Prospero Resource Manager 1.1 User's Guide

Santosh S. Rao

1 Introduction

The *Prospero Resource Manager* (PRM) enables users to run sequential or parallel applications on a network of workstations. Sequential jobs may be offloaded to lightly loaded processors while parallel jobs can make use of the collection of processors to achieve speedups. Parallel application programs are based on the message-passing model. Sequential programs need no modifications to run under PRM, while parallel programs are linked with a communication library that provides routines for sending and receiving tagged messages, broadcasting and global synchronization. This library exports two interfaces: a CMMD library interface (available to Connection Machine (CM-5) programmers) and a PVM interface (Version 3.3.5). These interfaces make it easy to use PRM as a platform for developing applications for the CM-5 or run PVM applications without having to modify any source code. PRM also supports terminal and file I/O activity by its tasks, such as keyboard input, printing to a terminal or access to files that may be on a filesystem not mounted by the host on which the task is running. Further, the components of an application may span multiple administrative domains and hardware platforms, without imposing the responsibility of mapping individual components to nodes on the user.

The end-user sees PRM application programs no differently than a sequential program executing on his or her workstation. Behind the scenes, PRM selects the processors on which the application will run, starts the application, supports communication between the tasks that make up the application, and directs input and output to and from the terminal and files on the user's workstation. At the application level, location transparency is achieved through dynamic address translation mechanisms that translate task identifiers to physical node addresses. PRM provides special I/O interfaces for terminal and file I/O, enabling a task to print to the terminal, accept keyboard input, or access files on remote filesystems.

PRM's resource allocation functions are distributed across three entities: the system manager, the job manager, and the node manager. The system manager controls access to a collection of processing resources and allocates them to jobs as requested by job managers. Large systems may employ multiple system managers, each managing a subset of resources. The job manager is the principal entity through which a job acquires processing resources to execute its tasks. The job manager acquires processor nodes from one or more system managers and initiates tasks on these nodes through the node manager. A node manager runs on each processor in the PRM environment. It initiates and monitors tasks on the node on which it is running.

2 New Features

2.1 Version 1.1

- Utility programs: `prm_jobs` for checking status of running jobs, `prm_stat` to query the status of a system manager and the nodes it manages, and `prm_shutdown` to gracefully shut down a PRM system.
- Use of ACLs: System manager uses an access control list to determine whether the host requesting resources is authorized to use nodes.
- Library routines : io_puts, io_gets, io_tputs, io_tgets for standard I/O to tasks, prm_create_task, prm_aq_rsc and prm_terminate_job for task creation and resource management.
- Local node manager not required: it is no longer necessary to run a node manager on the host on which a job is initiated.
- Unless configured with the `-r` option, the node manager will only execute programs residing in `$PRM_TOP/bin/$ARCH`.

2.2 Version 1.0

- Task migration : When a node makes itself unavailable to run jobs, tasks running on it are checkpointed and migrated to other nodes. Checkpointing support is provided by the Condor package developed at the University of Wisconsin. For `sun4` platforms, portions of Condor needed by PRM are available separately as the `checkpointing-extensions` distribution.
- Application debugging : By specifying debugging options, applications tasks can be started under a debugger. A front end transmits debugging commands to back end task-monitors that control program execution. Patches are provided that adapt the Gnu source level Debugger (gdb) to function as the task-monitor. For parallel programs, each target task is controlled by a separate task-monitor. Tracing and replay options enable the exact recreation of the sequence of events in a program's history.
- PVM 3.3.5 compatibility : Interprocess communication interface compatible with that of PVM Version 3.3.5 (including dynamic process groups).
- Customization of the terminal I/O task is no longer possible. All application specific initialization must now be handled by a user task that calls call the I/O routines from the PRM library.

