Cluster Computing: The Wave of the Future? *

G. A. Geist
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, TN 37831-6367

May 17, 1994

Abstract

Using clusters of workstations is becoming a very popular method to solve both large and small scientific problems. One of the driving forces behind this shift is the availability of portable robust software to utilize and manage a cluster of workstations.

PVM (Parallel Virtual Machine) is a byproduct of the heterogeneous network research project going on at Oak Ridge National Laboratory and University of Tennessee. It is a software package that permits a user defined collection of serial, parallel, and vector computers appear as one large distributed memory computer. PVM's popularity is quickly making it the worldwide de facto standard for distributed computing.

This talk will describe the driving forces behind cluster computing and the limitations imposed by this method of computing. It will describe the wide breadth of applications being developed in PVM and the features of the latest release of PVM. The talk will also raise several research and social issues that must be addressed before we can make the next big leap in distributed computing.

1 Introduction

Parallel processing, the method of having many small tasks solve one large problem, is being widely used both for high-performance scientific computing as well as for more "general-purpose" applications due to the demand

*This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract DE-AC05-84OR21400

for higher performance, lower cost, and sustained productivity. The acceptance of parallel processing has been facilitated by the the widespread use of distributed computing. Distributed computing is a process whereby a set of computers connected by a network are used collectively to solve a single large problem. As more and more organizations have high-speed local area networks interconnecting many general-purpose workstations, the combined computational resources may exceed the power of a single high-performance computer.

The most important factor in distributed computing's success is cost. Massively Parallel Processors (MPP) typically cost more than $10 million dollars while users see very little cost in running their problems on a local set of existing computers. While It is uncommon for distributed computing users to realize the raw computational power of a large MPP, they are often able to solve problems several times larger than they could using one of their local computers.

One of the difficulties of distributed computing is that the set of computers available on the network is often heterogeneous. For example, they are made by different vendors, have different compilers, etc. In an MPP every processor is exactly like every other in capability, resources, software, and communication speed. Not so on a network. When a programmer wants to exploit a collection of networked computers, he has to contend with several different types of heterogeneity:

- architecture,
- data format,
- computational speed,
- machine load,
- network load.

The set of computers could include a wide range of architecture types such as 386/486 PC class machines, high-performance workstations, shared memory multiprocessors, vector supercomputers, and even large MPP. Each architecture type has its own optimal programming method. In addition, a user can be faced with a hierarchy of programming decisions since some of the constituent computers may themselves be parallel computers. Even when the architectures are only serial workstations, there is still the problem of incompatible binary formats and the need to compile a parallel task on each different machine.

There are incompatibilities between data formats on different computers. This is an important point in distributed computing because data sent from one computer may be unreadable on the receiving computer. Message passing packages developed for heterogeneous environments must make sure all the computers understand the exchanged data. The early message passing systems developed for specific MPP are not amenable to distributed computing because they do not include enough information in the message to encode or decode it for any other computer.

Even if the set of computers are all workstations with the same data format, there is still heterogeneity due to different computational speeds. As an simple example, consider the problem of running parallel tasks on a virtual machine that is composed of one supercomputer and one workstation. The programmer must be careful that the supercomputer doesn't sit idle waiting for the next data from the workstation before continuing. The problem of computational speeds can be very subtle. Since networked computers can have several other users on them running a variety of jobs, the machine load can vary dramatically even when all the workstations are identical. The result is that the effective computational power across even identical workstations can vary by an order of magnitude.

Like machine load, the time it takes to send a message over the network can vary depending on the network load imposed by all the other network users, who may not even be using any of the same computers just the same network. This sending time becomes important when a task is sitting idle waiting for a message, and it is even more important when the parallel algorithm is sensitive to message arrival time. Thus in distributed computing heterogeneity can appear dynamically in even simple setups.

Despite the difficulties that heterogeneity causes in distributed computing, there are many advantages to using distributed computing: By using existing hardware, the cost of this computing can be very low. Performance can be optimized by assigning each individual task to the most appropriate architecture. Heterogeneous network computing is not just about a local area network connecting workstations together. There is the possibility to exploit the heterogeneous nature of the computation, for example, access to different data bases, access to special processors, and application tasks that can only run on a certain platform. The virtual computer resources can grow in stages and take advantage of the latest computational and network technologies. Program development can be enhanced by using a familiar environment. Programmers can use editors, compilers, and debuggers that are available on individual machines. Other distributed computing advantages include: The individual computers and workstations are usually stable, and substantial expertise in their use is readily available. User-level or program-level fault tolerance can be implemented with little effort either in the application or in the underlying operating system. Also distributed computing can facilitate collaborative work. All these factors translate into reduced development and debugging time, reduced contention for resources, reduced costs, and possibly more effective implementations of an application.

