Taskers and General Resource Managers : PVM supporting DCE Process Management

Graham E. Fagg[1], Kevin S. London[1] and Jack J. Dongarra[1,2]

[1] Department of Computer Science, University of Tennessee, Knoxville, TN37996-1301

[2] Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN37831-6367

Abstract. We discuss the use of PVM as a system that supports General Process Management for DCEs. This system allows PVM to initialise MPI and other meta-computing systems. The impetus for such a system has come from the PVMPI project which required complex taskers and resource managers to be constructed. Such development is normally too time consuming for PVM users, due to the in-depth knowledge of PVM's internals, which are required to facilitate such systems correctly and reliably.
This project examines contemporary systems such as various MPIRUN systems and other general schedulers, and compares their requirements to the capabilities of the current PVM system. Current PVM internal operations are explained, and an experimental system based on the experience gained from the PVMPI project is demonstrated. Performance of some standardised plug-in allocation schemes that will be distributed with the new PVM 3.4 release are then demonstrated.
It is hoped that this project will provide users of dynamic meta-computing environments the "user controlled" flexibility that has previously been difficult to achieve without the user having to rely upon external third party scheduler and job control systems.

1 Introduction

PVM is one of a number of parallel distributed computing environments (DCEs) that were introduced to assist users wishing to create portable parallel applications [9]. The system has been in use since 1992 [1] and has grown in popularity, leading to a large body of knowledge and a substantial quantity of code accounting for many man-years of development.

The PVM system has proven itself to be very flexible in terms of both process and machine/node management, with user accessible library functions allowing complete user application control of the Virtual Machine and any other application executing upon it.

For many users, the default scheduling scheme and process control from the PVM console is sufficient. When more complex process control such as batch scheduling or run-time load balancing is required, the user is forced to either implement their own Resource Manager (RM) or rely upon third party software such as NQS, DQS, load-leveller, LSF or Far[2].

The implementation of a RM is non-trivial due to the state that it is required to maintain and the level of knowledge concerning the PVM internals that a user would need to produce a “correct” and reliable RM system.

The aim of this work is to illustrate a General Resource Manager (GRM) that is designed to be customised by a user so that do not have to concern themselves with PVM internals but instead can concentrate upon load-balancing , queueing and scheduling schemes that match their particular needs.

2 Requirements for Process Control and Resource Management

Many early cluster or meta-computing system users and administrators realised that the simple process allocation schemes offered on many single-user (fixed partition) MPP systems were barely adequate for the specialist machines and not complex enough to manage the dynamic newly emerging cluster environments with multi-user MPPs.

This lead to the early development of many Cluster Management Systems (CMS) which are typified by allowing users to submit jobs for execution under some form of centralised control. The standard features of such systems are:

Load Balancing: Jobs are sent to unloaded machines to improve performance and to minimise effects on other users.

Queueing: Jobs can be submitted to execute at a later time upon specialist or reserved resources. Usually multiple queues exist to separate long running, quick turn-around and interactive applications.

Experience has shown that many of these systems need to tailored to individual site needs, as well as support more complex features such as task migration which for non monolithic applications can be highly expensive in terms of additional resources required.

3 PVM 3.3

One of PVM’s primary advantages over many other systems is its ability to inter-operate across a diverse number of heterogeneous platforms with over 30 different types currently supported. To facilitate this portability, PVM utilises a daemon to interface between the user and any particular operating environment. The PVM user is thus insulated from the system dependent functions of each host architecture and is instead given a consistent core set of functions to manipulate a Virtual Machine that PVM creates. Through these core services users can monitor all aspects of the virtual machine and its applications.

3.1 Resource and Process Control

The PVM system consists of a number of hosts which each run a daemon *pvmd*. In the case of MPPs this daemon usually only runs on the front-end or service node and the daemon maintains a list of available nodes. Each daemon additionally holds a complete list of all hosts enrolled into the PVM system although only the master daemon is considered to hold the authoritative list.

