Copyright © 1996, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

XDISTRIBUTE: A PROCESS DISTRIBUTION SYSTEM

by

Karl Petty and Nick McKeown

Memorandum No. UCB/ERL M96/67

13 November 1996

XDISTRIBUTE: A PROCESS DISTRIBUTION SYSTEM

by

Karl Petty and Nick McKeown

Memorandum No. UCB/ERL M96/67

13 November 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

XDISTRIBUTE : A PROCESS DISTRIBUTION SYSTEM

Karl Petty
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720
Email: pettyk@eecs.berkeley.edu

Nick McKeown
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305-9030
Email: nickm@ee.stanford.edu

Abstract - **Powerful, single-user workstations - now common in the workplace - spend most of their time sitting idle. In an attempt to find low-cost computational power researchers have attempted to take advantage of these idle machines. Many systems have been designed, ranging from simple process distribution like Condor [1], to completely new operating systems like Sprite [2] and Amoeba [3]. Although very powerful, these systems are out of reach of the typical user - they require special privileges, and require specialized installation, set-up and maintenance. We propose a simpler way to utilize these idle machines using a remote execution environment. We have written an application, *xdistribute*, that is extremely portable, simple to use, requires no changes to any source code, leaves no residual dependencies on the local machine, and can be used by any user without modification to the operating system or installation of any network daemons.**

1 Introduction

With the development of powerful yet inexpensive workstations it is common for universities and companies to have a large number of single user workstations available to users. It is well known that in such an environment many workstations are idle at any one instant [5][6]. There are several good systems that have been developed to take advantage of this situation, but we feel that there is a significant group of users that have been overlooked by the traditional process distribution community. This is the group of users who don't have the administrative support to setup and install a full-fledged process distribution system, yet need more computational power than one workstation can provide. With this group of users in mind, we have designed a new application, *xdistribute*, with the following characteristics:

1. **No special privileges are required.** *xdistribute* does not require the installation of any daemons on any machines, nor does it require any modifications to the kernel. The package can run with only a modicum of setup.
2. **Xdistribute works in a heterogeneous environment.** *xdistribute* is written in the machine independent, and commonly available, languages of Tcl/Tk and perl. Therefore *xdistribute* runs on a multitude of platforms.
3. **No residual dependencies are left on local machine.** *xdistribute* can distribute and manage 100's of processes at a time. Therefore it is very important that the process distri-

* This work was supported by the National Science Foundation, California MICRO (442427-41679) and Pacific Bell.

bution system not exceed the limit on the number of processes or open file descriptors for individual users on the local machine.

4. **There are no file system assumptions.** If a uniform file system is available then the user can take advantage of it with features provided by *xdistribute*. Otherwise, *xdistribute* doesn't make any assumptions about the current file system, and doesn't assume identical userIDs across platforms.

Xdistribute is designed for users who want a process distribution system yet don't have special privileges or want to maintain system software. However, since *xdistribute* uses no network daemons it has no idle resource detection mechanism. Therefore jobs cannot be placed in an optimal manner. The implication of these assumptions on the type of jobs that can be run will be discussed in Section 3.

Because of its portability we feel that *xdistribute* fills a significant gap that until now has not been addressed. We believe that this program helps harness the power of idle workstations for the average user. In several case studies in heterogeneous environments we have found *xdistribute* extremely easy to use.

In this paper we are not arguing against transparent process migration. Quite the contrary: if it were widely available across multiple platforms we would use it. Instead, we argue that for a common class of processes, a simpler process distribution system will often suffice. *Xdistribute* gives an average user - with no special privileges and no desire to maintain an operating system - the ability to take advantage of idle processors in a network of computers. *Xdistribute* is extremely portable: it was written in Tcl/Tk and perl and hence will run on a wide variety of platforms. There is no privileged setup or installation required.

2 Process Distribution

Previous techniques to take advantage of idle processor cycles fall into two categories: solutions requiring a new operating system, and solutions requiring only network daemons. We present a brief overview of these two techniques and cite several implementations. We refer the reader to [8] for a discussion of the features of distributed operating systems, and to [9] for the details of process distribution systems.

