Administration of SPK for Parallel Computation

SPK is designed to be executed on a cluster of computers loosely linked together by software known as the Parallel Virtual Machine (pvm). This document describes the system administration of SPK in the pvm environment. It is a companion to the document PVM Administration at RFPK [1]

The information in this document is targeted primarily toward the RFPK Software Team and associates and is specific to the computer systems and network installed in the RFPK Laboratory of the Department of Bioengineering of the University of Washington. RFPK is the Resource for Population Kinetics. Its work is supported, in part, by grant P41 EB-001975 of the National Institutes of Health (NIH) of the U.S. Department of Health and Human Services.

Table of Contents

Overview

The principal unit of work in SPK is called a job. It is created by a user when, with the help of a graphical tool called the MDA, she combines a model with data and various parameters and submits it via the internet for processing.

Before the job can begin its computational phase, it first travels to a server known as the Application Server for Population Kinetics *(aspkserver)*, where it is compiled by the ASPK compiler to create three C++ programs, called *spkjob*, *spkpop* and *spkind*. The job is then queued for execution.

The next step in the history of the job is to be selected for processing on a group of computers harnessed for parallel computation by software called the Parallel Virtual Machine (pvm). We refer to each of these computers as a *node* or a *host* and to the collection of computers as the *cluster* or the *pvm*.

One node of the pvm is distinguished as the *head node*. In terms of the SPK architecture, this is the Computational Server for Population Kinetics *(cspkserver)*.

On cspkserver, the three programs output by the ASPK compiler are further compiled and linked to the SPK library to create an executable binary for spkjob, for spkpop and for spkind.

Spkjob is then executed on cspkserver itself. It utilizes pvm functions to spawn spkpop as a subtask. Which node spkpop actually runs on is determined by load distribution algorithms in pvm.

Spkpop then performs an iterative computation with the goal of generating the results specified by the user. With each iteration, spkpop spawns *n* instances of spkind, where *n* is the number of individuals in the population being modeled. Which node a particular spkind runs on is determined by pvm.

Spkjob, spkpop, and the *n* instances of spkind all run as independent processes, under the linux or unix operating system. The parent process of spkjob is the SPK runtime daemon, which executes continuously on cspkserver. The parent process of spkpop is a pvm daemon running on the node to which it was assigned. Similarly, each instance of spkind is the child of a pvm daemon running on its node.

On every unix or linux host, parent/child relationships create a single process hierarchy. This hierarchy does not extend beyond that host, however, to other hosts in the cluster. With pvm, there is a second hierarchy, maintained by communication between daemons, which does span the boundaries between hosts. In pvm parlance the programs running in the hierarchy are referred to as *tasks* to distinguish these relationships from those between linux/unix *processes* and the hierarchy is the *pvm task hierarchy*.

SPK uses two channels for inter-task communications:

1. *PVM message passing*, using functions provided by a pvm. This communication channel is used for pvm event messages and application log messages.
2. *Shared file hierarchy*. The ubiquitous Network File System (NFS) works well for this purpose, although nearly any other distributed file system can be used in its place. This communication channel is used for the communication of application inputs and outputs.

There is no system administration required for pvm message passing, beyond installation and configuration of pvm as described in PVM Administration at RFPK [2]. Spkjob and spkpop set up the message channels and pvm does the rest automatically.

The shared file hierarchy is created and maintain entirely by linux/unix system administration functions. Here are some of the features of the file hierarchy:

1. The files actually reside on aspkserver, both because of its large disk capacity and its tape backup system. The aspkserver uses NFS to *export* this hierarchy.
2. All the nodes of the pvm *mount* the file hierarchy as a filesystem of type *nfs*.
3. The directory tree looks like this:

```
/usr/local/spk/share/working/spkprod
............................/spktest
....................../log/spkprod
......................../spktest
....................../include/spkprod
............................/spktest
....................../arch/LINUXI386/lib/spkprod
......................................../spktest
........................./LINUXX86_64/lib/spkprod
........................................../spktest
....................../arch/LINUXI386/bin/spkprod
......................................../spktest
........................./LINUXX86_64/bin/spkprod
........................................../spktest
```

4. The purpose of the key directory nodes is as follows:
   a. *working:* the hierarchy of working directories goes here. Spkjob creates a directory for the job and spkpop creates a subdirectory for each instance.
   b. *log:* message log files go here. A message log file consists of a sequence of messages in the order that they were created by spkjob or received by spkjob from spkpop or spkind. Each message has a header containing a

timestamp, job-id number, program name and, in the case of instances of spkind, instance number.

c. *include:* `.h` files to be included with the source generated by the ASPK compiler.

d. *arch:* binary executables and libraries for each of the processor architectures present in the pvm.

Group Configuration for Aspkserver and All PVM Nodes

The following configuration must be done to all nodes, including the head node, as well as for aspkserver. As the root user on each node execute the following commands:

```
/usr/sbin/groupadd -g 444 -r spkshare
/usr/bin/gpasswd -a pvm spkshare
```

Additional Group Configuration to Selected Nodes

It will be convenient to add developers to the spkshare group on aspkserver and cspkserver. As root, on each of these servers, execute the following commands:

```
/usr/bin/gpasswd -a alan spkshare
/usr/bin/gpasswd -a honda spkshare
/usr/bin/gpasswd -a jiaji spkshare
/usr/bin/gpasswd -a watrous spkshare
```

/usr/bin/gpasswd -a alan spkshare

NFS Configuration of aspkserver

In the following, it is assumed that the *nfs* and *portmap* services are running.

