient data copy on a cluster.
File copying can be carried out in the form of a multicast protocol or a unicast tree-structured protocol. Implementation of broadcasting algorithms is done in the framework of a general library for process management named rshp [1] whose main component is a program launcher.

The objective of the launcher is first to quickly start all the processes involved in an application then to convey the signals between each process properly and finally to manage the input/output flows of data. The solution developed within the laboratory uses the standard rshd daemons and a customized rsh client. This customization is twofold.
Considering the parallel rsh command as a multi-step process, one optimization is to minimize the number of steps. Our implementation carries out recursive calls to the rsh client so as to cover the computers involved with a tree. Another optimization is to parallelize each of the three traditional stages of the rsh (distant access, authentication, connection set up). In our implementation, asynchronous distant accesses allow the authentication part (the longest one) to occur in parallel on each client.

We have implemented four tree-structures: binary, binomial, flat (one level) and lambda-tree. Each one may use a synchronous or an asynchronous rsh client. For our 225-node cluster, the best launcher is a flat tree with asynchronous distant calls. In that experiment, the launching phase lasts less than half a second for 200 nodes.

b) File broadcasting

In addition to algorithms described in the previous section, a multicast algorithm and a chain algorithm have been designed for file broadcasting. The chain implementation is the same as for operating system cloning. Our multicast algorithm adds a reliability layer on top of IP multicast. Two techniques based on the IETF drafts [3] were used to achieve reliability, ACK and FEC.
In the ACK technique, the server starts sending multicast packets and waits at regular intervals (every M packets) for an acknowledgement of delivery (ACK) from each client. A client sends one ACK for N packets received, indicating the sequence number of the last received packets. The performance of the multicast algorithm mainly depends on the values M and N.

Multicast clients are launched using the best launching algorithm as presented in the previous section. On our 225-node cluster, we use the asynchronous flat tree algorithm.

We also tested an existing multicast library named mcl [4], based on the FEC (Forward Error Correction) techniques. It consists of transferring redundant packets so that a client could rebuild lost packets using redundant ones. This technique appeared not to be suited for our context because the cost of coding/decoding a packet is high compared to the network transfer costs.

The figures 3,4 show the performances of the binomial, binary, chain and multicast algorithm on our cluster for respectively 16 and 201 nodes.