2.1 The Operating System Approach

In distributed operating systems an entire process is moved to a remote processor and restarted. Moving a process requires that both its code and data be migrated to the new processor. In the best case, process migration will be transparent to the process; the process should not know that while it was executing it was moved from one processor to another. The problem with moving a process is that some of its state is stored in kernel data structures. When a process migrates, how should the kernel data structures be updated? How is an open file descriptor moved? When is the code and data moved to the new processor: immediately, or upon reference?

To solve these problems, developers of distributed operating systems have created transparent process migration systems. To do this, they have either 1) modified, or built on top of, existing operating systems, or 2) implemented completely new operating systems. This approach has several advantages: the file system seen by each machine can be uniform; processes can freely move

from machine to machine (to balance load or to evict processes when a machine's owner returns), and process signals can be propagated to migrated processes.

A few examples of completely distributed operating systems are Accent [10], Amoeba [3], Charlotte [11], Linda [12], Sprite [2], and the V-System [13]. GLUnix [14] is a compromise between a distributed operating system and a process distribution facility. Written as a layer of software that runs on top of an existing operating system,* GLUnix can make a network of computers look like a global operating system by catching system calls. Although GLUnix was written entirely at the user level, it supports transparent process migration.

Fully distributed operating systems offer the ultimate in harnessing idle computational power: everything is transparent to the process. The problem with these operating systems, however, is that just like any novel operating system, they need to be installed and maintained. This currently places them out of the reach of most users.

2.2 The Network Daemon Approach

For some processes, the complexity of designing and implementing a new operating system outweighs the benefits of sharing the computational load. Therefore systems have been implemented that allow users to distribute processes over workstations with generic or unmodified operating systems. These systems typically require the installation of a network daemon on each machine that assists in the distribution scheme.

Several systems have been implemented that use network daemons. Some systems are able to move processes from machine to machine using a "wrapper" around each process. At the time of process migration, the wrapper saves pertinent information about the process: open file descriptors, data segment, stack contents, etc. This allows the process to be restarted at a later time, possibly on a different machine. Some systems do not support process migration, but instead provide a remote execution environment. The user submits commands to the process distribution system and the system will execute the command on an idle machine in the network. The process is not moved to the other machine by the process distribution scheme; these systems typically require a uniform file system so that all programs are accessible by all machines in the network. Prototype process distribution systems of this type include Condor [1], Butler [5], Utopia [9], and the Portable Batch System [16]. Commercial systems include TaskBroker by Hewlett Packard [18] and Jobware by Ockham Technology [19].

While these systems and less complex than the operating system approach, they do require the installation of network daemons on each machine participating in the process migration scheme. The network daemons assist in the distribution scheme in providing the remote execution environment, maintaining communication links between the user process and the user, translating file system calls, and transferring process signals. They also typically assist in detecting idle machines and hence in the process scheduling.

While network daemons help the distribution scheme get around the operating system problems, they have drawbacks as well. The first problem is that network daemons, by nature, need to be installed and run with special privileges. If a user does not have special privileges (or abilities) on the machines then they can't install them. If the machines are not in the same administrative domain then coordination of daemon installation becomes difficult.† The second problem is that

* The developers of GLUnix refer to it as a *virtual operating system layer.*

when not being used, these daemons take up resources. In order to perform scheduling decisions based on idle resource detection, all machines need to have these network daemons running and periodically report their loads to a central server*. While typically small, this still places an unnecessary load on the machines and the network. The final problem is that network daemons can introduce security holes. Since the network daemons accept connections via the network it is crucial that they authenticate the sender. Failure to do so would result in an unauthorized person being able to run arbitrary commands on the machine.

2.3 The *xdistribute* Approach

Xdistribute is simple. It requires no changes to the operating system and uses no network daemons. As a result, *xdistribute* is easy to install, easy to configure and easy to use. It requires no special support, no special administration and no special privileges. Altogether, these attributes make process distribution available to a much wider population of users, and on a wider variety of platforms.

