The Use of PVM on a Large Campus Wide Netork

Graham E. Fagg and Shirley A. Williams

Department of Computer Science, Whiteknights, The University of Reading, Reading, UK, RG6 2AY. e-mail G.E.Fagg@rdg.ac.uk

Introduction

Within a shared, distributed network a typical workstation is idle 75%-90% of the time [1]. Using this spare capacity, via PVM, allows a powerful computing resource to be made available at apparently no cost. However using PVM on a shared and sometimes heavily utilized computing system gives rise to dilemmas not experienced by users of dedicated PVM systems. The use and operation of PVM systems on shared networks has to be carefully constrained to avoid unnecessary degradation to others running on that network. This paper will detail some of the problems discovered using PVM on the campus wide network, consisting of hundreds of machines, at Reading; how these difficulties were remedied and suggestions for future installations. Currently all solutions implemented have been constructed on top of PVM, using only standard PVM and UNIX system calls to aid portability and maintainability and as such **do not** require special system access such as *root*.

The network users view

Most computer networks are not homogeneous, even when the component computer architectures are of the same type. Some machines have special tasks such as mail handling, such machines may be unavailable to users, other machines in a network may be divided up into sub-networks and sub-domains that have different software configurations, leading to a need to use only certain sections of the network. In addition within a network there are political divisions and physical divisions. Political divisions are based on ownership of machines, this may be actual (a machine was purchased for a specific use) or imagined (a machine located in a particular room may lead to assumed ownership). Anybody wishing to use a system such as PVM should consult senior management before doing so. In any case utilizing remote machines so as to cause users of the machines to be unable to continue interactive work is unacceptable. Unobtrusive use of workstations must be a prime consideration, obtrusive use irritates other users and reduces the speed-up achieved by the PVM users.

System administrators view

To the system administrator the main concern is that the system function correctly, in terms of PVM this means:

- The computers do not become overloaded.
- The normal working of applications are not effected.
- The network traffic does not cause processes to fail due to excessive timeouts caused by constant packet collision on systems such as ethernet.

Hence there is a need for careful placement of tasks and regular checking to remove erroneous tasks. We have found PVM capable of locking a hundred workstations and causing much frustration among the user community.

Maintaining a large working system

Many users start their PVM system using a file containing a list of machines. In many instances a single set of PVM files centrally maintained will help, together with an advised host file of machines. If restricted access is required then exporting of the file system and the use of a special PVM user ID can prove to be an acceptable alternative.

Maintaining the availability of a PVM system on a large network requires attention. Individual machines may fail, be reset and power cycled, thus effecting the PVM system. To facilitate the maintenance of a PVM system we have developed a set of tools for starting PVM, maintaining, running, shutting down and tidying up.

- A start-up tool.
- Maintenance tool.
- Shutdown and tidy up tool.

Many users find that even using the same host file PVM starts up differing numbers of host each time initiated. This may be caused by many factors such as:

- the machines being too loaded and time-outs occurring,
- previous versions of PVM are still running or exited in a unsatisfactory way causing a restart to fail.

Most of these problems can be resolved by the removal of temporary files and elimination of erroneous processes. This is also true for maintenance and tidying up of PVM, thus the tools all use some common functions:

- **File store clearing**. Removal of `/tmp/pvmd` files and archiving of the PVM log files `pvml` so that certain re-starts are possible. Unwanted socket files are also removed on machines where this is necessary.
- **Process elimination**. This involves identifying the processes connected with PVM and removing the ones that are no-longer required.
- **Host table maintenance**. Certain hosts may crash and not re-boot correctly, while others may become unavailable for system operation reasons. This function restarts a PVM daemon on such hosts, if possible. (It checks the machine is alive before cleaning its filestore and adds it to the current machine configuration).

The use of these tool-sets makes maintaining PVM an almost trivial task, where as before we had frequently become involved in lengthy fault hunting.

Load balancing.

PVM attempts to spawn processes in a round-robin manner on the list of hosts in its current host table. On a dedicated network this is ideal, but on a loaded, very dynamic network this causes problems. Analysis of network loading showed that around 80% of our machines were less than half loaded almost continuously[2]. During the day the average load was some 20% higher, leading us to conclude that most users were doing mainly interactive work with few computationly intensive tasks. However some machines were continuously, excessively loaded.

Much of the application work we were undertaking fitted the master-slave paradigm (that is a master farming out work to some slaves, the slaves do some computation and then return their results to the master). Most applications require all the results to be calculated, the total execution time becomes approximately equal to the execution time of the slowest slave. If a task is given to an excessively loaded machine the whole execution time is lengthened.