The user library allows hosts to be added or deleted by sending either a *TM_ADDHOST* or *TM_DELHOST* message to the local daemon for the user application.

Each daemon maintains a list of the processes executing upon it (and its associated nodes), so there is no global list of processes thus reducing daemon to daemon messaging during fast process turn-over periods. Any process can start (*spawn*), signal or terminate (*kill*) any other process in the virtual machine by calling the user library which will send the appropriate message to the owning daemon. This daemon will then use local operating system services to perform the operation requested.

The task creation is performed by calling either pvm_spawn()/pvmfspawn() functions in a user application or the spawn command at the PVM console. This results in a *TM_SPAWN* request being sent to the local daemon. The local daemon would then decide on the allocation of resources using its own copy of the master host list, unless a *single* particular host was specified in the initial request. If two non-specific requests were sent from two different machines then the resulting process allocation would be determined by the previous requests that each particular daemon has already processed and would thus appear random. If an application needed to execute upon a set of machine $M_{1\ldots N}$, then N separate spawn requests would need to sent, unless this set could be specified by a filter based on the architecture classes available.

3.2 Fault Tolerance

An important feature of PVM is that it contains user library hooks to monitor the Virtual Machine configuration and to **notify** changes by sending user level messages as they occur. This allows careful programmers to produce fault tolerant applications, although facilities such as check-pointing are not included in the current public release. They are however provided by several vendors and research projects such as Co-Check[8], which is based upon the Condor system[6].

3.3 Additional Daemons

In PVM 3.3 certain process and host management functions could be passed to specialist daemons to reduce the load upon the normal PVM daemons allowing them to remain more responsive. Three special daemons types were offered:

1. Hoster daemon which intercepts *addhost* messages and starts new *pvmd*s.
2. Tasker daemon which is used to spawn tasks for the local *pvmd*s.
3. Resource Manager daemon which catches almost all the management function requests between the user library and the *pvmds* allowing it to alter the look and feel of the whole Virtual Machine.

These daemons are activated by calling an appropriate register function. As the Resource Manager intercepts messages from the user processes directly, they need to be spawned after it has registered, else they would not be aware of its existence. A common solution to this problem would be to have the Resource Manager start the Virtual Machine directly.

4 PVMPI and specialised taskers

The PVMPI project[3] required PVM to start MPI-1[7] applications to allow inter-application communication using PVM across different MPI_COMM_WORLDs even if they were running under different MPI implementations. The usual spawn fork/exec mechanism was not suitable for this, as each different MPI implementation relied upon different techniques to obtain their initial operating environment, such as the number and identity of each process. Since the user API was to remain unaltered, a method had to be derived to allow the user to specify a host group or *cluster* within a single spawn instruction.

The solution was a specialist set of taskers with local host lists that understood how to initiate processes under each different implementation of MPI.

Thus the PVM system became a set of tasker managed clusters within the virtual machine as shown in figure 1, allowing PVM to replace all the non-standard MPIRUN schemes.

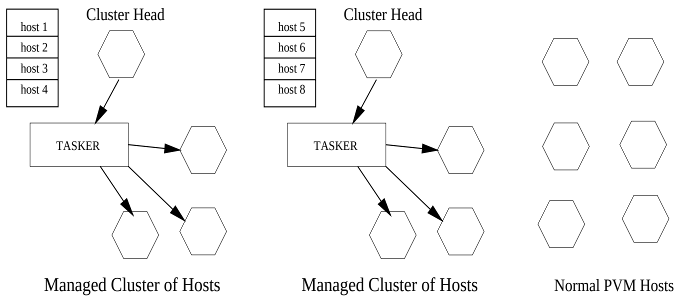

Fig. 1. Virtual Machine consisting of multiple host clusters and general hosts

Although the system suited the PVMPI projects needs, it could not be used as a general cluster management system for normal PVM applications as the processes started up by the taskers did not have the *correct* tids as excepted by the users library, due to the calling order within the PVM daemons. Other problems included inflexibility in the semantics of the current PVM API, made using a cluster only possible by spawning to the head of the cluster. Also manually starting up the taskers and giving them each a separate list was inconvenient and possibly error-prone. The solution was to keep the taskers (to avoid building their functionality into the already large PVM daemons) and migrate all the local host lists into a centralised Resource Manager which would pass spawn instructions to them when required.

