WAMM (Wide Area Metacomputer Manager): User's Guide
Version 1.0

Ranieri Baraglia, Gianluca Faieta, Marcello Formica, Domenico Laforenza

CNUCE - Institute of the Italian National Research Council
Via S.Maria, 36 - I56100 Pisa, Italy
Tel. +39-50-593111 - Fax +39-50-904052
email: R.Baraglia@cnuce.cnr.it, D.Laforenza@cnuce.cnr.it
meta@calpar.cnuce.cnr.it

Contents

1 Introduction

WAMM (Wide Area Metacomputer Manager) is a graphical interface with which to simplify the operations required to configure and manage a parallel virtual machine based on PVM. The key concept in WAMM is the geographical view of the system: hosts are grouped following a tree structure which represents their actual distribution in subnetworks (WANs, MANs and LANs). Each subnetwork is shown in a separate window, using geographical maps that simplify *exploration* of the resources.

WAMM comprises a set of functionalities to configure and manage the virtual machine; moreover, it is able to carry out remote compilation in parallel on several nodes, so it greatly reduces the time needed to develop metacomputing applications.

This guide describes the operations to install, configure and use WAMM. A more general description of the features of the interface can be found in [1]. Further information can be requested by e-mail at the following address:

`meta@calpar.cnuce.cnr.it`

The same address can be used to report bugs and suggestions.

2 Installation

2.1 Requirements

PVM must be available on *all* hosts which may belong to the metacomputer. Before installing the interface, check that PVM works correctly; in particular, PVM activation and host adding should work without any problems. Moreover, on each node, the directory containing PVM executables (usually `$PVM_ROOT/bin/$PVM_ARCH`) must be in the executables search path.

WAMM consists of the following programs:

`wamm`: the graphical interface itself;

`PVMTasker`: used by `wamm` to execute remote commands and to receive information on PVM tasks running in the system;

`PVMHoster`: used by `wamm` to control host insertion in the virtual machine;

`PVMMaker`: used by `wamm` to accomplish parallel remote compilation;

`PVMKill`: shell-script used by `wamm` to kill PVM daemons and remove temporary files on remote nodes.

Sources must be compiled using an ANSI C compiler (eg. `gcc`). `PVMTasker`, `PVMHoster`, `PVMMaker` and `PVMKill` must be installed on every node and only require the PVM library (version 3.3 or higher) to be compiled. `wamm` needs only to be installed on the *local* node (the machine normally used to start PVM) and requires the following packages to be compiled:

- PVM, version 3.3 or higher;
- X11, release 5 or higher;
- XPM[1], version 3.4e or higher;
- OSF/Motif, version 1.2 or higher.

WAMM has been developed and tested on the following systems:

- HP 9000/720 running HP-UX 9.01;
- Sun SparcStation 2 running SunOS 4.1.3;
- IBM RISC/6000 running AIX 3.2;
- IBM SP2 running AIX 3.2.

WAMM can also be compiled on DEC Alpha running OSF/1 version 3.2, however WAMM was not sufficiently tested on this platform.

2.2 Source distribution

The first operation to install WAMM is to get the sources and distribute them on all the nodes. Source files are available by anonymous FTP at:

```
ftp.cnr.it, /pub/wamm/wamm10.tar.gz
```

This package then needs to be copied onto every node (using `ftp` or `rcp`). For each copy, execute the following commands to unpack the distribution:

```
gzip -d wamm10.tar.gz
tar xvf wamm10.tar
```

All the files will be inserted into a directory named `wamm10`.

[1] `xpm` is a freely distributable library which simplifies the use of pixmaps in X11; it is available via anonymous FTP at `avahi.inria.fr`

2.3 Compilation

On each node, edit the `Makefile` in the `wamm10` directory, following the instructions inside it. On the *local* node execute the following commands:

```
make
make install
```

`PVMTasker`, `PVMMaker`, `PVMHoster` and `PVMKill` are installed in the directory normally used by PVM to hold executables; `wamm` is copied into the directory specified in the `Makefile`. At the end of the compilation, insert the contents of the `Wamm.ad` file into the `.Xdefaults` file, in the your home directory:

```
cat Wamm.ad >> ~/.Xdefaults
```

Alternatively, you can move the `Wamm.ad` file to the directory containing resource files for X11 applications, removing the `.ad` suffix:

```
mv Wamm.ad $XAPPLRESDIR/Wamm
```

On all the *other* nodes execute:

```
make slaves
make install-slaves
```

to compile and install `PVMTasker`, `PVMMaker`, `PVMHoster` and `PVMKill` in the right directory (in this case `wamm` is not compiled).

3 Configuration

To use WAMM, a configuration file with the description of the nodes that can belong to the metacomputer must be written; this file is read by **`wamm`** at startup time.

Even though this file in conceptually similar to a standard PVM *host file*, WAMM's configuration file contains a set of additional information that allow you to group the nodes into *subnetworks*. Subnetworks are structured like a tree (Figure 1).

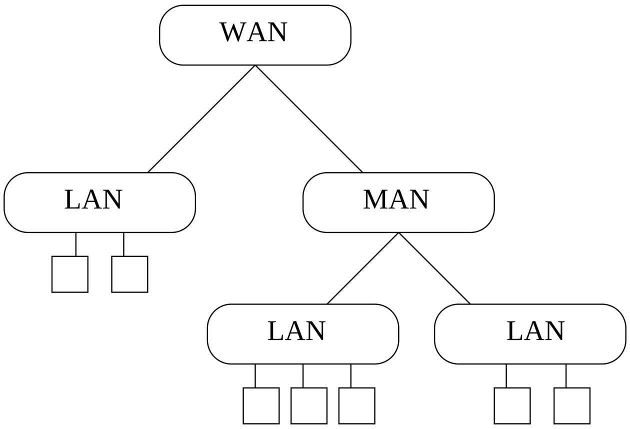


Figure 1: Structure of the network

- Leaves on the tree are single hosts;
- the parent of a host node is a LAN (Local Area Network) subnetwork;
- LANs can either be a child of a MAN (Metropolitan Area Network) subnetwork, or of the WAN (Wide Area Network);
- all the MANs are children of the WAN (the root of the tree).