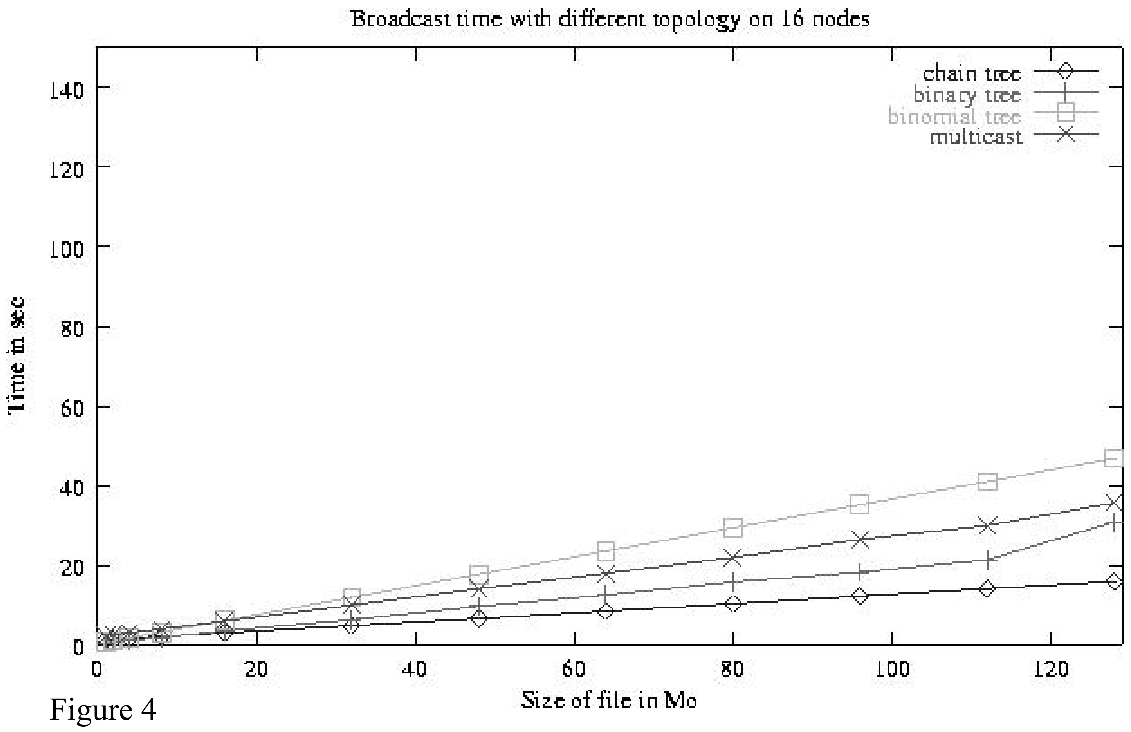

Figure 4

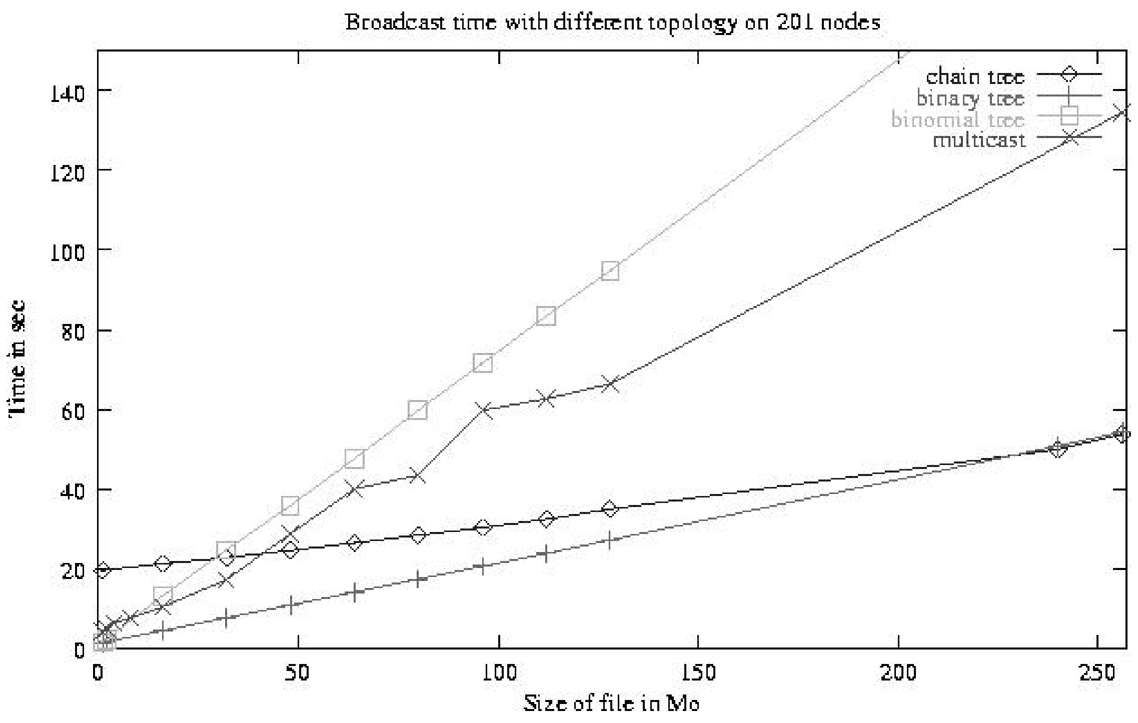

Figure 5

From this, we see that the binary algorithm is good for small files while the chain algorithm is the best for large files. What we should call "small files" depends on the number of nodes.