But the simplicity of *xdistribute* comes at a price - some processes will not run on our system. In particular, *xdistribute* cannot distribute processes that communicate with each other. *Xdistribute* does not offer support for process migration, process checkpointing, inter-process signals, or transparent file systems. *Xdistribute* is designed to run monolithic, independent jobs that require no coordination.

Xdistribute operates as follows. Given a list of processes and machines, it executes the processes on the machines until all the processes are complete. It performs remote execution using standard remote shells and hence does not rely on any installed daemons on either the local or remote machine. *Xdistribute* ensures that all processes complete - if one machine dies, or becomes unavailable, processes are restarted on a different machine. *Xdistribute* is careful to use only those remote machines that are idle, evicting its processes if the machine becomes busy. *Xdistribute* determines which input files to pass to the remote machine, and returns output files to the server on completion.

Xdistribute's simplicity invites two obvious criticisms. First, since the typical user already has the ability to run jobs on remote machines via a standard remote shell from the command line, what does *xdistribute* have to offer? Second, how can *xdistribute* be useful when it restricts the types of processes that can be run?

First, *xdistribute* should be viewed as an organizer of remote shell commands. But it does more than that. It allows the user to effortlessly distribute jobs to 100's of machines. It will restart jobs when machines fail; it allows users to view input and output files; and it evicts jobs (either by pausing them, or by restarting them on a different machine) when machines become busy. While *xdistribute* is running the user can add and delete machines from the pool of available machines. Processes can also be dynamically added to the queue. In short *xdistribute* provides a complete user level process management system.

Second, *xdistribute* is suited for any independent, asynchronous jobs. Typical examples of this type of job would be Monte Carlo simulations, Parallel branch-and-bound algorithms and most

† Also, it is possible (and has been witnessed by the authors) that network administrators think that process distribution is a bad idea. In situations like this the probability of good administrative support, and the installation of network daemons, is minimal.

* In the case of a decentralized scheduler [9] these daemons respond to broadcast queries.

custom built simulation routines. We feel that this is a large enough set of jobs to justify a user level tool. Jobs that require synchronization can't use *xdistribute*, and typically need tightly coupled processors with message passing support.

There are many advantages and disadvantages to using *xdistribute*. Most of them stem from the fundamental design philosophy. If one wants to solve the process migration problem completely then one designs a new operating system. If one wants to take advantage of idle processor cycles, do quick idle resource detection and be most considerate to other users, then one builds a process distribution system with network daemons. If, on the other hand, one wants to build a tool that any user can use without setup then one would build an application like *xdistribute*.

The fact that *xdistribute* is compatible with many different machine architectures is a consequence of the widespread support of `perl`. If you can compile your program on the remote machine and `perl` has been installed then you can distribute processes to that machine.

3 Typical *xdistribute* Usage

A typical usage of *xdistribute* is as follows. The user provides *xdistribute* with a list of machines and a list of processes. *Xdistribute* assigns each process to a machine, and ensures that every process is executed. If a machine fails or becomes busy, or if a process dies, the process is reassigned to a new machine.

Machines: The user provides *xdistribute* with a list of machines, and is required to have a regular account on each machine.

Processes: A user provides *xdistribute* with a list of independent, asynchronous jobs that they would like to run. These processes fall into three categories: *preamble*, *user* and *postamble* processes. Preamble and postamble processes are performed exactly once on *every* machine. User processes are executed exactly once on *one* machine. Preamble processes are executed before any user process, and are generally used to install or compile any software needed on the remote machines prior to execution. A typical preamble process will copy custom source code to a machine and compile it on that platform. The postamble process (the counterpart to the preamble process) is used to perform any cleaning up after the user processes have been executed. This could be, for example, the removal of the executable code. The postamble process will typically delete any software that was installed by the preamble process.