5 General Resource Manager (GRM)

A Resource Manager can only be general if it meets most of the requirements listed in section two, including full user dynamic configuration. The GRM presented here mets these requirements by allowing users to slot in different modules for each of its the major functions.

The Resource Managers main functions are to control the addition and removal of hosts from the virtual machine, and the distribution of processes upon these hosts.

The GRM consists of seven main components as shown in figure 2. These being:

1. Relay mechanism to pass routine messages that are redirected to the resource manager from the user library back to the local PVM daemons. This handles the default process and host functions in the PVM system.
2. Host database: includes details about hosts such as associated processes, host clusters, current load, peak performance, memory free etc. This is used primarily for intelligent scheduling of job/spawn requests. Fault-tolerance of hosts is also handled here, as is automatic shutting down and bringing up of specific parts of the virtual machine.
3. Process database: includes application descriptions, requirements and run states. A list of processes constitutes a job.
4. Queue System: contains lists of jobs awaiting placement onto available hosts or specific named clusters.
5. Notification System: relays notification messages between daemons and user processes, using the data to update internal databases en-route.
6. Time Based Controller: A simple interface for scheduling job and host changes, such as removing clusters from the virtual machine during know busy interactive periods.
7. Check-point System: interface with the Co-Check system, that includes RM checkpoints so that if the system fails both the RM and the applications it is supporting either running or queued can be restarted from a known state.

5.1 Scheduler Operation

The Scheduler receives spawn requests and builds a host list that matches the requirements specified by the user request. If a request cannot be met immediately then the spawn details become a job which is queued. The choice of queue depends upon the priority of the request. A spawn request to a named set of hosts can now be completed in a single request instead of the N previously required. As the scheduler depends on a coherent database, change of state requests to the RM are processed first i.e. a spawn request would be delayed by a delete host or even a task exit message, hence the spawn request queue.