Generally, the subdivision of the machines into subnetworks is stated on the basis of their real geographical distribution. For example, all the nodes belonging to the same computing center can be inserted into the same LAN.

In this case all the computing centers in a town should be inserted into the same MAN.

A configuration file contains one and only one WAN. If the WAN or a MAN contains a single host, a LAN node containing that host needs to be added to the tree.

Note that not all the hosts declared in the configuration file are added to the virtual machine: by using some options described later you can prevent some of these nodes from being inserted in the metacomputer at startup time. You can also change the metacomputer configuration dynamically, using the interface commands to add/remove hosts to/from PVM.

3.1 Configuration language

The configuration file is written in a simple declarative language. Nodes and subnetworks are declared using a syntax similar to the one used in C to define structures.

The following sections examine all the constructs available in the configuration language. Please refer to the configuration file `italy.net`, distributed with the package, which describes a complete metacomputer with over 60 nodes spread out in various Italian computing centers.

3.1.1 Language elements

The configuration language used by WAMM is based on the following elements:

- **comments**

  A comment is a string that starts with `#` and ends at the end of the line. The contents of a comment are ignored by `wamm`.

- **directives**;

  A directive is an instruction of the form:

  Name Arg1 Arg2...

  Example:

  ```
  PIXPATH /home/meta/pixmaps
  ```

Directives are used to control the configuration file parsing and to build the internal structures. Arguments can contain spaces but, in this case, they must be in double quotes (").

- **structures**

  A structure describes a host or a subnetwork. It takes the form:

  Type Name {
  Directive1 Arg1 Arg2...
  Directive2 Arg1 Arg2...
  ...
  }

  The *Type* of a structure can be `HOST` to describe a single node, or `LAN`, `MAN` or `WAN` to describe a subnetwork. The *Name* identifies the structure: it cannot contain blank spaces and must be unique inside the configuration file.

3.1.2 WAN

The root of the tree describing the metacomputer is specified using the following declaration:

`WAN` *WAN_name* {
...
...
}

The declaration of the WAN structure can contain a set of directives.

- `TITLE` *window_title*

  This directive allows you to specify the title of the WAN window. If this directive is not present, then the title of the window will be the name assigned to the WAN structure (*WAN_name*).

- `PICT` *file_name*

  `wamm` shows the information related to WAN, MAN and LAN subnetworks in separate windows. This directive allows you to specify a picture (in *xpm* format) to use as the background for the WAN window.

Generally, a geographical map representing the area comprising the network is used. If it is not possible to show the map (colors not available, file not found, etc.) `wamm` will show an error message.

- `MAN` *MAN_name X_pos Y_pos*

  Each `MAN` directive allows you to declare a MAN structure with name *MAN_name* as a child of the current WAN structure. `wamm` will show a pushbutton, used to open the corresponding window, on the map specified with the `PICT` directive. The parameters *X_pos* and *Y_pos* are the X and Y coordinates of the pushbutton.

- `LAN` *LAN_name X_pos Y_pos*

  Each `LAN` directive allows you to declare a LAN structure with name *LAN_name* as a child of the current WAN structure. `wamm` will show a pushbutton, used to open the corresponding window, on the map specified with the `PICT` directive. The parameters *X_pos* and *Y_pos* are the X and Y coordinates of the pushbutton.

For each `MAN` or `LAN` directive inside the WAN structure you need to declare a corresponding MAN or LAN structure with the same name, following the syntax that will be described in the following sections. There are no ordering constraints for the declarations: each structure can be declared either before or after its parent structure.

The following example shows a WAN structure for a geographical network comprising the Pisa metropolitan network and a computing center in Florence:

```
#
# just an example...
#

WAN italy {
   TITLE "WAN Italia"
   PICT italy.xpm
   MAN pisa 100 130      # Pisa
   LAN firenze 150 120   # Florence
}
```

3.1.3 MAN

Each MAN node is declared in the following way:

`MAN` *MAN_name* `{`
    ...
    ...
`}`

The structure can contain the following directives:

- `TITLE` *window_title*

  Allows you to specify the title of the MAN window. If this directive is not present, then the title of the window will be the name assigned to the MAN structure (*MAN_name*).

- `PICT` *file_name*

  Allows you to specify a picture (in *xpm* format) to associate with the MAN structure. Generally, a map representing the area comprising the MAN is used. If it is not possible to show the map (colors not available, file not found, etc.) **`wamm`** will show an error message.

- `LAN` *LAN_name X_pos Y_pos*

  Each `LAN` directive allows you to declare a LAN structure with name *LAN_name* as a child of the current MAN structure. **`wamm`** will show a pushbutton, used to open the corresponding window, on the map specified with the `PICT` directive. The parameters *X_pos* and *Y_pos* are the X and Y coordinates of the pushbutton.

Once again, for each `LAN` directive inside the MAN structure you need to declare a corresponding LAN structure with the same name.