The binomial algorithm and the multicast algorithm based on ACK are useless in our switched network context. They should be useful for a cluster whose overall bandwidth is limited. Note the multicast algorithm might be improved by using more dynamic time windows.

c) Mixed algorithms

In the context of our switched cluster, it is possible to compare unicast algorithms.
Suppose d is the time needed to establish a connection from one node to another. Suppose L is the time needed to transfer a file of size s from one node to another once the

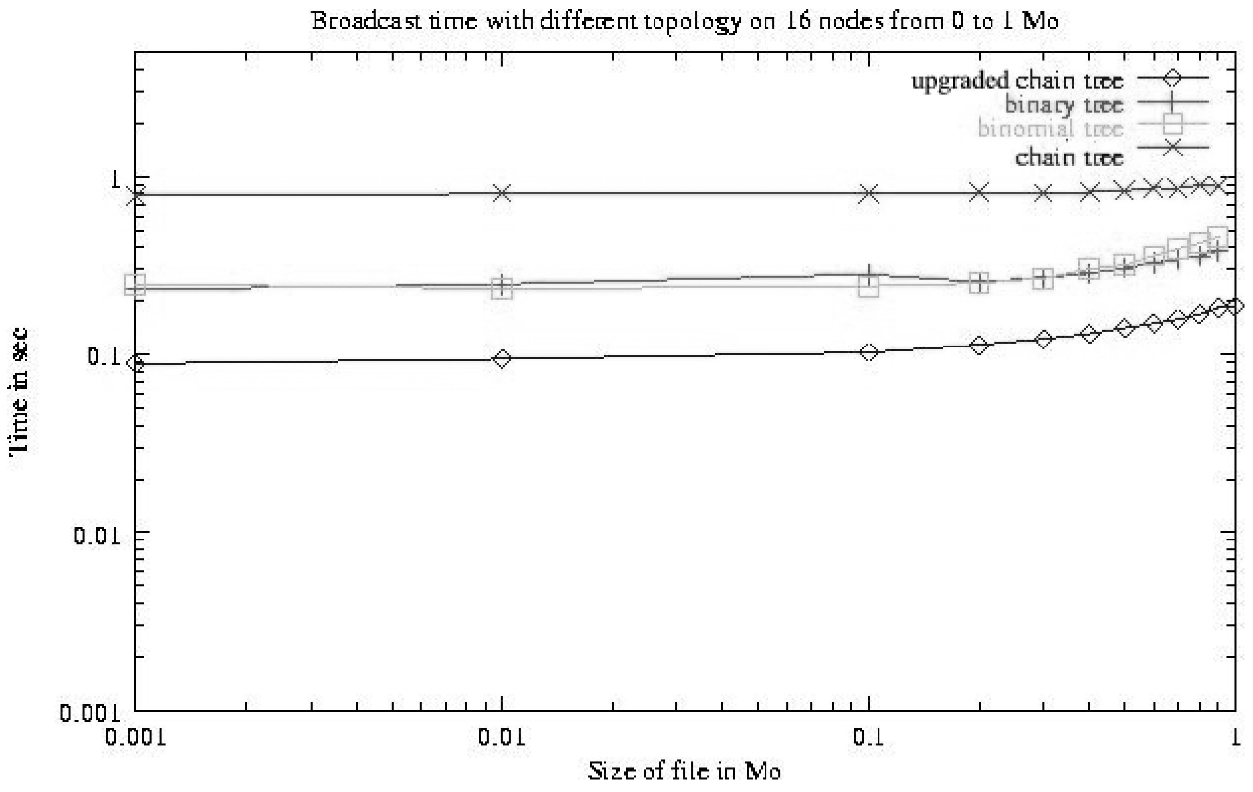

Figure 6

connection has been established. We assume that a node cannot do anything else while opening a connection.
Therefore, the total time to transfer a file from one node to another is $T = d + L$

In the case of an n-ary tree: at each level, the time needed by a node to contact all the nodes below him is $d \times n$. This will have to be done for each level, so if the tree has k levels, the total time for establishing the connections is $(k-1) \times d \times n$.

The data will be pipelined from the top of the tree down to all the leaves, and the node that has the most children will limit the speed of this transfer. In this case, it has n children, so the transfer time for all the nodes will be $L \times n$. Therefore the total time for an n-ary tree is $t = (d \times (k-1) + L) \times n$.

If n=2, then we have a binary tree with $N=2^k-1$ nodes and $t = (d \times (k-1) + L) \times 2$. In the case of a chain, we take n=1 and then k=N, so $t = d \times (N-1) + L$.

In the case of a binomial tree: at each step, the transfer time is d + L. For a binomial tree with k steps, the total transfer time is $t = (d + L) \times k$.

From this, we can produce a formula to decide which algorithm is the best depending on d, L and k and provide the user with a program that automatically chooses the good algorithm depending on the file size and the number of nodes involved in the broadcast.

