PARALLEL LAUNCHER FOR CLUSTER OF PC

CYRILLE MARTIN
OLIVIER RICHARD
APACHE project, INPG, ID-IMAG, antenne de Monbonnot - ZIRST, 51 ave Jean Kuntzmann 38330 Monbonnot - France
E-mail: Cyrille.Martin@imag.fr, Olivier.Richard@imag.fr

This article describes an experimental parallel program starter for large-sized clusters of personnal computers (PC). The starting of programs is parallelized by broadcasting a remote request execution. Broadcasts are performed using a spanning tree. Several experiments were conducted with N-ary and binomial trees, the latter performing better when the size of the cluster increases.

1 Introduction

Large-sized clusters (over 100 processors) are more and more used for high-performance computing. In order to exploit a large cluster, users or system administrators need efficient tools to install, administrate and monitor a cluster.
To do that, several types of tools have been developed. To install software on a set of PC, the tool Ghost [1] uses a server which safetly broadcasts binary files. The SCMS [2] tool developed in the SMILE project provides the most basic Unix administration commands for a cluster. It is based on remote command executions performed iteratively (remote shell: rsh, rexec, ssf, ssh). The C-PLANT cluster project[3] uses a set of dedicated deamons to perform administration and monitoring of large clusters. The monitoring tool Bwatch [4] probes iteratively memory and CPU load. This list is not complete because many other tools have been developed to solve specific problems.
Those problems can be easily resolve with efficient multicast and reduce (merge) mechanisms [5]. These mechanisms have been studied intensively to provide collective communications for parallel computing. Vadhiyar study [6] gives a specialization algorithm taking to *accent* message size and network topology, to customize collective communications of MPI. Kielmann, from the MagPIe project [7], extends the collective communications of MPI to metacomputing. He distinguishes two communication levels : efficient inside a cluster, and slower between two clusters.
However, there exist no efficient and generic mechanism to build tools requiring efficient broadcast and reductions for system administration and monitoring. In a first step to build a generic tool, we propose in this article to study

an efficient parallel program launcher for large-sized clusters. We assume that the interconnection network of the cluster can perform parallel communications between independent node pairs. This is the case of switched Ethernet networks, multi-stage interconnections such a Clos Myrinet switched [8] and 2D or 3D grids of SCI [9].
In the following sections, we explain the design of parallel program starter prototype. The description of the algorithm used will be followed by a presentation and analysis of performance results. We will conclude with the problems raised by the design of a generic tool.

2 Case study: launcher program for clusters

In this article, we will focus on two approaches of the launching of programs we will only focus on the starting of a program or a command on a cluster. On small-sized clusters, the classical technique used for example in **mpirun** of MPICH [10], is to start one node after another. The cost of this algorithm is linear in the number of processors and therefor not scalable. For larger clusters (C-PLANT [3], Score [11]), solutions used dedicated daemons were developed.
To build an efficient launcher without dedicated daemons requires to parallelize the remote execution call mecanism ("remote shell Unix": rsh, rexec, ssh, ssf). A simple parallelization technique is based on recursive starting of programs on nodes. The starting scheme equals to a spanning tree composed by the cluster nodes [12]. Our prototype has been built following this way.

2.1 Principle of the standard rsh protocol

The principle of the **rcmd(A)** (fig1) client command consists in requesting from a remote **rshd** server to create a process which executes the **A** command. A TCP connection is used to redirect the standart input and output of the remote process. This connection is closed when the **A** remote program is finished. Figure 1 shows the different stages of a remote execution call : setting up of a TCP connection, remote user identification and loading of program.

2.2 Principle of the launcher

The parallel starter triggers a broadcast to N other nodes parallelized in the following way:

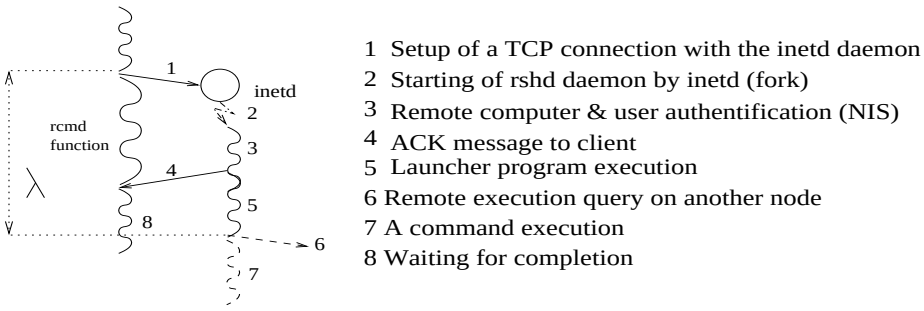

Figure 1. **rcmd** command execution: Setup of a TCP connection with the **rshd** daemon (via **inetd**), user authentification, A command execution.