Xdistribute stores processes and machines in various queues depending on their state. These queues are displayed graphically in the GUI of Figure 1. The queues on the left hold the processes; those on the right hold the machines. Each queue corresponds to a different state. Processes can be in one of four states: idle (they haven't started yet), active, in error or done. The level of the queue indicates the number of the processes or machines in that state. The processes are color coded to represent the process-type: preamble processes are blue, user processes are red, and postambles are green.

When the system starts up, all of the processes and machines are **idle**. When told to start, *xdistribute* distributes the processes one-by-one to idle machines. When a process is assigned to a machine, both the machine and the process are moved from the **idle** queues to the **active** queues. They remain there until the process completes or dies prematurely. If the process completes successfully then the process is placed in the **done** queue and the machine is placed back into the **idle**

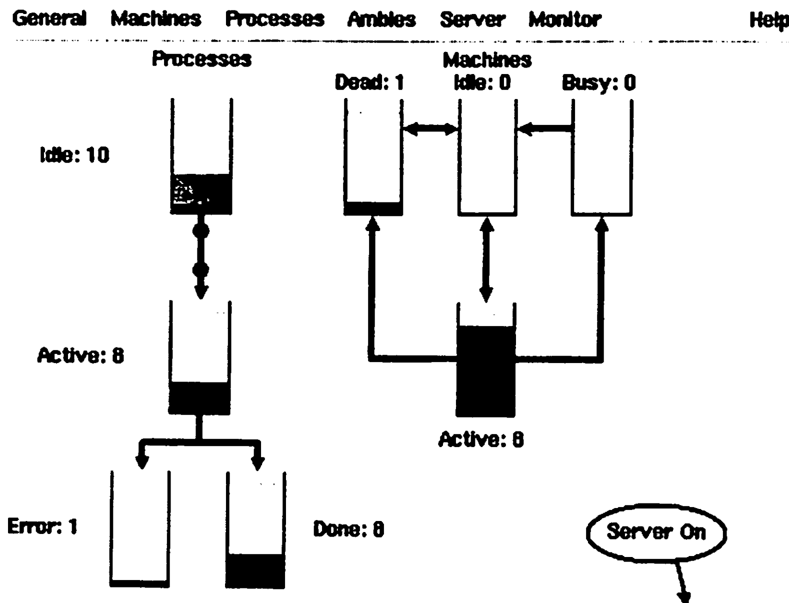

FIGURE 1 Graphical user interface of *xdistribute*. The buffers on the left represent the number of processes in each state. The buffers on the right represent the machines. The different colors of processes reflect the different types: preamble, user, and postamble.

queue. When a process error occurs, the process is placed in the **in error** queue and the machine is placed back in the **idle** queue. If a machine fails, the machine is placed in the **in error** queue and the process is placed back in the **idle** queue. The machine **busy** queue is for machines that are currently being used by another user.

In Figure 1 there are 10 processes that are in the **idle** queue waiting to be assigned to machines. There are 8 processes that are actively running, 8 processes that have completed, and 1 process that is in error. Of the 9 machines in the system, 8 are currently running and 1 is in error.

While *xdistribute* is running the user can manually move processes between queues. For example, a process that crashed and ended up in the **in error** queue can be fixed and manually moved into the **idle** queue to be run again. Or when a crashed machine comes back up the user can manually move it from the **dead** queue to the **idle** queue. The user also has the option of killing a process or a machine that is currently running.

4 Comparison Of Process Distribution Schemes

In this section we will compare the research systems that are similar in design to the *xdistribute* program. These systems are the process distribution systems of Condor [1], Utopia [9], and Butler [5]. The key categories of features that we compare are ease of setup, distribution capabilities, and how considerate the systems are to other users. The comparisons are summarized in Table 1. As seen from the table, the key features of *xdistribute* are that any user can set it up and use it, and it can run on multiple platforms. The discussion below is a brief review of these systems. Readers wanting more detailed descriptions are referred to the references given at the end of this paper.