The simplest solution to this was to omit heavily loaded machines from the host file used to start PVM, this does not allow for the dynamic use of machines. An alternative approach was for some algorithms to replicate the work of the slaves [3 and 4] and ignore the results from the slowest. Although successful in its aims, this method was not general and is considered unfriendly; in that loaded machines became more loaded and then their work was ignored. A practical solution was the use of `ru_spawn` which is a run-time load balancer that we have developed, taking the same arguments as `pvm_spawn`. The function `ru_spawn` polls the current list of hosts, requesting their current and past loading, a simple normalised polynomial loading is calculated, tasks are then spawned on the least loaded machines. This method is susceptible to some sudden computationly heavy tasks starting on a machine after spawning. The function `ru_gettid` automatically `nice` spawned processes to limit inconvenience to interactive users. Using `ru_spawn` in conjunction with a small amount of slave replication enabled us in a number of cases to achieve good speed-ups, however to obtain higher sustained performance we believe task migration is required.

Task and load migration

Task migration at a process or application level allows a more general approach to using the best machines in a cluster. Process level migration [5] needs changes to the operating environment outside the scope of this work, which aims to build on PVM.

Application level load migration can be implemented by the user, maybe using a suitable library, on top of PVM or UNIX. In simple terms it is where the actual tasks do not move but the data and hence the work load does. The simplest implementations are for data parallel computations of the master-slave type. The PVM example XEP is a good demonstration of load migration where data is farmed out to the current hosts, but a record is kept of which hosts complete their current work so they can be allocated more work. This means that the work is distributed to the fastest machines just as in models such as Piranha-Linda[6]. This form of parallelism is encouraged by the use of templates for application development. We have also developed a *delayed race model* where any slow tasks were restarted on spare machines and the two identical processes raced each other to produce the results first. This is used in conjunction with `ru_spawn`, so that each time a process is started it goes to the least heavily loaded machine.

Group support

The use of groups allows PVM to be used to implement applications that rely on certain topological interconnection of tasks. The misuse of groups, through lack of experience, frequently causes problems, such as:

- Application failure
- Degradation of performance

Application failure is often as a result of: incomplete start-up operations, failing to joining a group and waiting on a barrier. The introduction of functions performing extra checks removed most of these failures. Performance degradation is usually due to process communications within groups that are widely distributed. The introduction of a process placement scheme utilising extra information about actual physical networking improved overall performance. Initial empirical testing has shown promising results, but fuller long term testing are required before any concrete conclusions can be drawn.

The process placement system has also shown itself useful in the context of certain other group-like models such as hierarchical trees. These models are similar to the master-slave model, with the slaves being masters themselves. One of the problems encountered with this was that the second layer masters all started up their slaves on the same set of hosts, giving very poor performance. All of the lower level masters were concurrently accessing the main PVM host table. The process placement system allowed more even distribution without any comparable increase in start-up times.

Future requirements and work

There are many areas of the use of PVM that still need to be addressed. The area we intend to examine next is program correctness in terms of message ordering and consistency. In our experience once the message passing pattern is recognised the production of the application should be automatic. Using this premise we hope to create a set of proofing tools and code generators to perform this.

Conclusions

A simple set of tools can allow inexperienced users to maintain a large PVM system with relative ease. The use of load balancing through the process placement scheme can allow the unobtrusive and efficient use of PVM without complicating the users applications and still allowing them sufficient control. Without using these tools PVM users can cause much disruption to other network users unless they are continuously checking the system. This together with loss of performance and even failure of the applications due to lack of error checking can discourage its continued use which can only harm the user in the long term.

References

[1] M.Litzkow, M.Livny & M.Mutka., *Condor - A hunter of idle workstations*, Proceedings of the Eighth Conference on Distributed Computing Systems, San Jose, California (1988).

[2] G. E. Fagg, *To Load or Not to Load*, Internal report, Computer Science Department, the University of Reading, SEG/HE/94/4, (1994)

[3] S. A. Williams, P.C.H. Mitchell and G.E. Fagg, *A Cluster Computing Implementation of a Monte Carlo Simulation of a Particle Growth Mechanism*, Parallel Algorithms and application, 3 (1994) (to appear).

[4] G.E. Fagg and S. A. Williams, *Improved Program Performance with PVM*, Internal report, Computer Science Department, the University of Reading, SEG/HE/94/4, (1994) (submitted for publication)

[5] J.Casas, R.Konuru, S.W.Otto, R.Prouty & J.Walpole. *Adaptive Load Migration Systems for PVM*. Internal report. Department of Computer Science and Enginerring, Oregon Graduate Institute of Science and Technology, Portland, Oregon 97291-1000. (994).

[6] D.Gelernter & D.Kaminsky. *Supercomputing out of recycled garbage: Preliminary experience with Piranha.* 1992 International Conference on Supercomputing,pp417-427.ACM,ACM press (1992).