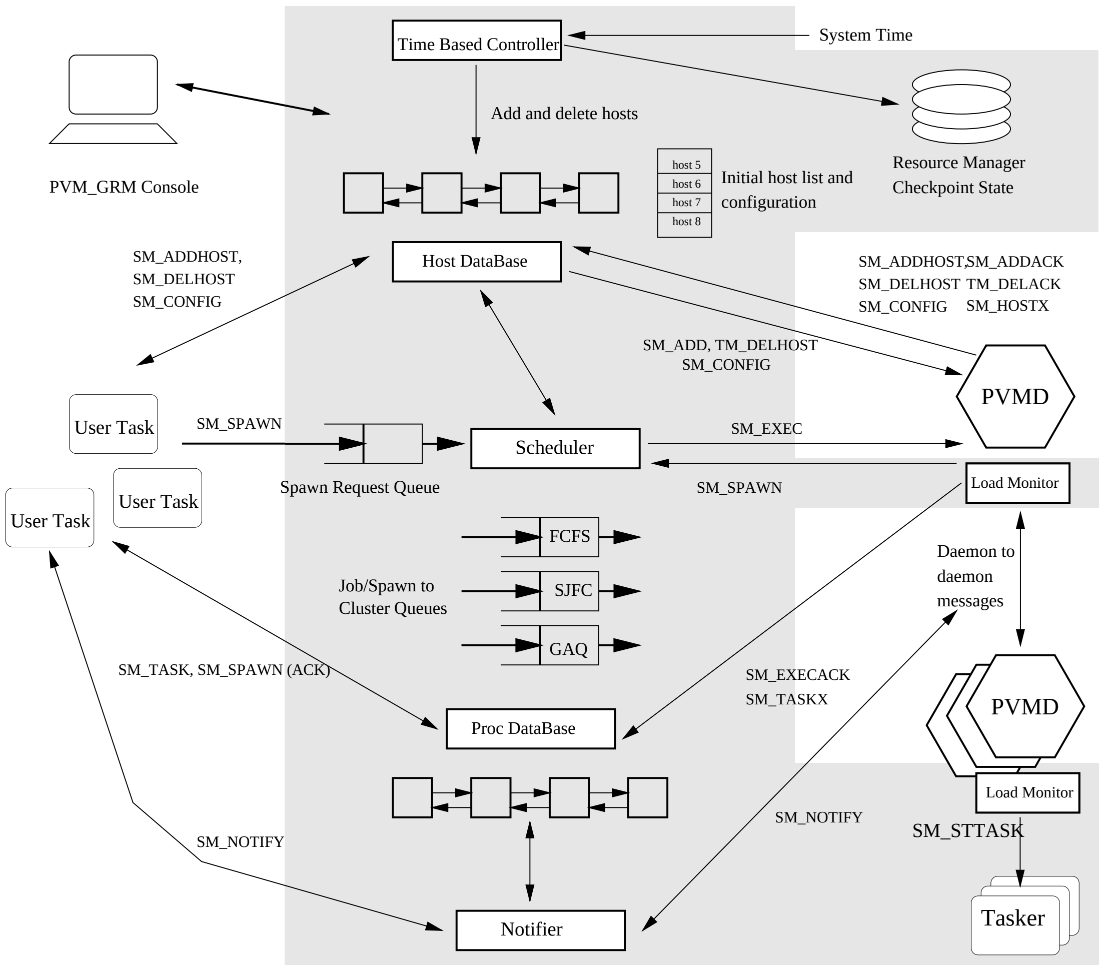

Fig. 2. Overall structure of the General Resource Manager (grey area) and its interaction with the rest of the Virtual Machine

Standard Scheduler The scheduler included with the standard distribution uses one of a set of best fit methods to allocate tasks to hosts if an open (non-specific) request is received. These rely on the load monitor system that has been added to the current PVM daemons, which on startup tests each host to measure their peak performance. The possible performance of each host is calculated as:

$$perf_{estimate} = \frac{perf_{peak}}{loading_{avg} + 1.0} \quad (1)$$

where $perf_{peak}$ is calculated as:

$$perf_{peak} = p_1 * int_{perf} + p_2 * float_{perf} + p_3 * double_{perf} + p_4 * memory\_bw_{perf} \quad (2)$$

and $loading_{avg}$ at time t is:

$$loading_{avg} = l_1 * loadavg_t + l_2 * loadavg_{t-\alpha} + l_3 * loadavg_{t-\beta} \quad (3)$$

The values $p_{1...4}$ and $l_{1...3}$ are user configurable defaults. α and β depend upon the particular operating system implementation of the remote statics daemons used, and are included so that the scheduler can take into account previous loading to help predict the future loading.

$p_{1...4}$ can be varied per job request by adding them to the spawn command so that specific application characteristics such as high memory or networking bandwidth requirements can dominate process placement.

Four basic systems are included with standard distribution:

1. Round robin: similar to the original PVM spawn except it uses a single host table.
2. Spare cycles: uses least loaded machines, irrespective of their peak performance.
3. Low load performance: uses peak performance as calculated in equation (1), but allows only one PVM task per host.
4. High load performance: may run multiple jobs on each host (even if not MP) if it is expected to be faster than executing on lightly loaded machines. Upper limit to number of jobs per host is configurable.

Figure 3 shows the effects of each scheme on a large scale Monte Carlo chemistry simulation[10] in terms of execution time on a forty seven host heterogeneous network containing five separate architecture classes. It is interesting to note that although load average alone produces good results on homogeneous networks[4] more detailed information is required for optimal performance on heterogeneous systems.

Additional user level modules for scheduling can be included by either adding modules to the GRM source code which already includes software hooks, alternatively users can register the modules as separate PVM processes using the well defined interface supplied.