Category	Item	Xdistribute	Condor	Utopia	Butler
Setup					
	Need new operating system?	no	no	no	no
	Need special daemons installed?	no	yes	yes	yes
	Need root access to install?	no	yes	yes	yes
	Must re-compile or re-link application?[a]	no	yes	yes	no
	Can server run on any platform?	yes	no	sort of	no
	Can process run on any platform?	yes	no	no	no
Distribution					
	Supports process distribution?	yes	yes	yes	yes
	Performs process migration?	no	yes	no	no
	Supports process checkpointing?	no	yes	no	no
	Has residual dependencies?	no	yes	yes	yes
	Requires uniform file system?	no	no	yes	yes
	Process sees uniform file system?	no	yes	yes	yes
Consideration					
	Can machine owners block processes?	no	yes	yes	yes
	Starts processes only on idle machines?	set-able	yes	yes	yes
	Evicts processes on non-idle machines?	set-able	yes	yes	yes
	Automatically determines idle machines?	no	yes	yes	yes

TABLE 1. : Comparison of different process distribution schemes.

a. This only refers to homogeneous hosts. Everybody has to re-compile on different architectures.

The Condor system provides for limited process migration and checkpointing. Condor is unusual in that processes are forced to migrate by physically moving them to a remote machine across the network. Other process distribution systems assume that the process is already on the remote machine and accessible via a uniform file system. So when Condor moves a process, it must enable the process to refer back to the file system on the original machine. This is achieved by encapsulating each process with a layer of code that catches file-system calls, forwarding them to the machine that the process started on. The advantage of this is that the user process never knows that it's being moved to different machines; everything is hidden by the Condor wrapper.

As was mentioned earlier, the problem with process migration is that it is difficult to transfer the entire state of a process to a different machine. It's even more difficult to do so at the user level. The Condor wrapper achieves this by storing a subset of the process state: open file descriptors and

file positions. However, Condor is unable to keep track of inter-process communications, process signals or file operations that read and write to the same file.

Butler was developed to work on top of the Andrew File System [17]. Since this system provides each machine with a uniform view of the file system, Butler does not need to worry about moving processes or files to the remote machine. When a process wants to access a file, it can do so independently of which machine it is executing on. Therefore, unlike Condor, Butler does not use process wrappers. In fact, Butler does not actually move processes around from machine to machine - it only executes commands on different machines. To execute a user process on a remote machine Butler first finds an idle machine to use. This is achieved by a status monitor program that runs on every machine in the workstation pool, and periodically reports its status.

Utopia is the system that is most like *xdistribute*. Utopia gives the user a remote execution environment on a variety of hosts. Like Butler, the user submits a command to be executed; Utopia then finds an appropriate idle machine to execute it on. Since there is no transfer of the process to the machine, Utopia requires that all machines view an identical file system. The developers argue that this is acceptable because most large computing environments are moving towards a uniform file system. In order to facilitate the Utopia process distribution system a network daemon that is started on each host handles the remote execution. When a task is transferred to the remote host, the remote execution server (RES) is contacted. It is passed any information such as environment variables that the application might require. Since the RES will also propagate signals from the user to the application and back, there is a residual dependency in the form of a network connection on the local machine. Utopia performs efficient load-balancing. This is achieved by requiring the use to specify the resources that each task will need. The system matches these resources with specific computers. The system also gathers statistics on every host in the system so that when a task needs to be run the system will know what resources are available.

When comparing these four systems, it is important to note that Condor is the only system that moves processes around from machine to machine. The other three, Utopia, Butler, and *xdistribute*, only execute commands remotely on other machines. Condor, Utopia and Butler all rely on network daemons to facilitate the process distribution and leave residual dependencies on the local machine. *Xdistribute* does not use network daemons nor leave any dependencies on either the local or remote machine.