3 Getting Started

PRM startup scripts and manager programs use two environment variables: `PRM_TOP` and `ARCH`. These must be set before starting up PRM programs. `PRM_TOP` is the top level of the directory hierarchy containing the PRM software and `ARCH` is the architecture of your host machine. Set `PRM_TOP` to the name of the top of the PRM directory hierarchy. Set `ARCH` to the value returned by the `arch` command. If this command is not available on your system, set `ARCH` to sun3, sun4 or hp700 on a Sun3, Sparc or HP-7xx host respectively. You may want to initialize `PRM_TOP` in your `.cshrc`, `.login` or `.profile` files. It is also useful to create an alias for the `arch` command. For example, C shell users on a Sparcstation can create this alias:

```
setenv ARCH sun4
alias arch 'printenv ARCH'
```

To install the software, see Section 4 below. Before running application programs, PRM's manager programs must be started up. A shell script, `PRM_startup`, is provided for this purpose. `PRM_startup` need execute only once to initialize the manager programs, that run as servers, executing in an infinite loop. If one of the managers dies due to a machine crash or other reasons, kill PRM using the script `PRM_kill` and restart it.

Two demo programs are provided in the directory `$PRM_TOP/demo`, with instructions for running them. See the file demo.doc for instructions on running the demo.

To run your own applications, you need to prepare:

1. executable files for the tasks (Section 6).
2. a job-configuration file (Section 7).

To run an application, invoke the job manager with the name of the job configuration file

```
jobmngr -j jcfile
```

The job manager's acquires nodes and initiates tasks on them transparent to the user, to whom the job appears to be executing on his local workstation.

4 Installation

1. Ensure that the environment variables `PRM_TOP` and `ARCH` are set.
2. Edit `$PRM_TOP/src/include/prm_config.h` to set site specific variables:

(a) If you do not have root privileges, skip this step. If you have root privileges on the machines on which you intend to run PRM, and you wish to install PRM as superuser, uncomment the line in `prm_config.h` that defines the macro `ROOT`.

(b) Edit the macro definitions for DefaultPrmBinDir and DefaultSmgrHost as described.

3. If you did not execute Step 2(a) above skip this step. Else, create a new group `prmsrv`. This is the group id under which the system and node managers will run.

4. If you are not installing PRM with application debugging capabilities skip this step. If you are, you must install the Gnu Debugger (`gdb`) first, and its source tree must be accessible in the steps that follow. First obtain the gdb-4.12 distribution (available at sites that distribute Gnu software, for e.g., `prep.ai.mit.edu`) and uncompress and untar it. We will refer to the complete pathname of the gdb subdirectory (gdb-4.12/gdb) in this distribution as `$GDBDIR`.

   (a) Copy the file gdb-patchfile in this directory to `$GDBDIR`.

   (b) cd to `$GDBDIR` and apply the patches:

   ```
   patch -c < gdb-patchfile
   ```

   (c) Compile gdb using the instructions provided. Ignore any error messages during linking. The executable file (`gdb`) produced in this step will be with further linked with some object files in Step 6.

5. cd to `$PRM_TOP`. Configure your PRM installation using the config script.

   ```
   config [debug GDBDIR=$GDBDIR] [ckpt]
   ```

   Specify as arguments to config those configuration options you have selected: debug if you executed Step 4 above; ckpt if you would like to provide task migration capabilities and have retrieved and unpacked the checkpointing extensions. Config creates the `bin, lib, obj` directory hierarchies, copies Makefiles into the `obj` directory, generates make dependencies and creates a working makefile, `Makefile.$ARCH` from the master makefile, `Makefile.master`.

6. Make and install the executables:

   ```
   make -f Makefile.$ARCH
   ```

   This compiles the managers and server programs and installs them in the directory `$PRM_TOP/bin/$ARCH`. It also compiles the demo programs, `ringsim`, `sort-master`, `sort-worker` and `ckpt_migr`.

`make clean` removes all object files and enables complete recompilation. If at a later time, you wish to install/remove either or both of the `debug` and `ckpt` options first use `make clean`, then repeat Steps 5 and 6.

This completes the installation procedure.

5 Starting up and shutting down PRM

Before startup, prepare an Access Control List (ACL) for the system manager. The ACL is specified in a file that contains one entry per line, specified as a symbolic name or an Internet address in the '.' notation. The system manager uses the ACL to determine whether a host is authorized to use its nodes. For example, a file containing the lines

```
pollux.usc.edu
128.125.72.4
128.9.160.0
```

tells the system manager that users on hosts `pollux.usc.edu`, `128.125.72.4` and any of the hosts on the `128.9.160` network are permitted to use its nodes. A default ACL file, `smacl.default` is provided in `$PRM_TOP/bin/$ARCH`. Edit this file to include hosts from which you would like to start PRM jobs.