On our 200-node cluster, we have d=0.1 seconds thus d x N, the time for initializing a connected chain is about 20 seconds for 200 nodes. Yet, we presented in a previous section a parallel asynchronous launcher that takes less than 1 second to start a command on 200 nodes. The reason is that the authentication phase in the rsh daemon is very long compared to the network latency. It then happens to be worth mixing the two techniques (asynchronous launching and chain) to perform optimal broadcasting.

Suppose now that we establish the chain using the asynchronous rshp algorithm as described in previous section. Let I be the duration of this initialization. Let d'<<d be the TCP/IP latency of the network.
The corresponding transfer time for the mixed algorithm is then t'=I+ L + d' x (N –1).

The figures 5 and 6 show that the upgraded chain algorithm is better than all other algorithms tested even for small files. The curve is almost the same as would be the curve for a simple point-to-point file transfer and the time overhead is very small, between 10% and 25%.

Although the initialization phase could also be improved for the binary algorithm, we guess that the upgraded chain algorithm will outperform the binary algorithm except for very small files.

4. Related work

There are a numerous tools for PC cloning mainly in the “Microsoft Windows” environment, Ghost, being the most famous one.
The Unix “ghost-like” software, CloneIT, g4u and Dolly also use a floppy (or a dedicated hard disk partition) to start the cloning process, thus are not adequate for large sized cluster. Yet one could combine the quoted software with a PXE-compliant boot system to provide a scalable

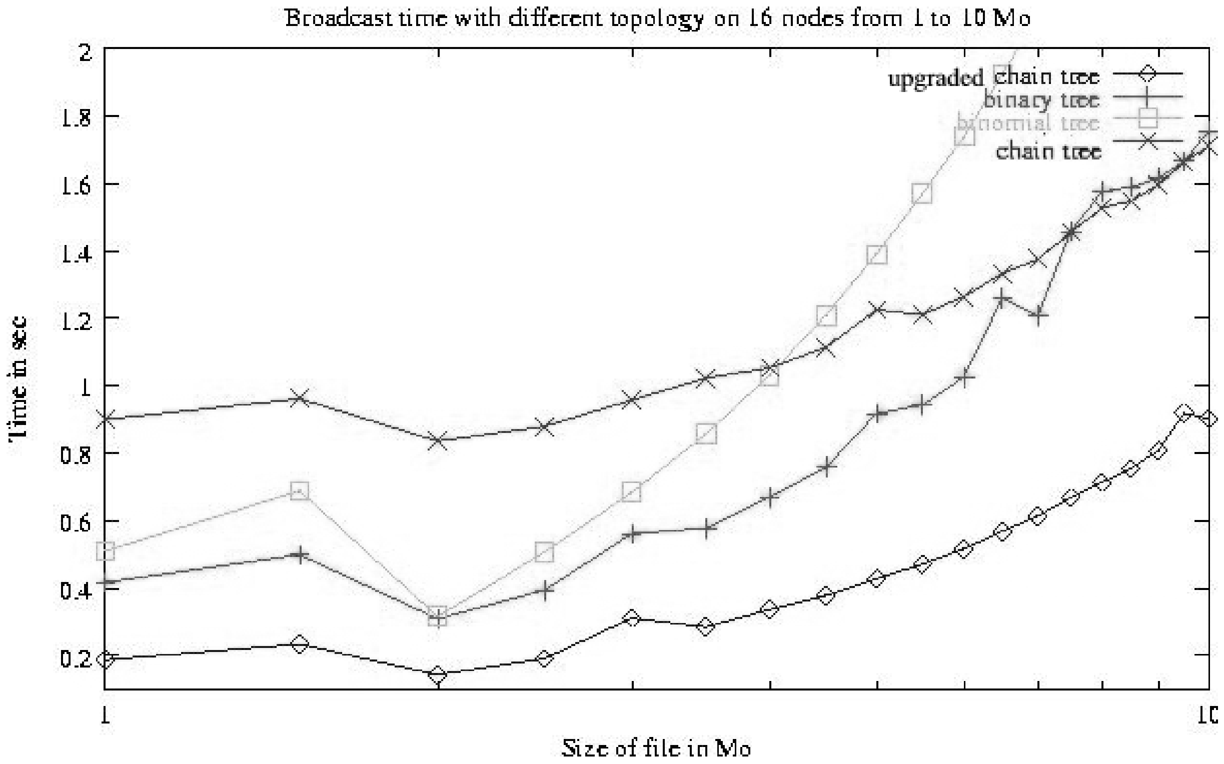

Figure 7

cloning system in the same way as Ka do.