Let us expand the preceding example:

```
#
# just an example...
#

WAN italy {
   TITLE "WAN Italia"
```

```
   PICT italy.xpm
   MAN pisa 100 130      # Pisa
   LAN firenze 150 120   # Florence
}

MAN pisa {
   TITLE "MAN Pisa"
   PICT pisa.xpm
   LAN cnuce 200 100  # Institute CNUCE-CNR
   LAN sns 280 55     # Scuola Normale Superiore
}
```

3.1.4 LAN

LAN nodes must be declared according to the following scheme:

`LAN` *LAN_name* `{`
    ...
    ...
`}`

The declaration can include the following directives:

- `TITLE` *window_title*

  This directive allows you to specify the title of the LAN window. If this directive is not present, then the title of the window will be the name assigned to the LAN structure (*LAN_name*).

- `HOST` *Host_name*

  Each `HOST` directive allows you to declare a HOST structure named *Host_name* as a child of the current LAN structure. **`wamm`** will show the informations related to the host inside the associated LAN window.

As in the preceding cases, for each `HOST` directive inside the LAN structure you need to declare a corresponding HOST structure with the same name. LAN structures do not have `PICT` directives, because the associated LAN windows contain the icons of the hosts.

Let us add three machines to our example network:

```
#
# just an example...
#

WAN italy {
   TITLE "WAN Italia"
   PICT italy.xpm
   MAN pisa 100 130      # Pisa
   LAN firenze 150 120   # Florence
}

MAN pisa {
   TITLE "MAN Pisa"
   PICT pisa.xpm
   LAN cnuce 200 100  # Institute CNUCE-CNR
   LAN sns 280 55     # Scuola Normale Superiore
}

LAN cnuce {
   TITLE "CNUCE-CNR"
   HOST calpar        # hosts in the LAN: these are user
   HOST miles         # defined names, not Internet addresses!
   HOST evans
}
```

3.1.5 HOST

The last type of structure in WAMM is the one used to declare hosts:

`HOST` *host_name* `{`
    ...
    ...
`}`

The name assigned to the structure (*host_name*) has no relation to the internet address of the machine, which is specified using the `ADDRESS` directive. The structure can contain the following directives:

- `ADDRESS` *host_address*

to specify the internet address of the machine. This address must be specified in the same format (complete name or nickname) as the one used by PVM. A common mistakeis to specify a complete name in the directive (eg. `pink.foo.org`) whereas PVM applications use the nickname `pink`.

- `PICT` *file_name*

  to specify a picture (in *xpm* format) to associate with the machine. Generally, an icon representing the architecture is used (some example icons are distributed along with the sources, in the directory `pixmaps`); the icon is shown inside the LAN window containing the host. Picture dimensions must be at most 80×60; overlappings are possible using larger pictures. If no `PICT` directive is specified, `wamm` tries to use, by default, the file obtained by appending the suffix `.xpm` to the name of the architectural class of the machine (see `ARCH` directive). If it is not possible to show the icon (file not found, colors not available, ecc.), `wamm` will show an error message.

- `ARCH` *arch_name*

  to specify the architetcural class of the machine. The name (*arch_name*) will be shown in an information row, on the right of each host's icon. Generally, the names used for architectural classes are the ones defined by PVM (eg. `SUN4`, `RS6K`, etc.). In the current version, WAMM does not use this information, but future versions may use them to execute operations on groups of hosts belonging to the same architectural class.

- `OPTIONS` *PVM_options*

  to specify the PVM activation parameters for the particular host. For example, you can insert the option `so=pw` which tells PVM to use `rexec` instead of `rsh` to start the daemon on this machine. Multiple options can be used, provided they are in double quotes and are separated by blank spaces. To use the flag `&`, which instructs PVM not to immediately insert the node in the virtual machine, it must be the first option in the string *PVM_options*.

- `INFO` *information*

  to associate additional information with each node. Such information will be displayed in a status row, on the right side of each host's icon.

- `CMD` *menu_item command*

  Each host shown in the LAN window has a popup menu associated with it; this menu allows you to execute various operations. This directive allows you to add a command to the menu. By selecting the item *menu_item* in the popup up menu, *command* is executed on the *associated* host. For example, to get the complete listing of all the processes running on a machine, use the directive:

  ```
  CMD "All processes" "ps -aux"
  ```

  In this way, by selecting the menu item `All processes`, the command `ps -aux` is executed on the associated node. Command output is shown in a message area inside the LAN window to which the host belongs. `CMD` cannot be used for X11 programs; they require the use of the `XCMD` directive, described later.

  To insert a separator in the popup, just use the `CMD` directive in the following way:

  ```
  CMD --- ---
  ```

- `XCMD` *menu_item command*

  `XCMD` directive is similar to the last one and is used for X-Windows programs. The requested arguments are: the text to insert in the popup menu and the command to execute. When the command is executed from the popup menu, its windows are shown on the same display used by `wamm`. Authorization to accept X11 connections from the host on which commands are executed is automatically requested by `wamm`. When the program terminates, the error code it returns is shown in the message area. For example:

  ```
  XCMD XLoad xload
  ```

Let us add to our example the definition for the machine named `calpar`:

```
#
# just an example...
#
```

```
WAN italy {
   TITLE "WAN Italia"
   PICT italy.xpm
   MAN pisa 100 130      # Pisa
   LAN firenze 150 120   # Florence
}

MAN pisa {
   TITLE "MAN Pisa"
   PICT pisa.xpm
   LAN cnuce 200 100  # Institute CNUCE-CNR
   LAN sns 280 55     # Scuola Normale Superiore
}

LAN cnuce {
   TITLE "CNUCE-CNR"
   HOST calpar        # hosts in the LAN: these are user
   HOST miles         # defined names, not Internet addresses!
   HOST evans
}

HOST calpar {
   ADDRESS calpar.cnuce.cnr.it   # Internet address
   PICT /home/meta/pix/HP720.xpm # icon
   ARCH HPPA                     # architecture
   INFO "HP 9000/720"            # various informations
   XCMD "NEdit" "nedit"          # remote commands: an editor..
   CMD --- ---                   # ... separator ...
   CMD "Uptime" "uptime"         # and "uptime" (system load)
}
```