6 Conclusions

A GRM has been presented that will be distributed with PVM 3.4[5]. This resource manager is designed to be customised by the user and will provide support for job control and scheduling. The supplied scheduling schemes have been shown to provide improved performance using a number of user controlled criteria. The ability to checkpoint application queues and alter the default operation mechanisms makes the GRM highly reliable and flexible.

References

1. A. L. Beguelin, J. J. Dongarra, A. Geist, R. J. Manchek, and V. S. Sunderam. Heterogeneous Network Computing. *Sixth SIAM Conference on Parallel Processing*, 1993.

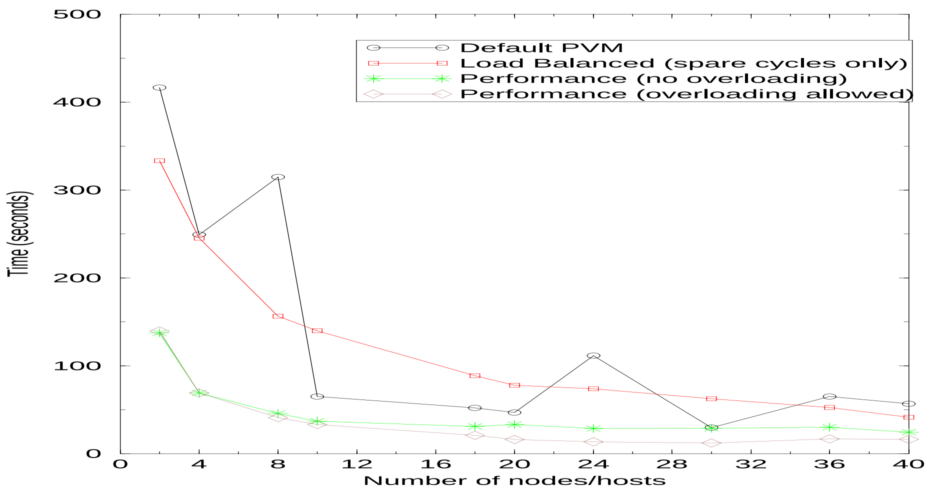

Fig. 3. default scheduling schemes performance on a heterogeneous network

2. Mark Baker and Geoffrey Fox. “MetaComputing: The Informal Supercomputer”, Proc. of CRPC annual meeting, Argonne National Laboratory, Chicargo, pp. 26, 1996
3. Graham. E. Fagg and Jack J. Dongarra. PVMPI: An Intergration of the PVM and MPI Systems To appear: Calculateurs Parallhles, Paris, June 1996. Also: Department of Computer Science Technical Report, University of Tennessee at Knoxville, TN-37996, April 1996.
4. G.E. Fagg and S.A. Williams. Improved Program Performance using a Cluster of Workstations, *Parallel Algorithms and Applications*, Vol 7, pp. 233-236, 1995.
5. G. Geist, J. Kohl, R. Manchek, and P. Papadopoulos. New Features of PVM 3.4 and Beyond. Proceeding of *EuroPVM 95*, pp. 1-10, Hermes, Paris, 1995.
6. M. Litzkow, M. Livny, and M. Mutka. Condor – a hunter of idle workstations. In *Proceedings 8th IEEE International Conference on Distributed Computing Systems*, pp. 104-111, June 1988.
7. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994. Special issue on MPI.
8. Georg Stellner and Jim Pruyne. Resource Management and Checkpointing for PVM Proceeding of *EuroPVM 95*, pp. 130-136, Hermes, Paris, 1995.
9. Louise Turcotte. “A Survey of Software Environments for Exploiting Networked Computing Resources”, *MSSU-EIRS-ERC-93-2*, Enginerring Research Center, Mississippi State University, Febryray 1993.
10. Shirley A. Williams and Graham E. Fagg. “A Comparison of Developing Codes for Distributed and Parallel Architectures”, Proc. of BCS PPSG *UK Parallel' 96*, pp. 110-118, Springer Verlag, London, 1996.

This article was processed using the LaTeX macro package with LLNCS style