Ensure that the environment variables `PRM_TOP` and `ARCH` are set and the PRM binaries directory `$PRM_TOP/bin/$ARCH` is included in your search path.

PRM is usually started using the shell script, `PRM_startup`. It can be gracefully shut down using the script `PRM_shutdown` and killed using another script, `PRM_kill`. Node managers and system managers can also be started individually. Both startup procedures described below.

5.1 Startup using the `PRM_startup` script

`PRM_startup` is a script that takes as input, the name of the host on which the system manager runs, and list of hostnames on which node managers are to be started. `PRM_startup` starts all node managers and a system manager. `PRM_startup` needs to run only once, since the manager processes continue running until they are killed by `PRM_kill` or by the Unix kill(1) command or a system crash.

`PRM_startup` can execute in interactive mode in which it prompts for hostnames, or non-interactive mode in which hostnames are specified in a file and the file name is passed as a command-line argument to `PRM_startup`. `PRM_startup` is executed as

```
PRM_startup [ -f Nodes-file ] [ other-options ]
```

where Nodes-file contains a list of host names on one or more lines and the system manager's host name is preceded by a `-s`. For example, a Node-Config file containing

```
zephyr.isi.edu
nii-server.isi.edu
```

```
bhs.isi.edu
-s zephyr.isi.edu
```

will start up node managers on the hosts `zephyr.isi.edu`, `nii-server.isi.edu`, and `bhs.isi.edu` and a system manager on the host `zephyr.isi.edu`. Unlike PRM Version 1.0, running a node manager on your local host is now optional. Other options to `PRM_startup` are described in the man pages.

`PRM_startup` creates a config file in the current directory: by default, a file `PRM_config.`*pid*, where *pid* is the process id of the shell that executed the startup commands. This file contains the list of hosts on which managers are running and their process ids, and is used as an argument to the `PRM_kill` and `PRM_shutdown` commands. From the list of node-manager hostnames, `PRM_startup` also creates a node-configuration file for use by the system manager.

5.2 Manual startup

1. On each host that will be part of the PRM environment start up a node manager in the background:

   ```
   nodemngr -s sysmngr_host_name  other-options &
   ```

   Write a node-configuration file for the system manager's use. This file should contain the list of nodes that the system manager will manage. The format of this file is:

   ```
   hostname-1   architecture-type
   hostname-2   architecture-type
     ..                ..
     ..                ..
   ```

   A sample node-config file can be found in `$PRM_TOP/demo/ncfile.example`. If sysmngr is started through the `PRM_startup` script, a node-config file is automatically created by the script.

2. Start up the system manager on host sysmngr_host_name.

   ```
   sysmngr ncfile &
   ```

Note: If a node or system manager is aready running, attempts to start another node or system manager on the same host will result in an error message of the form "can not bind socket". If so, use ps(1) to check whether a node manager, system manager, PRM server (such as term_iotask or file_iotask) or PRM user tasks are already running and kill them. Rerun the startup procedure.

5.3 Shutting down PRM

If PRM was set up using the `PRM_startup` script, use `PRM_shutdown` to shut down all managers. `PRM_shutdown` takes names of one or more `PRM_config` files as command line argument:

```
PRM_shutdown prm-config-file
```

`PRM_kill` is another script that accomplishes the same. It however kills node and system managers using signals, and is useful when some managers have died and PRM is in an inconsistent state.

```
PRM_kill prm-config-file
```

If PRM was started manually, or if you'd like to shut down part of a PRM configuration, use `shutdown` *host* to shut down a node manager running on host *host*. `shutdown -`*host* shuts down a system manager running on *host*. You may also kill a manager using the kill(1) command in Unix.

