WAMM: A Visual Interface for Managing Metacomputers

R. Baraglia, G. Faieta, M. Formica, D. Laforenza
CNUCE - Institute of the Italian National Research Council

Abstract

While good at providing users with low-level communication and management functions, PVM does not have adequate high-level tools to help configure the virtual machine and design applications. To alleviate these problems, we have developed WAMM, a graphical interface based on PVM. This paper outlines its main features.

1 Introduction

PVM [1, 2, 3] has become a very popular environment for the development of parallel applications to be run on a heterogeneous network of computers. While good at providing users with low-level communication and management functions, it does not have adequate high-level tools to help configure virtual machines and design programs. In this situation, the user often has to deal with UNIX commands to accomplish these operations; besides being complicated, this solution is not feasible when the system is composed by many nodes. For example, compiling a program for several machines, without suitable tools, usually entails writing source files on a local node, copying them onto remote hosts, and manually starting compilation on each machine: a great deal of time-consuming, error-prone operations.

To alleviate these problems, we have developed *WAMM*[1] (Wide Area Metacomputer Manager), a graphical interface based on OSF/Motif and PVM.

2 WAMM's features

Geographical view of the system. WAMM uses extended PVM hostfiles to allow a comprehensive description of the virtual machine. Hosts are grouped into subnetworks

[1]An exhibit of WAMM was held at the IBM-EMEA stand during the HPCN Europe '95 International Conference, May 3–5, 1995, Centro Congressi Milanofiori, Milano, Italy.

Figure 1: WAMM interface — snapshot during use

(LANs, MANs or WANs), following a tree structure. Each item in the tree is shown in Motif windows, using geographical maps for networks and icons for hosts (fig. 1); each graphic detail can be customized using standard X11 resource files. The user can move through the tree and *explore* all the resources by selecting push buttons on the maps: it is possible to zoom from a wide-area map to the single host of a particular site. A full, traditional list with Internet addresses for hosts is also available. The program does not impose any constraints on the number of nodes and networks in the system, nor on what type or where they are.

Virtual machine configuration. One of the first operations to perform in order to work with PVM is to set up the virtual machine. WAMM allows this to be accomplished without requiring the use of the PVM console. By choosing nodes with the mouse (via their icons or the address list), the user can verify if they are alive and reachable (WAMM calls `ping` and `traceroute`, and shows their output in separate windows), and add them to the virtual machine. Host removal and virtual machine *halt* are possible as well. All these operations can be carried out on group of hosts, thus simplifying tedious operations such as adding several hosts.

Also, WAMM is informed of any changes in the virtual machine configuration by using `pvm_notify` [2]. The notification mechanism is able to recognize any variations produced by host failures or external programs. For example, if hosts are added or removed using the PVM console, the modification is detected by WAMM as well and shown to the user.

Remote commands. It is often useful to execute commands on remote nodes: for example `uptime` or `xload` to find out the system load. Each host defined in WAMM's configuration file can be assigned a set of commands; they can be invoked through popup menus associated with each host's icon. Thus, the execution of a remote command is as simple as selecting a menu item and does not entail using `rsh` or `rexec`. In fact, execution requests are handled by "satellite" processes (`PVMTaskers`), spawned by WAMM at startup time on each node. Command output is sent back to the interface and shown in a status area. This way, `.rhosts` files are not needed on remote nodes and WAMM does not add any security weakness to the systems it is run on. For X11 programs, WAMM takes care of enabling local display to accept X11 connections from remote nodes. Thus it is very simple to open sessions on remote nodes: the only thing to do is to insert `xterm` (or similar terminal emulators) into the list of commands executable on remote nodes.

Remote compilation. One of the main features of WAMM is the ability to carry out the compilation of a program in parallel on remote machines. As for the remote command execution, a remote compilation facility has been designed to allow users to do most of their work on a local node.

Programmers can develop all sources on a local machine. To compile them on remote nodes, they only have to select hosts where they want to do the compilation, and call `Make` from the `Apps` menu. Using a dialog box, the local directory containing the source files and the related `Makefile` can be specified, along with any parameters needed for the `make` command. No restrictions are made on the type of source files to compile: they can be written in any language.

WAMM carries out some operations needed to compile an application. Specifically, all the source files are grouped into one file, in the `tar` standard format used on UNIX machines, and compressed using the `compress` command. A PVM task, which deals with the compilation (`PVMMaker`), is spawned on each selected node; the compressed file is sent to all these tasks. Now WAMM's work ends: the remaining operations are carried out in parallel by all `PVMMakers`, each on its own node. Each `PVMMaker` receives the compressed file and extracts sources in a temporary working directory; it then executes the UNIX `make` command.

Each `PVMMaker` notifies WAMM of all the operations that have been executed and sends back messages produced by the `make` process. These messages are shown in separate windows (one for each host) to let the user see in "real time" how the compilation is going on. The compilation process can be stopped at any moment with a menu command. If a node fails (for example due to errors in the source code), this does not affect the other nodes.

If the compilation is successful, the same `Makefile` that was used to carry it out can copy the executable files produced into the `$PVM_ROOT/bin/$PVM_ARCH` directory used by PVM as a "storage" for executable files. This operation can be carried out on each node.

At the end of the compilation the working directory is not destroyed (but it will be if there is a subsequent compilation). If necessary, the user can thus connect to the host, modify the source code, and manually start a new compilation on the same files.

Task management. WAMM allows PVM tasks to be spawned and controlled. Programs are executed by selecting `Spawn` from the `Apps` menu. A dialog box is opened where all parameters (e.g. number of copies, command line arguments, etc.) used by PVM to request the execution of the tasks can be inserted.