The PVM (Parallel Virtual Machine) software package was developed at Oak Ridge National Laboratory as a research tool to explore heterogeneous network computing. The project has since expanded to include several other institutions including the University of Tennessee, Emory University, and CMU. Under PVM, a user defined collection of networked computers appears as one large distributed-memory computer. Due to its robustness, portability and cost, PVM has quickly become the de facto standard for heterogeneous distributed computing around the world with thousands of users and hundreds of applications ported to it.

In the next section we describe the basic features available in version 3.3 of PVM. Section 3 presents some applications that have been ported to PVM. Section 4 describes some of the social issues that must be addressed to expand the scope of distributed computing. Section 5 extrapolates where distributed computing is heading in the future.

2 PVM 3 features

Since its public release in 1991, PVM has continued to evolve due to changing technology, and user feedback. This section describes the features in PVM 3. Throughout this report the term *virtual machine* is used to designate the logical distributed-memory computer, and *host* will be used to designate one of the member computers.

PVM supplies the functions to automatically start up tasks on the virtual machine and allows the tasks to communicate and synchronize with each other. A *task* is defined as a unit of computation in PVM analogous to a Unix process. Applications, which can be written in Fortran77, C, or C++, can be parallelized by using message-passing. By sending and receiving messages, multiple tasks of an application can cooperate to solve a problem in parallel.

PVM supplies routines that enable a user process to become a PVM task and to become a normal process again. There are routines to dynamically add and delete hosts from the virtual machine, routines to start up and terminate individual or groups of PVM tasks, routines to send signals to other PVM tasks, and routines to find out information about the virtual machine configuration and active PVM tasks.

If a host fails, PVM will automatically detect this and delete the host from the virtual machine. The status of hosts can be requested by an application, and if required a replacement host can be added by the application. It is still the responsibility of the application developer to make his application tolerant of host failure. PVM makes no attempt to automatically recover tasks that are killed because of a host failure, but this is a future research area. Another use of this feature would be to add more hosts as they become available, for example on a weekend, or if the application dynamically determines it needs more computational power.

Dynamic task groups are implemented on top of PVM 3. In this implementation, a task can belong to multiple groups, and groups can change dynamically at any time during a computation. Functions that logically deal with groups of tasks such as broadcast and barrier use the user's explicitly defined group names as arguments. Routines are provided for tasks to join and leave a named group. Tasks can also query for information about other group members. Several group communication functions are included in PVM 3.3 such as global max, global sum, and scatter/gather.

PVM provides routines for packing and sending messages between tasks. The model assumes that any task can send a message to any other PVM task, and that there is no limit to the size or number of such messages. While all hosts have physical memory limitations which limit potential buffer space, the communication model does not restrict itself to a particular machine's limitations and assumes sufficient memory is available. The PVM communication model provides asynchronous blocking send, asynchronous blocking receive, and non-blocking receive functions. In our terminology, a blocking send returns as soon as the send buffer is free for reuse regardless of the state of the receiver. A non-blocking receive immediately returns with either the data or a flag that the data has not arrived, while a blocking receive returns only when the data is in the receive buffer. In addition to these point-to-point communication functions the model supports multicast to a set of tasks and broadcast to a user defined group of tasks. Wildcards can be specified in the receive for the source and message "type" allowing either or both of these contexts to be ignored. A routine can be called to return information about received messages.

The PVM model guarantees that message order is preserved in the following sense. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

Message buffers are allocated dynamically. So the maximum message size is limited only by the amount of available memory on a given host. PVM includes routines to create and manipulate multiple message buffers. For example, a message can be forwarded by switching the receive buffer to the send buffer. The principle reason for using multiple message buffers is to allow libraries and graphical interfaces to interact with an application without interfering with messages the application is sending or receiving.

PVM 3 is designed to use native message passing routines when available within a multiprocessor. Messages between two nodes of a multiprocessor go directly between them while messages destined for a machine out on the network go to the user's single PVM daemon on the multiprocessor for further routing.

The Intel iPSC/860, Paragon and Thinking Machines CM-5 distributed memory multiprocessors have been integrated into PVM 3.3 so that their native message-passing routines are used for inter-node communication. In addition, Cray, Convex, DEC, KSR, Meiko, Hitachi, and IBM have announced they are supplying optimized PVM 3 vesions for their respective multiprocessors.

Shared-memory multiprocessors have also been integrated into PVM 3. The SGI Challenge series and multiprocessor SPARCs running Solaris, such as SPARC-10 and Cray Super Dragon, have ports in PVM 3.3 that use the underlying shared memory to move data between processes on the same host. Using this method data can be transferred up to an order of magnitude faster than using TCP between processes on the same host. Messages destined for another machine are routed through the PVM daemons.