6 Writing and compiling PRM applications

A job running under PRM consists of one or more user-tasks, a server task that handles terminal I/O on behalf of the user-tasks. Server tasks for handling file I/O and collective communication are automatically created where necessary. In this release, all tasks are implemented as Unix processes. Server task programs are provided with the distribution. The user writes the application program and links them with PRM's libraries. User-level library routines for intertask communication and terminal and file I/O are documented in Lib_ref.doc. To run the application, the job manager requires a description of the job in a *job configuration file*. Section 7 describes the contents and format expected in job configuration file. Below are some general guidelines for writing applications:

- Application program modules must be written in C.
- Each source file that makes calls to PRM's communication library must include the header file comm.h. This file defines certain constants, and declares global names used by the communication library. If you call any of the CMMD library functions in your source file, CMMDlib.h must also be included.
- invoke the procedure

  `prm_init_task(argv);`

  before calling any functions in the communication library. argv is the list of command-line arguments passed to the main procedure. This requirement is relaxed if any PVM functions are called first, since PVM library functions call internally invoke `prm_init_task()`.

- use task identifiers 1..n to specify message destinations and sources. Task identifier 0 is reserved for server tasks.
- use io_printf() instead of printf() to print to stdout.
- use PVM's communication routines (version 3.3.5) just as you would in any PVM application program.
- if a PVM application program is already available, simply recompile and relink it as described below. Write a job-config file for the application and it is ready to run under PRM.

6.1 Compiling and linking application programs

gcc is the recommended compiler for compiling application programs. Use the following include directives in the compilation phase:

```
-I $PRM_TOP/src/include -I $PRM_TOP/src/include/$ARCH
```

Link with the following options and libraries:

- for generic PRM's communication functions, terminal and file I/O interface, PVM library routines and CMMD library routines:

  `-L $PRM_TOP/lib/$ARCH -lcomm -lardp`

- applications that must be migration capable should be linked with Condor's checkpointing library as follows:

  `-nostartfiles $PRM_TOP/lib/$ARCH/condor_rt0.o` (appl.obj.modules here)

  `-L $PRM_TOP/lib/$ARCH -lcomm -lardp -lckpt`

  Note that application object files must appear after Condor's runtime module, `condor_rt0.o`.

7 Format of the job configuration file

A job configuration file provides a description of the job to the job manager. Such a description includes the number of tasks to create, the path names of the executables, command line arguments to the tasks and some additional information described below. A job configuration file is essential for every job. Configuration information is specified as attribute-value pairs and takes the form:

```
attribute = < value >
attribute = < value >
....
....
```

More than one attribute-value pair may be specified on a single line. For example a config file containing

```
t  =  foo  a = -opt1  a = arg
n  =  3
d  =  /home/srao/prm/bin
c  =  false

s  =  zephyr.isi.edu
```

tells the job manager to start up 3 copies of the executable `foo` residing in the directory `/home/srao/prm/bin/$ARCH` with command line arguments `-opt` and `arg`, on nodes acquired from the system manager running on host `zephyr.isi.edu` and that the tasks cannot be checkpointed/migrated.

Note that the `$ARCH` component of the pathname was not specfied in the config file. PRM assumes that all executables reside in a directory whose name is the same as the architecture name of the host machine. Therefore you must place sparc executables in a directory `sun4` and HP pa-risc executables in a directory `hp700`.

Some examples of job configuration files can be found in `$PRM_TOP/demo`. A complete listing of attributes and the values they take are provided in the man pages for jobmngr.

Starting with Version 1.1, the default configuration of the node manager expects all application binaries to reside in `$PRM_TOP/bin/$ARCH`. This places restrictions on the set of applications available to a remote user. The PRM administrator can override the restriction by starting node managers with the `-r` option. As this option also allows remote users to transfer programs from remote locations and execute them on the node, the administrator is advised to consider security implications before running node managers with the `-r` option.

8 Running an application under PRM

Ensure that the environment variable `PRM_TOP` is set and the system and node managers are running. Invoke the job manager specifying the name of a job configuration file:

```
jobmngr -j jcfile  other-options
```

or

```
jobmngr -D -j jcfile  other-options
```

In the latter form, the `-D` option activates the printing of some informational messages. The manual page for jobmngr describes all the other options available.