- The initial node starts **k** remote processes, and keeps TCP connections established by the remote execution call (See 2.1).
- Every one of those **k** nodes start **k'** nodes too, until all the **N** nodes concerned with the execution of the **A** program have been reached.
- At the end of the **A** program, a node terminates when all of its children nodes have closed their TCP connection.

We construct a diffusion tree which links all of the processes executing the **A** program. Starting a program rapidly is equivalent to performing a collective communication in which the message is composed by the program name, its arguments and the user ID. Since at each step of the algorithm, as those starting is composed of communications between independent node pairs, it is possible to exploit the physical network parallel communication possibilities.

Before the program or command execution, we must consider two situation. The first one is when the program reside on each node, in this case not any additional operation is required. Other, the program reside on the server side and must be broadcast on all nodes. We have considered two approaches : the use of distribued filesystem facilities (NFS) and the exploitation of the launcher's diffusion tree.

3 Evaluation

In the first prototype, we have just considered N-ary and binomial tree. Figure 2 shows the temporal development for the two types of tree used. First, we

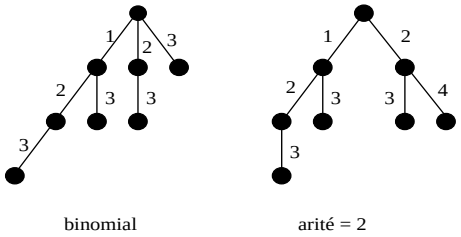

Figure 2. Construction of a 8 nodes binomial and binary tree.(figures indicates the steps of the algorithm)

evaluate the cost of starting time when the program reside on each node. Afterwards, we consider additional time for the diffusion of the program on all nodes.

3.1 Cost model

Let λ be the cost of one step (a **rcmd** time fig 1). The cost of starting an A-ary tree is $\lambda A \log_A(N)$ where N is the number of nodes. For a binomial tree this cost is $\lambda \log_2(N)$. This cost model can be see as a simplification of the model of Bernaschi and Iannello [5].

In the following sections, we will describe the experimental conditions and analyze the performance results obtained with our program launcher prototype.

3.2 Experiments

The experiments were conducted on the ID-IMAG Laboratory cluster composed of 100 PCs connected by a 100 Mbps switched Ethernet network including 3 switchs connected by two 1Gbps links(33 nodes per switch). Each workstation is a 733 Mhz Pentium III, with 256 Mbytes of memory running the Linux operating system. The interconnection network is not completly switched. When the binomial tree algorithm is used, it is possible that each of the 33 nodes connected to one of the switchs sends a message (execution request) to 33 nodes to another switch. This situation may overload the link between those two switchs. To avoid this situation, it is sufficient that the next node of the spanning tree distributes its children on the different switchs.

In order to reduce perturbation experiment, we have not used centralised server for the authentification mecanism(NIS).

We have conducted two set of experiments. In the first set, the step of the broadcast of the program is not considered. In the second set, the NFS filesystem or the launcher's spanning tree are used to broadcast the program. The results were obtained with 20 tests by measure with a confidence interval of 95%. Times measured include the construction of a spanning tree, the broadcast time according to the case and a null program execution time. These tests were made with different N-ary tree and the binomial tree. Times measured include a termination time which depends on the heigth of the spanning tree. As the cost of a remote request execution is important ($\lambda \geq$ **rcmd** time $\simeq$ 100 ms), the termination time (termination time $\simeq h * t$, with h: tree heigth, t: time to close a TCP connection) can be ignored in this first experiment since.

3.3 results analyzis without broadcast

Figure 3 compare launching time obtained with N-ary and binomial trees. On a small number of nodes (fig 3.b), the performances of the N-ary trees are nearly identical to the performances of the binomial tree. A binomial tree is better in theory when the tree is complete (number of nodes $= 2^n$). When the number of nodes is not a power of 2, N-ary trees can have the same efficiency. When the cluster size increases, the binomial becomes the more efficient, but the performance gain is limited to 20% for 64 nodes.