3.1.6 PIXPATH

`PIXPATH` directive defines the default search path for the pictures declared using the *subsequent* `PICT` directives. It can be used in any point in the configuration file, but not inside the structures. For example:

```
PIXPATH /home/meta/pix
...
HOST calpar {
    PICT HP720.xpm
    ...
}
```

In this case, `wamm` searches for the file `/home/meta/pix/HP720.xpm`. If the `PIXPATH` directive is inserted more than once, the new path replaces the one declared with the previous directives. To cancel the effect of the `PIXPATH` directives, the following form can be used:

```
    PIXPATH ---
```

4 WAMM Activation

To start WAMM, execute the command:

`wamm` [*X11_options*] [*config_file*]

You can specify, as *X11_options*, all the optional arguments for X11 applications (a complete listing of such arguments can be obtained using the command `man X`). The optional parameter *config_file* is the name of the configuration file to be used. If it is omitted, `wamm` uses, by default, the file specified in the environment variable `WAMMNET` or, if this variable is not defined, the file `$HOME/.wammnet`.

When `wamm` is started, it executes the following operations:

1. the configuration file is read and parsed;
2. all the *xpm* files, specified with the `PICT` directives, are preloaded; this phase may take some time;
3. the main window, corresponding to the WAN structure, is opened;
4. if PVM is not running, it is automatically started using a temporary *host file* derived from the configuration file: all the hosts without the `&` option are added to the virtual machine. If PVM was already running, `wamm` only checks the current configuration of the system; in this case, the options specified in the configuration file, by means of the `OPTIONS` directive, are ignored;
5. a `PVMHoster` process is spawned on the local node;
6. a `PVMTasker` process is spawned on each node specified in the configuration file and belonging to the virtual machine.

The results of these operations are shown in the message area at the bottom of the WAN window. Upon completion of the initialization phase, you can use the interface via menu and buttons.

5 Working with WAMM

5.1 WAN Window

The WAN window is the "base" window of `wamm`. It is used to work on groups of hosts, which may even belong to different subnetworks, and to control the general running of the virtual machine. It is subdivided into three parts:

- **map** (at top left)

  This section is used to show the picture specified with the directive `PICT` inside the WAN structure. For each `MAN` and `LAN` directive contained in the WAN structure to define a child structure, a pushbutton is placed onto the picture. By selecting it with the left mouse button, the corresponding window is opened. The position of the button is specified by the coordinates given as parameters to the `MAN` and `LAN` directives; the label corresponds to the title of the window that is opened (see `TITLE` directive).

- **host list** (at top right)

  The list contains internet addresses of *all* the hosts declared in the configuration file. Many actions that are executables from the WAN window menu require one or more hosts in the list to be selected; you can select/deselect all the nodes by selecting the right item in the `Host` menu. You can select a group of contiguous hosts by using the left mouse button along with the SHIFT key. To select a group of non contiguous hosts, use the left mouse button along with the CTRL key. The list allows you to rapidly access the LAN containing a particular host by double clicking on it.

- **message area** (at the bottom)

  This is used to show control and error messages produced by `wamm` or by PVM. In particular, information related to specific *events* (host insertion/removal, etc.) are shown. For example, if a PVM task modifies the virtual machine configuration, a set of messages is printed in this area to inform you of the changes. The item `Clear Messages` in the `Misc` menu can be used to remove old messages.

5.1.1 Configuration management

One of the most frequent operations while working with a metacomputer consists in dynamically changing the configuration of the virtual machine. For example, it is frequently necessary to reinsert in the virtual machines some nodes that were deleted by PVM because they were temporarily unreachable.

Before adding one or more hosts, check their status:

1. use `PVM Check`, in the `Host` menu, to verify if the nodes are already in PVM. For each machine, selected in the list of hosts, `PVM Check` shows the current status (`not in PVM` or `OK!`) in the message area;

2. to verify if a host is reachable, use the command `Ping`, in the `Host` menu. In many cases, if the machine does not reply to ping, the problem is due to the network: `Traceroute`, in the same menu, can give further details. `Ping` and `Traceroute` use the standard UNIX programs `ping` and `traceroute`, which must be available on the local machine. Command output is shown in separate windows, one for each selected host;

3. if the hosts are reachable and are not in PVM, they can be added using `PVM Add`, in the `Host` menu. The result of the insertion is shown in the message area as soon as this information is available for *all* selected nodes. The operation can take several seconds; to avoid blocking the interface, it is executed in background. Therefore, PVM does not allow you to execute a new `PVM Add` before the old one is terminated; if this function is invoked while another one is running, an error message is shown in the message area.

If PVM has been started by `wamm`, host addiction (to the virtual machine) is performed using the activation flags specified through the `OPTIONS` directives. In particular, to add nodes with the option `so=pw` you need to supply the related password. In this case, the password is requested by `wamm` in a dialog box.

The insertion of a node often fails, producing the error message `already in PVM` even if `PVM Check` says that the host is not in the virtual machine. In the majority of cases, this is due to the presence of an old `/tmp/pvmd.<uid>` file on the node. In such cases you can try to "repair" the host by selecting the item `PVM Repair` in the `Host` menu, and then retry with `PVM Add`. `PVM Repair` executes the script `PVMKill` on all selected nodes. This script kills all your `pvmd` processes and deletes PVM files in the `/tmp` directory.