8.1 Checking on job status

PRM Version 1.1 provides utility programs for checking status of PRM jobs. The script `prm_jobs` prints information about running jobs that belong to the user, including its job identifer, submitting host and the tasks belonging to the job with their current status. `prm_jobs` is a shell script that looks for job status files of the form `.prm_job.`{*jid*} in the user's home directory and then invokes the command `jobstat` to query job managers. `prm_jobs` optionally takes a username as argument.

`prm_jobs` *user*

prints information about jobs belonging to user *user*.

8.2 Checking on system status

`prm_stat` *host*

queries the system manager running on host *host* and prints the status of nodes managed by this system manager. Also printed is the list of jobs (if any) to whom one or more nodes are currently allocated.

9 Task migration

If built with checkpointing extensions, PRM Version 1.1 provides the option of automatic task migration through checkpointing. Migration is transparent to the user, and is initiated by the node manager (by checkpointing tasks) and carried out by the job manager in conjunction with another node manager. For a task to be migration capable, (a) its executables must be linked with Condor's checkpointing library, libckpt.a (b) the job configuration file must indicate that the executable has been so linked.

For linking with the checkpointing library, follow the instructions provided in Section 6. The `c` attribute in the job configuration file indicates that a task is checkpoint capable. Include a line

```
c = true
```

in the job configuration file if task executable is checkpoint capable,

```
c = false
```

otherwise.

In this release, checkpoint capability is the default if PRM has been compiled and installed with the `ckpt` option. The c attribute may be omitted from the configuration file if task executable is checkpoint capable.

10 Debugging PRM applications

By specifying debugging options, PRM tasks can be started under a debugger. A debugging setup consists of a command interface at the front end and task-monitors at the back end. The interactive front end enables the user to monitor and control program execution. At the back end each target task is controlled by a separate task-monitor that is co-located with the target. Task-monitors receive debugging commands from the front end and apply them to the target task. In this release, the Gnu Debugger (gdb) has been adapted to function as the task-monitor and interact with the front end. Use of gdb gives the task-monitor all the features of a traditional debugger.

An additional feature of the PRM debugger is playback debugging using traces. Instrumented communication functions enable the communication activity of the program to be captured in trace files. Using the information from these trace files, to programs can be replayed to exactly recreate the sequence of events in a program's history.

Debugging options must be specified in the job configuration file. To startup a task under the task-monitor, set the rd attribute in your job configuration file to true

```
rd = true
```

Similarly, `rt = true` writes message traces to a tracefile, `rl = true` logs contents of messages to a logfile and `rf =` *filename* sets the name of the tracefile to *filename*. For any of these options and to take effect, the job manager must be invoked with the -r option. Also note that tracing/logging are effective only for PVM's communication functions. If tracing is in effect and a trace file name is not specified in the job configuration file, the name of the trace and log files default to task_pid.trc and task_pid.log respectively.

When invoked under a task-monitor, a task is not started until the gdb command `run` is issued by the user. Each time the task stops at a breakpoint, the task-monitor prints a prompt to stdout and expects a debugging command from the user. The prompt indicates the task id of the task that is being controlled. In addition to all gdb commands, the new commands `_setmon`, `_all`, and `replay` can also be issued to the front end as described below.

When two or more tasks are running under task-monitors, the front-end designates one of the task-monitors as 'current task-monitor'. This task-monitor receives command inputs and has the capability of printing to the terminal (through the term_iotask). To change the current task-monitor, use the `_setmon` command. For example, the command

```
_setmon 3
```

makes task-monitor 3 the current task-monitor.

To apply the same debugging command to every task-monitor, prefix the command with an _all. For example,

```
_all break 100
```

broadcasts the `break 100` command to all task-monitors. Note that `_all quit` cannot be used. Quit each task-monitor separately.

When used in conjunction with tracing and logging, the "replay" command sets/toggles the replay mode.

```
replay on | off
```

If replay is on, task-monitor will force the task to obtain intertask-communication messages from the trace and log files, enabling a task to be replayed in isolation. Note that when replay mode is toggled, the task must be rerun from the beggining toggle to take effect.