The `Windows` menu can be used to open a control window that contains some status information on all the PVM tasks being executed in the virtual machine. Output of processes activated with the `Output` option can be seen in separate windows and can also be saved in a file. If the output windows are open, new messages from the tasks are shown immediately. `Kill` and `Signal` are possible for all PVM tasks, including those not spawned by WAMM.

Data on tasks are automatically updated: if a task terminates, its status is changed. New tasks that appear in the system are added to the list, even if they were not spawned by WAMM. This feature is implemented by using the same `PVMTaskers` described earlier: each `PVMTasker` periodically queries its own PVM daemon to get the list of tasks running on the node. When variations from the previous control are found, WAMM is sent the relevant information. This implementation overcomes a limitation of the PVM's notification mechanism. In fact, with `pvm_notify` it is possible to find out when a given task terminates, but not when a new task appears in the system.

Using `PVMTaskers` is just one way to emulate a more complete `pvm_notify`. One drawback is that `PVMTaskers` have to be installed on each node indicated in the configuration file. An alternative is to let the interface itself request, from time to time, the complete list of tasks (`pvm_tasks` can be used to do this). This method does not need satellites but does have some drawbacks; specifically:

- data from all the daemons have to be transmitted to WAMM, even if there have not been variations in the number of tasks, compared to the previous control. In the first solution messages are only sent when necessary;

- if one of the nodes fails, then `pvm_tasks` waits until a timeout is reached. Some minutes may pass before the function resumes with the next nodes and sends the list of tasks to WAMM. The first solution, which is based on independent tasks, does not have this problem: if a node fails, then only its own tasks will not be updated.

There is a third solution, which exploits PVM's concept of *tasker*, introduced in version 3.3. It is possible to write a *tasker* in such a way that not only does it deal with the requests coming from `pvmd`, but it also notifies the interface of the activation and the termination of PVM tasks. This solution is the least expensive in terms of communications (periodic messages between the `PVMTasker` and the daemon of its node are eliminated too), but it is not without drawbacks. Infact, a program still has to be installed on each node and no information is available on PVM processes created before the *taskers* are registered. When developing WAMM we tried out all three solutions. We opted for the first as it was by far the best both in terms of network usage and control capabilities.

Performance issues. `PVMTaskers` periodically query their own `pvmd` to check the presence of new tasks in the system. The impact of this polling on the machine load is very little and can be furtherly reduced by increasing the delay between requests. The interface has to wait for two kinds of events: arrival of PVM messages and X11 events. The easiest way to solve this problem was to wait for X11 events and to perform periodical queries for the presence of new PVM messages. We noticed that this polling raised the machine load by about $0.7 \div 0.8$. This extra load can be reduced by increasing the polling delay but a better solution is needed: future versions of WAMM will solve this problem by waiting for events both from X11 and PVM.

3 Future developments

The current version of WAMM is only the starting point to build a complete metacomputing environment based on PVM. The version 1.0 of the package, along with some documentation material, is available at `ftp.cnr.it`, inside `/pub/wamm` directory. Some possible new features are outlined in the following.

WAMM currently lets users work only with the simple mechanisms implemented in PVM to control the nodes and task execution. For complex applications, more sophisticated mapping algorithms are needed. Such algorithms could be implemented in the module that activates the tasks.

Also, the current version of WAMM allows the activation of a task in trace mode: whenever the task calls a PVM function, a trace message is sent to the interface. At the moment WAMM does not make any use of the trace messages it receives; a future version should have a module which can collect these data, show them to the user either as graphics or tables, and save them in a "standard" format, which can be used in subsequent examinations via external tools, such as ParaGraph [6].

Finally, the ability to execute remote commands could be exploited to run the same operation on several hosts simultaneously. This type of functionality, not implemented yet, would solve some problems regarding the development and maintenance of PVM programs. For example, with one command an old PVM executable file could be deleted from a group of nodes. Manual deletion is not feasible if there are many nodes, it would therefore be advantageous to use the interface.

4 Related works

XPVM is a graphical console for PVM with support for virtual machine and process management. The user can change virtual machine configuration (by adding or removing nodes) and spawn tasks, in a way similar to WAMM's. With respect to WAMM, XPVM does not provide the same "geographical" view of the virtual machine and is probably more suitable for smaller systems. XPVM has no facilities for source file distribution, parallel compilation and execution of commands on remote nodes. However, it does include a section for trace data analysis and visualization, not yet implemented in WAMM.

HeNCE [4, 5] is a PVM-based metacomputing environment which greatly simplifies the software development cycle. In particular, it implements a system for source file distribution and compilation on remote nodes, similar to the one used by WAMM: source files can be compiled in parallel on several machines and this task is controlled by processes comparable with `PVMTaskers`. HeNCE lacks all the virtual machine management facilities provided by WAMM; in fact, the PVM console often has to be used. However, HeNCE was designed with different goals: the simplification of application development is mostly achieved by using a different programming model, with communication abstraction, rather than providing remote compilation facilities.

References

[1] J. J. Dongarra, G. A. Geist, R. Manchek, V. S. Sunderam. The PVM Concurrent System: Evolution, Experiences and Trends. Parallel Computing 20(1994)

[2] A. L. Beguelin et al. PVM3 users' guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Lab, May 1994.

[3] R. Baraglia et al. Network Computing: definizione e uso di una propria macchina virtuale parallela mediante PVM. CNUCE, Rapporto Interno C94-05, 1994

[4] A. L. Beguelin et al. Graphical Development Tools for Network-Based Concurrent Supercomputing. Proceedings of Supercomputing 91, Albuquerque 1991

[5] A. L. Beguelin et al. HeNCE: A Users' Guide, Version 2.0

[6] M. T. Heath and J. E. Finger. ParaGraph: a tool for visualizing performance of parallel programs. Oak Ridge National Lab, Oak Ridge, TN, 1994.