To remove one or more nodes from PVM simply select them and recall `PVM Remove` from `Host` menu. A confirmation line is shown in the message area.

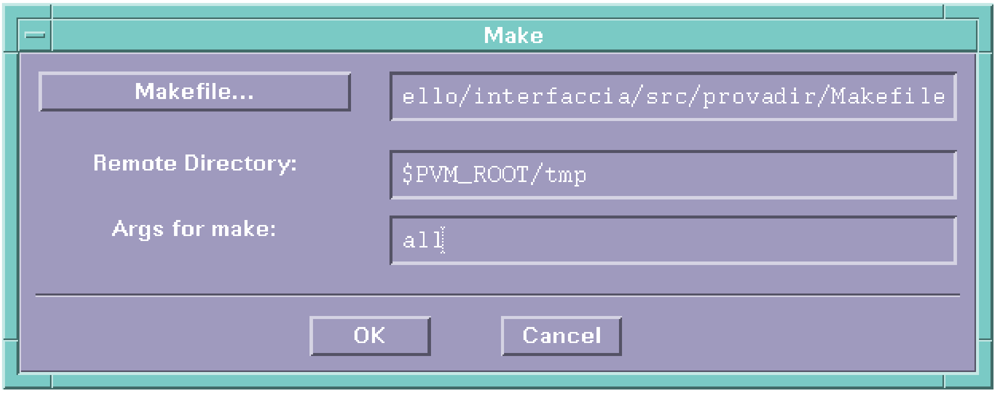


Figure 2: Dialog box for the compilation

5.1.2 Program compilation

WAMM provides you with a set of facilities to develop metacomputer applications. In particular, you can write sources for the whole application on the local node and then use the interface to distribute and compile them on the remote nodes. WAMM can compile applications written in any language (provided the related compilers are available on the right hosts); the programs to compile do not need to use the PVM library.

To use the remote compilation feature you need to define a *top level* directory containing the `Makefile`. Sources can be arranged in the same directory or can be placed in subdirectories.

When such a directory, containing application sources and the `Makefile`, has been set up, you can select the nodes to compile on and start the compilation process by picking up the item `Make` from the `Apps` menu in the WAN window. You will be shown the dialog box in Figure 2.

With this dialog box you can to specify some parameters needed for the compilation:

Makefile: complete path of the `Makefile` to be used for the compilation. By selecting a `Makefile` the *top level* directory is implicitly specified too. The `Makefile` can be specified using a file selection dialog box which is opened by the **Makefile** pushbutton;

Remote Directory: remote directory to use to temporarily store source files to be compiled. The current version of WAMM uses the directory `$PVM_ROOT/tmp` and you cannot use a different one;

Args for make: command line arguments to pass to the `make` process. This features allows you, for example, to specify a particular target (`clean`, `install`, etc.) or to use a different `Makefile` (`-f my-makefile`, etc.).

By selecting the **OK** button, `wamm` performs the following operations:

- all the source files are grouped into a single file, in the `tar` standard format used on UNIX machines;
- the resulting file is compressed using the `compress` command;
- a `PVMMaker` process is spawned on each node involved in the compilation; the compressed file is sent to all these processes.

The remaining operations are managed in parallel by `PVMMaker`s, each on its own node. Each `PVMMaker` receives the compressed file, extracts the source files in the temporary directory `$PVM_ROOT/tmp` and calls the `make` command with the command line arguments specified in the dialog box (Fig. 2).

The directory `$PVM_ROOT/tmp` is automatically created by each `PVMMaker`; if the directory already exists, its contents are erased *before* the compilation takes place. The files are held in the directory until the next compilation. It is thus possible to connect to the host, modify sources, and (if necessary) manually start a new compilation.

Each `PVMMaker` informs the interface of all the operations performed. Messages received by `wamm` are printed in the control window shown in Figure 3. The window shows the list of all compilation processes, both running and terminated. It is automatically opened when you start the compilation and can also be opened by using the `Make` item in the `Windows` menu of the WAN window. On each line the following information is shown: the name of the host, the state (**running** or **completed**) and the specific operation in execution (eg. source extraction, compilation, etc.). The `Cleanup` item in the `File` menu removes from the window those lines describing completed compilations.

In the example shown in Figure 3 the compilation was started on seven machines. For each of them it is possible to see the step in execution when the picture was "grabbed". For example, `astro.sns.it` has received its copy of

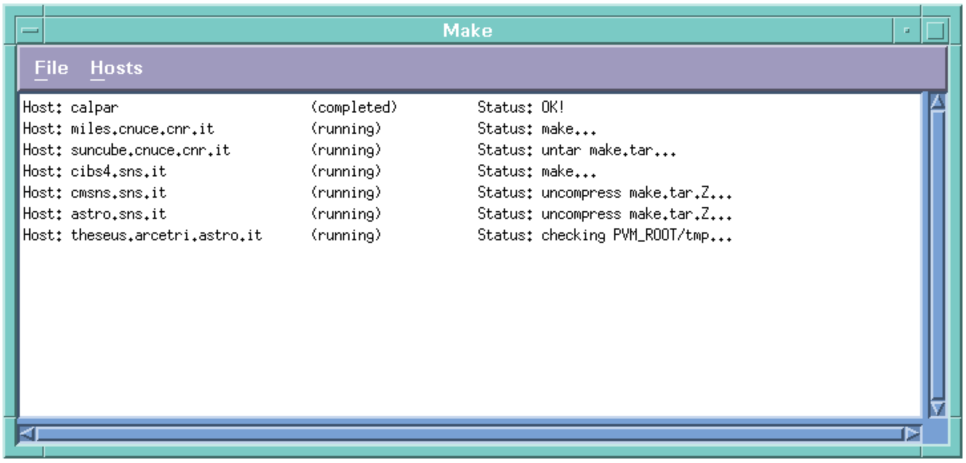


Figure 3: Control window for the compilation

the compressed file and is uncompressing it, whereas `calpar` has succesfully completed the compilation.