It is worth noting that Ghost and Dolly respectively use a multicast and a TCP chain protocol that are both available in Ka.

BpBatch, Rembo and Beoboot [6] are non-open source cloning tools. The latter tools have been used to install large infrastructures. In addition, BpBatch provides a scripting language to perform pre-installation configuration.
SIS [12] is an operating system installation project whose main objective is to define and maintain a rich set of image "flavors".

As far as parallel commands are concerned, parallel programming environment such as SCE [13] and Cplant [5] provide a set of commands for managing a cluster including file copying and process management. TACO [12] is a general API for collective communication operations. SUT [14] and C3 [2] which focus only on parallel commands, do not provide, to our knowledge, the same wide range of algorithms as Ka do for collective operations or for operating system installation.

5. *Conclusion*

We have presented a toolkit for optimizing file distribution on a large-size infrastructure. It provides both the software for operating system installation and file broadcasting based on multicast, chain and several tree-structured algorithms. The toolkit should find a good distribution process for various network architecture.
In the case of a switched network architecture, a chain algorithm is combined with the use of an optimized launcher. Broadcasting a file to hundred of nodes takes not much more time than a single PC-to-PC file copy.

References

[1] C. Martin, O. Richard. Parallel launcher for clusters of PC, submitted to World scientific
[2] C3, Cluster Command & Control (C3) Tool Suite, Submitted March 2001 for review to special issue of PDCP (Parallel Distributed Computing Practices) available at http://www.epm.ornl.gov/torc/C3/
[3] Reliable Multicast http://search.ietf.org/rfc/rfc3048.txt, January 2001
[4] V. Roca, ``On the Use of On-Demand Layer Addition (ODL) with Multi-Layer Multicast Transmissions Schemes", 2nd International Workshop on Networked Group Communication (NGC 2000), Stanford University, California, USA, November 2000
[5] R. Brightwell L. A. Fisk, Parallel Application Launch on Cplant, SC2001 technical paper, November 2001
[6] BpBatch: http://www.bpbatch.org/
[7] CloneIt http://www.ferzkopp.net/Software/CloneIt/CloneIt.html
[8] Cluclo http://members.linuxstart.com/~flux/cluclo/
[9] Get4u http://www.feyrer.de/g4u/
[10] Dolly http://www.cs.inf.ethz.ch/CoPs/patagonia/dolly.html
[11] SIS http://systemimager.sourceforge.net/
[12] J. Nolte and M. Sato and Y. Ishikawa, TACO -- Exploiting Cluster Networks for High-Level Collective Operations, http://citeseer.nj.nec.com/417863.html
[13] P. Uthayopas, S. Phatanapherom, T. Angskun, S. Sriprayoonsakul, SCE: A Fully Integrated Software Tool for Beowulf Cluster System», Proceedings of Linux Clusters: the HPC Revolution, National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana, IL, June 25-27, 2001
[14] SUT “Unix Tools on Massively Parallel Processors” William Gropp and Ewing Lusk http://www.unix.mcs.anl.gov/sut/old/long2/long2.html
[15] M. Bernaschi and G. Iannello. Collective Communication Operations: Experimental Results vs. Theory. Concurrency: Practice and Experience, 10(5):359--386, April 1998
[16]T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MAGPIE: MPI's Collective Communication Operations for Clustered Wide Area Systems. In Proc. Symposium on Principles and Practice of Parallel
Programming (PPoPP), Atlanta, GA, May 1999
[17] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subronomian, T. van Eicken, LogP: Towards a Realistic Model of Parallel Computation, Proc. of 4th ACM Symp.
on Pinciples and Practice of Parallel Programming, 1993
[18] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model --- One Step Closer Towards a Realistic Model for Parallel Computation. In Proc. Symposium on Parallel Algorithms and Architectures (SPAA), pages 95--105, Santa Barbara, CA, July 1995.