3.4 results analyzis with broadcast

Figure 4 presents the launching time with the broadcast of the program (NFS and diffusion tree) and for sevral spanning trees. The size of program is 1 Mbytes for graph **a** and the binomial tree is used for the launching in the NFS test. The graph **a** shown that NFS server can't scale when cluster size increases. Broadcasting with the binomial or binary tree are nearly identical. At this size of program, the time for the establishment of diffusion trees is predominant. The graph **b** of the figure 4 compares the launching time between the 1-ary (chain) and binary trees with a 16 Mbytes program. The pipeline effect explains that the time for chain is better than for binary tree up to 12 nodes. The bandwith for binary tree is twice lower than for chain. Over 12 nodes, the time for the etablishment of the chain is too higth compared to the gain on the bandwith.

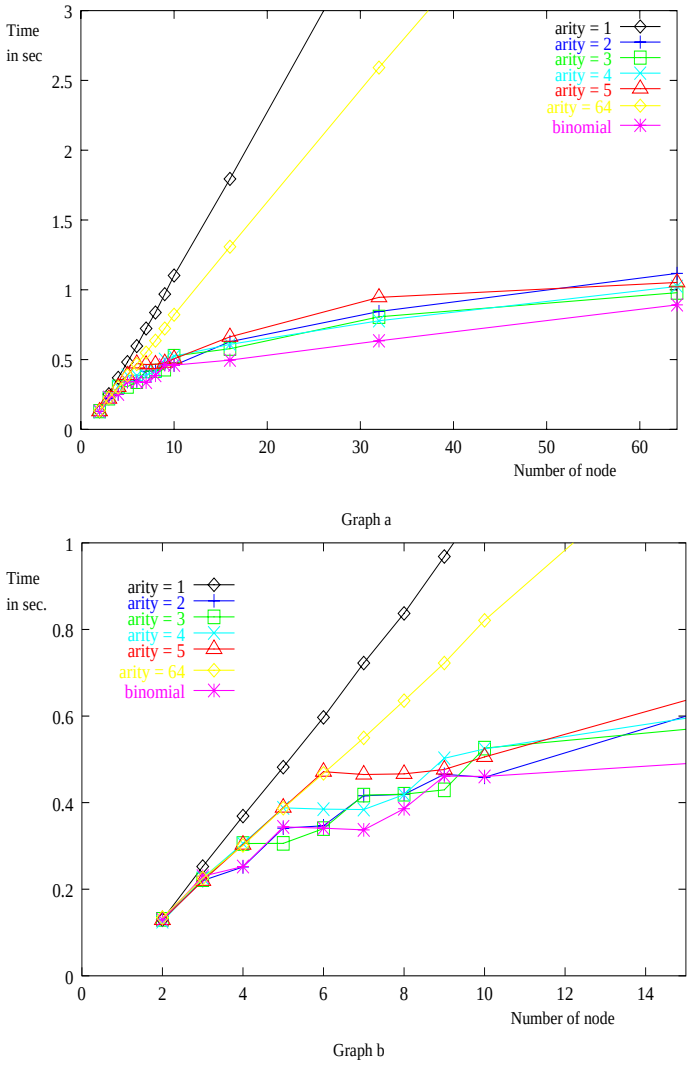

Figure 3. Comparaison of program starting time without the broadcast of the program for several spanning tree topologies: N-ary and binomial, as function of the number of nodes. The **b** graph focuses on a small number of nodes.