You can select one or more lines and, for each of these, open a window showing the output of the `make` command (`Stdout/Stderr` item in the `Hosts` menu); an output window can also be opened by double clicking on the related line.

You can stop the `make` process whenever you want, using `Stop` from the `File` menu. Note that the failure of the compilation on a node (for example, due to errors in the source) does not affect the activity on the other nodes.

If the compilation terminates succesfully, the same `Makefile` used to carry it out can be used to copy executable files into the directory

```
$PVM_ROOT/bin/$PVM_ARCH
```

used by PVM to hold executables. This operation would be executed on all nodes. The remote compilation feature can also be used to distribute one or more files (executables, libraries, data, etc.) on several nodes: simply insert a set of commands into the `Makefile`, which then copies such files into the appropriate directories, without running any compiler.

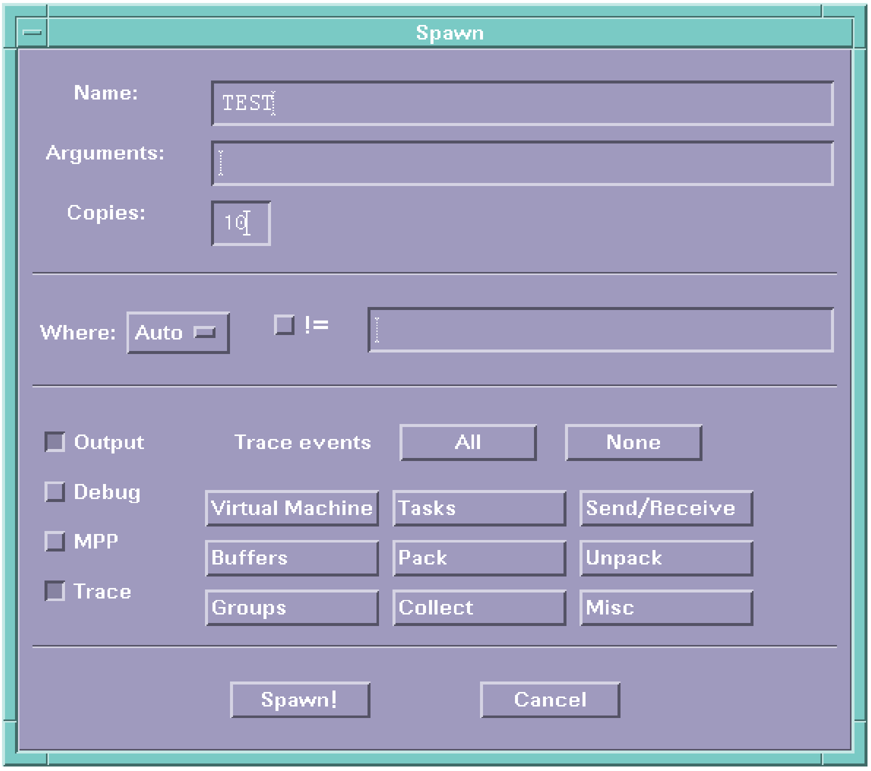


Figure 4: Control window to spawn processes

5.1.3 Process activation and management

The dialog box (Fig. 4) brought up by `Spawn` in the `Apps` menu can be used to execute PVM applications. This dialog box allows you to specify the activation parameters needed;

Name: name of the program to execute;

Arguments: command line arguments to pass to the program;

Copies: number of copies;

Where: allows you to choose the nodes on which to spawn the various copies of the selected program. By selecting one of the three items (**Auto, Arch, Host**) in the popup menu, hosts are chosen as follows:

- **Auto:** hosts to execute copies of the program on are automatically chosen by PVM on a round robin basis. It corresponds to the `PvmTaskDefault` flag;
- **Arch:** you need to supply a PVM architecture (eg. `SUN4`, `RS6K`, etc.) to execute the processes on. PVM selects target machines from the ones with the specified architecture and on a round robin basis. If the **!=** checkbutton is selected (`PvmHostCompl` flag), PVM choses the hosts with an architecture that is *not* the one specified. The **Arch** item corresponds to the `PvmTaskArch` flag;
- **Host:** you need to specify a node to execute the copies on. If the **!=** checkbutton is selected, PVM choses the nodes from all the machines except the one specified, using a round robin scheme. The **Host** item corresponds to the `PvmTaskHost` flag;

Output: by selecting this flag you can redirect the output from the spawned tasks back to the interface. You can thus see the output from the processes in separate windows which are updated as soon as possible. This feature is implemented by modifying the value of the `PvmOutputTid` variable;

Debug: allows you to execute the processes under the control of a debugger. Corresponds to the `PvmTaskDebug` flag;

MPP: starts the program on the front-end of a multiprocessor machine. Corresponds to the `PvmMppFront` flag;

Trace: starts the processes in **Trace** mode (`PvmTaskTrace` flag). For each PVM function you can specify whether it has to generate trace messages or not. The related messages are sent to `wamm`, but in the current version they are ignored.

When all the parameters have been set, the selected process can be started by clicking on **Spawn!**. For each copy of the process succesfully started, the following information will be shown in the message area:

- the name of the process;
- the TID;
- the name of the host on which it is running.