As the popularity of PVM grows so does the need to administer and schedule the PVM jobs to get the best utilization of existing resources. Several groups around the world have been working on adapting existing distributed batch schedulers to be able to handle parallel PVM applications. The DQS package developed at Florida State University was the first to be able to use PVM, followed by the Condor package developed at the University of Wisconsin [1,2].

PVM 3.3 defines a clean interface between PVM and scheduler packages.

This allows organizations that wish to set up a single scheduler that controls a organization-wide virtual machine to do so. Their users could submit serial or PVM jobs to the scheduler with a suggested number of machines to use. The scheduler keeps track of which machines are idle and which are being used. The scheduler assigns tasks to the virtual machine and migrates them in a way that yields a high utilization of the available resources. The goal is to achieve a high utilization without impacting researchers who are actually typing on their workstations. Trials done at Florida State and Wisconsin have shown that utilizations over 85% are possible on configurations consisting of hundreds of machines.

It is often useful and always reassuring to be able to see the present configuration of the virtual machine and the status of the hosts. It would be even more useful if the user could also see what his program is doing – what tasks are running, where messages are being sent, etc. We have developed the PVM GUI called XPVM to display this information and more. XPVM combines the capabilities of the PVM console, a performance monitor, and a call level debugger in single easy to use X-Window interface. Using XPVM as a run-time monitor, the network traffic, host loads, and PVM activities inside an application can be easily monitored. XPVM can also be used to save trace files to be analyzed later for performance tuning.

3 Applications

Each Fall the PVM developers send out a survey to all the people who have obtained the PVM source from netlib over the past year. From this information we have found that industrial users of PVM include automotive, aerospace, chemical, computer, environmental, medical, pharmaceutical, and oil companies. Distributed computing is viewed by industry as a cost effective way to solve their computational problems.

The survey results indicate that distributed computing is very popular in universities around the world. PVM is used for teaching parallel programming, as a base for graduate student projects, and as a computational engine to assist professors in their research studies.

The reported applications that PVM is being used in is quite diverse including: meteorology, contaminant transport, combustion, molecular biology, 3D seismic migration, photochemistry, neurobiology, DNA sequencing, nuclear transit, medical imaging, hypersonic flow, ion channel currents, hydrodynamics, quantum chemistry and physics, protein structure, CAD tomography, volume rendering, etc. Several Computational Grand Challenge applications were included in the replies. One of the key observations from this list of uses is that distributed computing is not limited to "embarrassingly parallel" applications.

Users also reported using PVM inside more generic algorithms such as linear algebra, finite elements, multigrid, integer programming, interior point linear programming, grid generation, Navier-Stokes solvers, and combinatorial optimization. The development of general purpose math libraries in PVM, such as the ScaLAPACK project at ORNL, will allow more researchers to be able to exploit distributed computing.

While the majority of PVM users utilize a cluster of workstations, the survey shows that a growing number of users are exploiting virtual machines composed of a supercomputer with supporting workstations to do I/O. We have experimented with connecting multiple supercomputers together in a virtual machine and predict that future surveys will show Grand Challenge applications taking advantage of this capability of distributed computing.

The replies to the 1992 and 1993 surveys exceeded 150 and 300 pages respectively. We can only summarize the general findings in this section. To receive a copy of the surveys, you must send an explicit request to `pvm@msr.epm.ornl.gov` saying "Please send the 199[2,3] PVM User Survey results".

4 Social issues

The social problems in distributed computing involve both policies and administration. The complexities arise when distributed computing cuts across both intra and inter administrative domains. Multiple computer centers each with their own system administrators, accountants, and security personnel become involved. Access and use of local and remote file systems and I/O devices may be required by distributed applications but may be restricted by local policies. Bandwidth and access to remote databases are local decisions that can affect distributed applications.

One of the first concerns of local management when asked to make machines available to a distributed computing environment is the impact of the environment on internal priorities. While computers are often idle, they were purchased for a particular purpose by the organization. If they are not available when needed because of outside users, then there is a conflict between the external priorities of the distributed computing environment and the internal priorities of the organization's mission. The argument that they could reciprocate and accomplish their mission using the distributed environment may not be viable if the applications in question are not suited or have not been developed to work in a distributed environment.

Accounting is a complicated operational issue when a heterogeneous distributed network of computers and storage media are involved. First there is the issue of multiple organizations being involved, each with a potentially different charging algorithm and collection method. The user of the distributed environment does not want to receive 10 or 100 bills from different organizations because of how an automatic scheduler happened to distribute his parallel application. Part of the infrastructure of the distributed computing environment must be a consolidated and consistent accounting method.