5 Conclusion

In this paper we have discussed the various approaches taken to harness idle processor cycles in a heterogeneous workstation environment. We have argued that while the traditional approaches of new operating system development and network daemons provide useful tools, they are out of reach of most users. We feel that the research community needs a truly user level process distribution system. In response to this need, we have designed a tool that has a user-friendly graphical interface, is run-able on many heterogeneous platforms without modification, requires no changes to the source code, does not leave residual dependencies on the local machine, and can be set up by any user without installation of network daemons. This tool is useful to individuals with access to a large number of workstations and no administrative support.

References

[1] M. Litzkow, M. Livny, "Experience With the Condor Distributed Batch System," *Proc. of the IEEE Workshop on Experimental Distributed Systems*, Huntsville, Al., October 1990.

[2] F. Douglis, J. Ousterhout, "Transparent process migration: Design alternatives and the Sprite implementation," *Software - Practice and Experience*, 21(8):757–785, August 1991.

[3] S. Mullender, B. van Rossum, A. Tanenbaum, R. van Renesse, H. van Staveren, "Amoeba: A distributed operating system for the 1990s," *Computer*, 23(5):44–53, May 1990.

[4] M. Nuttall, "Survey of Systems Providing Process or Object Migration," Technical Report 94/10, Department of Computing, Imperial College, London, England, U.K., May 1994.

[5] D. A. Nichols, "Using idle workstations in a shared computing environment," *Proc. of the 11th ACM Symposium on Operating System Principles*, pages 5–12, Austin, Tx., November 1987. The Association for Computing Machinery.

[6] R. H. Arpaci, A. C. Dusseau, A. M. Vahdat, L. T. Liu, T. E. Anderson, D. A. Patterson, "The Interaction of Parallel and Sequential Workloads on a Network of Workstations," Technical Report CS-94-838, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1994.

[7] M.M. Theimer, K. A. Lantz, "Finding Idle Machines in a Workstation-Based Distributed System," *IEEE Transactions on Software Engineering*, 15(11):1444-1457, November 1989.

[8] F. Douglis, J. K. Ousterhout, M. F. Kaashoek, A. S. Tanenbaum, "A comparison of two distributed systems; Amoeba and Sprite," *Computing Systems*, 4(4):353–385, Fall 1991.

[9] S. Zhou, J. Wang, X. Zheng, P. Delisle, "UTOPIA: a load sharing facility for large, heterogeneous distributed computer systems," *Software - Practice and Experience*, 23(12):1305–1336, December 1993.

[10] E. R. Zayas, "Attacking the process migration bottleneck," *Proc. of the 11th ACM Symposium on Operating System Principles*, pages 13–24, Austin, Tx., November 1987. The Association for Computing Machinery.

[11] Y. Artsy, R. Finkel, "Designing a process migration facility: the Charlotte experience," *Computer*, 22(9):47–56, September 1989.

[12] N. Carriero, D. Gelernter, "The S/Net's Linda Kernel," *ACM Transactions on Computer Systems*, 4(2):110–129, May 1986.

[13] M. M. Theimer, K. A. Lantz, D. R. Cheriton, "Preemptable remote execution facilities for the V-system," *Proceedings of the 10th Symposium on Operating System Principles*, pages 2–12. The Association for Computing Machinery, December 1985.

[14] R. Wahbe, S. Lucco, T. Anderson, S. Graham, "Efficient software-based fault isolation," *Proc. of the Fourteenth ACM Symposium on Operating System Principles*, pages 203–216. The Association for Computing Machinery, December 1993.

[15] P. Mehra, B. W. Wah, "Automated Learning of Workload Measures for Load Balancing on a Distributed System," *Proc. of the 1993 International Conference on Paralled Processing*, pages 263-270, section III.

[16] R. L. Henderson, D. Tweten, "Portable batch system: Requirements specification," Technical report, NAS Systems Division, NASA Ames Research Center, Moffett Field, CA, April 1995.

[17] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, M.J. West, "Scale and performance in a distributed file system," *ACM Transactions on Computer Systems*, 6(1):51–81, Feb 1988.

[18] "HP Task Broker for HP 9000 Servers and Workstations," HP Product Brief.

[19] Ockham Technology; "Jobware," http://www.ockham.be/jobware.htm