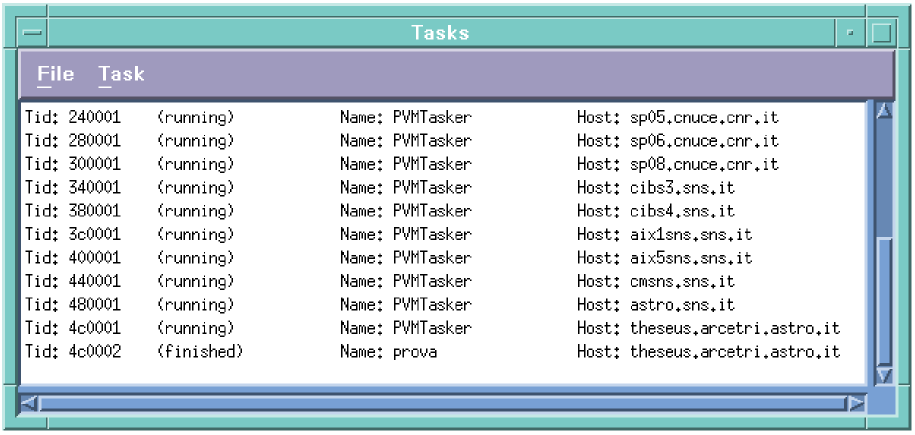


Figure 5: Control window for the tasks

If the spawning fails, an error message will be printed.

Tasks running in the virtual machine can be managed by using the window shown in Figure 5. This window is opened by selecting `Tasks` in the `Windows` menu. The window contains a list of the PVM processes. For each process the following information is shown: the TID, the status (**running** or **finished**), the name, and the address of the host it is running on. By default, this list does not include the service tasks (`PVMTasker`, `PVMHoster` and `PVMMaker`); these can be shown/hidden by changing the status of the `Show system tasks` flag in the `File` menu. Task status is automatically updated by `wamm` on the basis of information it receives from `PVMTasker` processes running on each node of the virtual machine. The `Remove old tasks` item in the file menu can be used to delete lines related to terminated tasks.

By double clicking on a process, you can open a window showing its output. You can also do this on a group of processes by selecting them and calling `Output` from the `Task` menu. This feature is only available for processes spawned with the **Output** flag (see Fig. 4).

This window also allows you to kill one or more processes: after selecting the desired processes, recall `Kill` from the `Task` menu. Likewise, you can send specific signals to the selected processes (`Signal` item in the `Task` menu).

5.1.4 Quitting the interface

You can exit by selecting one of the two items in the `File` menu of the WAN window:

Quit: terminates the program, asking for confirmation. Service processes are terminated (`PVMTasker`, `PVMHoster`, `PVMMaker`);

Halt: terminates the program by calling `pvm_halt()`. This causes virtual machine shutdown and all PVM processes are killed. Confirmation is requested.

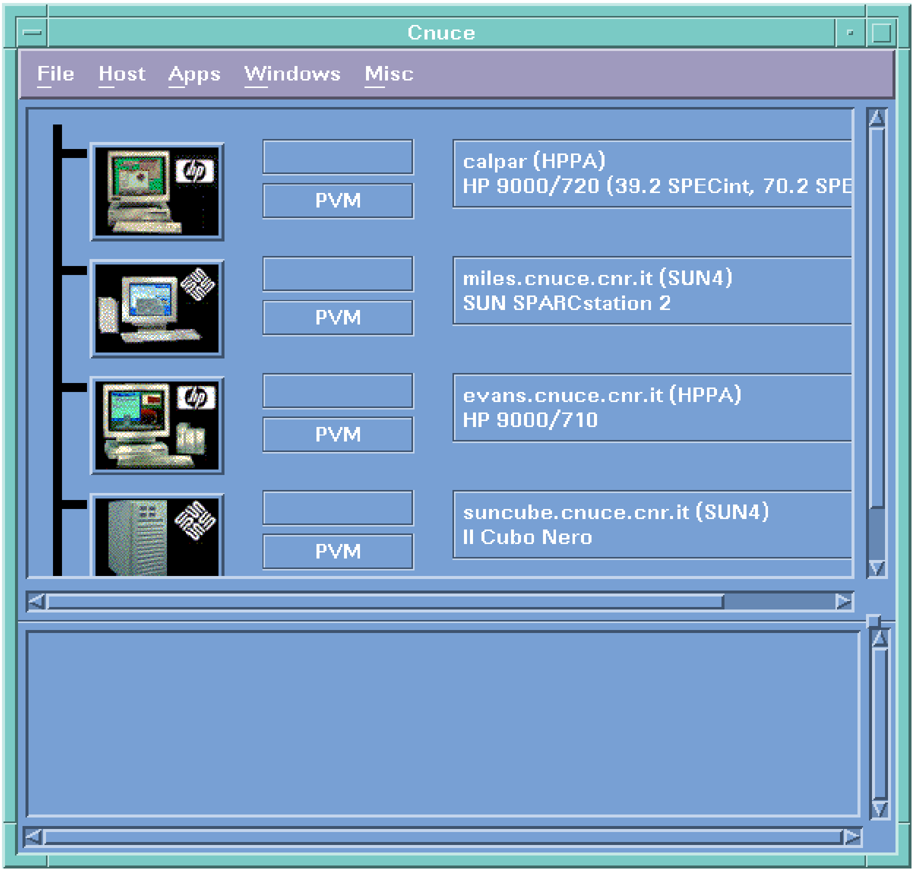
Service processes periodically check the status of `wamm`: if it is not alive (due to abnormal termination) they terminate too.

5.2 MAN window

The MAN window has a similar structure to the WAN window. The top left side contains the picture specified by means of the `PICT` directive, inside the declaration of the MAN structure. The top right side contains the list of all nodes belonging to the MAN. The bottom side contains the message area.