Several operational issues are involved with scheduling. Does each organization manage the scheduling of jobs on their local machines or is there a master wide-area task scheduler? In the first case, tasks from one site might be distributed to another site, but their scheduling and priority would be handled locally. In the second case, questions arise about who controls the master scheduler. This person or group could potentially disrupt the operation of several organizations by a mistake or purposeful setting of the schedulers priorities.

The scheduler's task is to make good utilization of the entire distributed environment. A second task for the scheduler is to load balance parallel applications over a heterogeneous network with machine loads changing dynamically due to other users. This is a largely unexplored research area that must be addressed in the operation of the distributed computing environment.

Another operational issue involving scheduling is the types of machines jobs are sent to. Some machines will be personal workstations, while others will be shared resources like compute or storage servers at an organization. Different scheduling procedures need to be addressed for the personal workstations. A researcher's productivity may plummet if the scheduler keeps sending his machine new jobs every time he stops typing for a few seconds. But if the scheduler ignores this class of resource, then the overall utilization of the distributed environment decreases.

5 Wave of the Future?

The most common form of distributed computing utilizes 10 to 50 powerful workstations hooked together on a local area network like ethernet. Over the next two years we see this view changing as high performance wide area networks make it possible to tie together machines located at several collaborating institutions. And these machines may include MPPs with performance in the hundreds of Gigaflops.

There are reasons other than computational performance that make geographically distributed computing attractive. There are applications in business and the military where fault tolerance is of utmost importance. By using machines distributed over a wide area, the application can continue to run even if parts of the virtual machine are incapacitated by disasters such as lightning strikes and city-wide blackouts.

Another reason geographically distributed computing will become more popular is to utilize computational and data base information not locally available. For example, a climate modeling code may need to access a remote weather data base for input, compute on a collection of hosts (some local, some remote), and then write the results to another site for storage. Similarly, the computational resources required, for example, a Paragon for atmospheric calculations and a C90 for solar radiation modeling, may not be located at the same site, which again leads to the need for geographically distributed computing.

A less grandiose but more practical use for this form of distributed computing is in research collaborations between universities and also between universities and businesses. In this case researchers at several different sites are collaborating on a single computational problem. Each institution has some hardware dedicated to the project, but by pooling all the institutions' project resources, bigger more realistic problems can be solved. This scheme of distributed computing also promotes closer collaborations.

Network technology will be rapidly changing over the next few years. Network backbone speeds will reach and exceed a gigabit per second, and more gigabit LANs will appear. Initially several protocols will be available such as ATM, FCS, and HIPPI. In order to prepare for these coming changes, the PVM plan calls for the development of a high-speed interface layer in PVM. The idea is to add a smart routing layer into PVM that can determine the best or fastest method to move data between two tasks. Vaidy Sunderam has developed a prototype that switches to lower protocol layers when possible and showed potential performance improvements of 2-3 times

TCP [3]. The more general layer planed for PVM would do this as well as choose an alternate network if appropriate and available. For example, if some of the hosts are connected by ethernet and a high-speed network like HIPPI, then PVM may decide to move data between these hosts over the faster network using a high-speed, low-level protocol. The MPP port where PVM moves the data over the multiprocessor's built in network is an early prototype of this more general research topic.

6 Conclusion

PVM is the de facto standard in distributed computing. All the major computer vendors have announced that PVM will be available on their systems. Thousands of users are solving their scientific and engineering problems using PVM. And several production software packages have been converted to PVM and many more are in the process of being converted. PVM continues to evolve to meet the needs of users and to keep up with the changing technology. For example, as gigabit networks become more common PVM will be modified to exploit the full potential of this technology.

If distributed computing is going to jump to the next level, which involves geographically distributed, multi-organization resources, then several social issues must be addressed and solved. The most important problems involve scheduling, accounting, and support. The goal is to make it easier for users to exploit the resources made available by the world-wide computer network infrastructure.

Notes: The PVM source code, users guide, and reference manual are available by ftp from netlib2@cs.utk.edu in directory pvm3.

References

[1] Bricker, A., Litzkow, M., Miron, L.: Condor Technical Summary. CS technical report No. 1069 University of Wisconsin, (Jan. 1992)

[2] Green, T., Snyder, J.: DQS, A Distributed Queuing System. Preprint of SCRI technical report. Florida State University, (May 1992)

[3] Sunderam, V.: An Inclusive Session Level Protocol for Distributed Applications. *Proceedings – ACM Sigcomm 90 (Computer Communications Review)*, **20**(4), pp. 307-316, Philadelphia, September 1990.