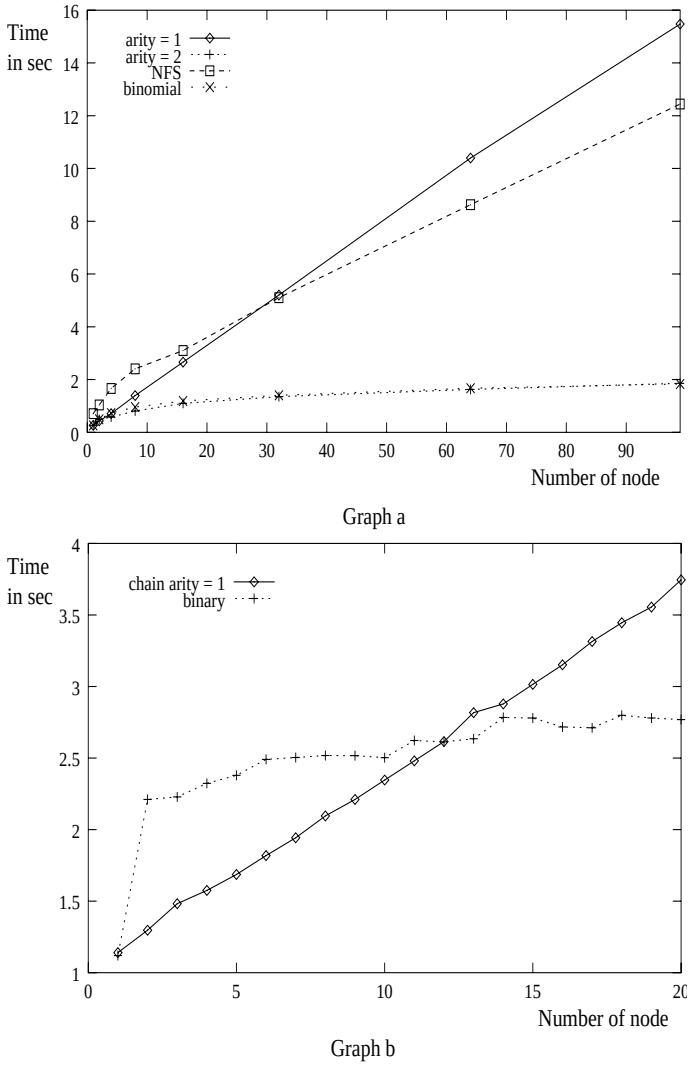

Figure 4. Comparaison of program starting time for several spanning tree topologies and use of the NFS filesystem and the launcher's spanning tree for the broadcast of the program. The **b** graph focuses on the difference between two topologies : 1-ary (chain) and binary.

4 Discussion and future work

The experimental results given above indicate that a binomial tree is more efficient to launch applications on a cluster when the program is on each node. However the performance differences between N-ary trees (1 up to 4) and a binomial tree are limited. When the program must be diffused on all nodes, the binomial and binary trees have nearly the same performance. With large programs the chain topolgie had better performances on small number of nodes. For increase the performance of chain, the time of its establishment must be reduce. Future works include that point and performance measures for larger size clusters.

To conclude, we have built an efficient launcher for cluster, which shows several critical points : the remote execution request cost and the broadcast of program.

References

1. symantec. http://www.symantec.com/ghost.
2. Scalable Multicomputer Implementation using Lowcost Equipement. Smile cluster management system,thailand, http://smile.cpe.ku.ac.th/.
3. R. Riesen, R. Brightwell, L. Fisk, T. Hudson, J. Otto, and A. Maccabe. Second extreme linux workshop, monterey, california, 1999.
4. Jacek Radajewski. Bwatch http://www.sci.usq.edu.au/staff/jacek/bwatch/.
5. M. Bernaschi and G. Iannello. Collective communication operations: Experimental results vs.theory, to appear in concurrency: Practice and experience, 1998.
6. Sathish S. Vadhiyar, Graham E. Fagg, and Jack Dongarra. Automatically tuned collective communications. In ACM, editor, *SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA, November 4–10, 2000*, pages 46–46, New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 2000. ACM Press and IEEE Computer Society Press.
7. T. Kielmann, R. Hofman, H. Bal, A. Plaat, and R. 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., 1999.
8. K. Verstoep, K. Langendoen, and H. Bal. Efficient reliable multicast on myrinet. in proceedings of the international conference on parallel processing, pages iii:156–165, aug 1996., 1996.
9. The Institute of Electrical and Electronics Engineers. Ieee standard for scalable coherent interface, 1992.
10. W. Gropp, E. Lusk, N. Doss, A. Skjellum, and A. high performance. portable implementation of the mpi message passing interface standard. parallel computing, 22(6):789–828., 1996.
11. Parallel and Japan Distributed System Software Laboratory. Real world computing partnership. mpich-pm/clump : an mpi librairy based on mpich 1.0 implemented on top of score. http://pdsw3.trc.rwcp.or.jp/mpich-pm/., 1998.
12. Jean de Rumeur. *Communications dans les rseaux de processeurs.* 1994.