All the operations that are available from the WAN window can also be accessed through the MAN window; the only limitation being that visible hosts are only those belonging to the MAN.

5.3 LAN window



The LAN window is divided into two parts. The top side shows a segment of an Ethernet network; hosts specified inside the LAN structure declaration are connected to this network. Each host is represented through an icon. If the icons cannot be shown (colors not available, file not found, etc.) a label with the name of the machine will be shown. In the LAN windows, hosts can be selected by clicking the icon of each host with the left mouse button. All nodes can be selected/deselected using the appropriate items in the `Host` menu.

Two status flags are placed on the right side of each icon, stating whether the machine is in PVM and whether it has been selected. The PVM flag is updated automatically when the virtual machine configuration changes. On the right side of the status flags there is a box with the following information: the name of the machine, the architectural class specified through the `ARCH` directive and the additional information specified by means of the `INFO` directive.

The bottom side of the MAN window contains the message area. Error messages produced by WAMM as well as output of remote commands are shown in this area.

All the operations that are available from the WAN window can also be accessed through the LAN window; the only limitation being that visible hosts are only those belonging to the LAN.

5.3.1 Working on single nodes

Each icon (or label shown instead of it) has a popup menu associated with it, recallable with the right mouse button. This menu has two parts. The first one duplicates some commands, which are also accessible through the main menu:

PVM Add: adds the node to PVM;

PVM Remove: removes the node from PVM;

PVM Check: checks if the node is in PVM;

PVM Repair: "repairs" the node;

Ping: recalls the UNIX `ping` command for the associated node and shows the output in a separate window;

Traceroute: recalls the UNIX `traceroute` command for the associated node and shows the output in a separate window;

Make: starts the compilation on the associated node;

Spawn: spawns one or more copies of a program on the associated node;

Recalling an operation from a popup menu associated with a host, implicitly selects this machine as the *target* machine for the operation.

The second part of the menu contains the remote commands that you specified in the configuration file. Before executing a command on a remote node verify that this node is in PVM. Output from remote commands is shown in the message area at the *end* of their execution.

As regards X11 programs (specified by means of `XCMD` directives in the configuration file), windows are shown on the same display that shows `wamm`'s windows. To make this possible, the interface automatically requests authorization from the X server for the host executing remote commands. There are therefore some situations when `wamm` cannot enable X connections. For example, if `wamm` is running on a workstation and its windows are shown on the console of another workstation, you have to manually request the authorization on the second machine (the one showing the windows).

6 X resources

Each graphical element of an X11 program (*widget*) is labeled with a *name*, a *class* and a set of *resources* that define its appearance (eg. color, dimensions, font, etc.). These elements are arranged in a tree structure: the root is a widget whose class name generally derives from the name of the application.

In our case, each window (WAN, MAN or LAN) has a widget tree associated with it; the root always has class `Wamm`. The name of the WAN window is `wamm`; the name of the other windows (MAN or LAN) is the same as the name of the corresponding structures declared in the configuration file.

Each graphical element in the tree can be identified through a string that defines the root-node path. For example, the message area of the WAN window (a widget with name `status`) has the following path:

```
wamm.mainform.panedwin.statusSW.status
```

In many cases it useful to shorten the path by using the symbol `*`:

```
wamm*status
```

identifies the same widget.

The syntax to change the value of a resource is the following:

```
<widget path>.<resource name>: <value>
```

For example, to use the `fixed` font in the message area of the WAN window, the following line can be used:

```
wamm*status.fontlist: fixed
```

inside the `.Xdefaults` file in your home directory. There are alternative methods to specify the value of X resources; for further details consult the online manuals (`man X`)

The `editres` program can be used to obtain the complete listing of the widget used by `wamm`; this program can also be used to interactively modify the value of the resources without having to write a resource file.

7 Source structure

Each complex function in WAMM is achieved by an independent module. The modules are structured in a similar manner: each of them is coded in a pair of source files:

```
modXXX.c
modXXX.h
```

where `XXX` is the name of the module. Files that have `.h` suffix contain the declarations of the exported functions; files with `.c` suffix contain their implementation. Each module exports two *standard* functions, namely:

```
XXXInit()
XXXEnd()
```

that must be invoked to initialize and to release the allocated resources (data structures, etc.). The program does *not* use global variables (there are no `extern` declarations for variables). Access to private data (managed by a specific module) is obtained using a set of interface functions that are exported by the module itself.

Modules can be divided into three groups:

Applicative modules. These are high level modules that implement specific functionalities (eg. task activation, task management, remote compilation, etc.):

Spawn: manages task activation;

Tasks: controls and visualizes PVM tasks;

Make: coordinates remote compilation;

Internet: implements `Ping` and `Traceroute` features;

Parser: parses the configuration file.

Graphic modules. Include all the functions needed to realize the graphical interface:

WAN: manages the *look&feel* of the WAN window;

MAN: similar to the previous one for MAN windows;

LAN: similar to the previous one for LAN window.

Network modules. These are control modules which act as interfaces between the application and the underlying virtual machine.

PVM: executes the configuration operations of the virtual machine (adding, removing, checking, repairing) and collects the information from the remote nodes;

NetworkObj: contains all the information on the structure of the network, as specified in the configuration file.

The initialization of all modules is executed in the `main` function (inside `main.c`) at startup time.

References

[1] R. Baraglia, G. Faieta, M. Formica, D. Laforenza. *WAMM (Wide Area Metacomputer Manager): una interfaccia visuale per la gestione di un metacalcolatore.* CNUCE Rapporto Interno C95-23