1. *Open a terminal window on aspkserver and become root.*
2. *Make the directory hierarchy:*

```
mkdir -p /usr/local/spk/share/working/spkprod
mkdir -p /usr/local/spk/share/working/spktest
mkdir -p /usr/local/spk/share/log/spkprod
mkdir -p /usr/local/spk/share/log/spktest
mkdir -p /usr/local/spk/share/include/spkprod
mkdir -p /usr/local/spk/share/include/spktest
mkdir -p /usr/local/spk/share/arch/lib/LINUXI386/spkprod
mkdir -p /usr/local/spk/share/arch/lib/LINUXI386/spktest
mkdir -p /usr/local/spk/share/arch/lib/LINUXX86_64/spkprod
mkdir -p /usr/local/spk/share/arch/lib/LINUXX86_64/spktest
mkdir -p /usr/local/spk/share/arch/bin/LINUXI386/spkprod
mkdir -p /usr/local/spk/share/arch/bin/LINUXI386/spktest
mkdir -p /usr/local/spk/share/arch/bin/LINUXX86_64/spkprod
mkdir -p /usr/local/spk/share/arch/bin/LINUXX86_64/spktest
cd /usr/local/spk/share
chgrp -R spkshare .
chmod -R 2775 .
```

3. *Export the directory hierarchy.*

   Edit the file /etc/exports, appending a line *for each node in the pvm*:

   ```
   /usr/local/spk/share   cspk(rw, sync, no_root_squash)
   /usr/local/spk/share   jambutty(rw, sync, no_root_squash)
          .
          .
          .
   /usr/local/spk/share   rose(rw, sync, no_root_squash)
   ```

   Activate the list by executing the following command:

   ```
   /usr/sbin/exportfs -a
   ```

NFS Configuration of the PVM Nodes

Apply the following procedure to each node of the pvm, including the head node.

1. *Open a terminal window on the node and become root.*
2. *Edit the hosts table.* Make sure that there is a line in the `/etc/hosts` file for the IP address of aspkserver. If such a line is already there, but the name *aspkserver* is not among the aliases for the address, add it to the line.

   ```
   192.168.1.2      aspkserver
   ```

3. *Edit the filesystem table*. Add the following line to `/etc/fstab`:

   ```
   aspkserver:/usr/local/spk/share  /usr/local/spk/share  nfs rsize=8194,wsize=8192,t
   ```

4. *Create a node on which to mount the filesystem:*

   ```
   mkdir -p /usr/local/spk/share
   ```

5. If *aspkserver* has not yet been configured to export the directory hierarchy to this node, do it now .
6. Mount the filesystem in order to check your work. Normally, the filesystem will be mount automatically, each time that the node reboots.

   ```
   mount /usr/local/spk/share
   ls -l /usr/local/spk/share
   ```

Environment Variables and File Creation Mask

As root on cspkserver, add the following lines to the end of `~pvm/.bashrc`:

```
SPK_SHARE=/usr/local/spk/share
PVMHOSTFILE=$SPK_SHARE/pvmhosts
export SPK_SHARE PVMHOSTFILE
```

```
PATH=$PATH:$SPK_SHARE/arch/bin/$PVM_ARCH/spktest
umask 0002
```

Building and Installing the Prototype

The prototype must be built and installed both for the LINUXI386 architecture and the LINUXX86_64 architecture.

LINUXI386 architecture

In the following, all of the steps except the last one are concerned with configuration and verification of the configuration. The first time through, you should go through all the steps. Subsequently, you can just skip to the last step.

For this procedure to work, you must be logged in under your normal ordinary user name to a workstation with LINUXI386 architecture (Intel or AMD 32-bit architecture) and must have a cvs workspace.

1. If the workstation is not already configured to be part of the pvm, follow the instructions in PVM Administration at RFPK [3] and in this document to make it so.
2. If the command

   ```
   set | egrep '(PVM)|(SPK)'
   ```

   shows that PVM_ROOT, PVM_ARCH and SPK_SHARE are all defined, *skip to the next step*.

   Otherwise, edit `~/.bashrc` to add the following lines:

   ```
   PVM_ROOT=/usr/share/pvm3
   SPK_SHARE=/usr/local/spk/share
   export PVM_ROOT SPK_SHARE
   PVM_ARCH=$($PVM_ROOT/lib/pvmgetarch)
   export PVM_ARCH
   ```

   then exit the terminal window and open a new one, in order to activate the definitions.

3. Make sure that your user name is a member of the *spkshare* group. You can determine this by executing the command

   ```
   grep spkshare /etc/group
   ```

   and seeing whether or not your user name is in the list at the end of the entry that is displayed. If not, execute the following from a terminal window owned by your ordinary user name:

   ```
   su
   gpasswd -a $USER spkshare
   exit
   ```

4. In your cvs workspace, go to the directory `r2/src/misc/parallel` and execute the following commands:

   ```
   make
   make install
   ```

LINUXX86_64 architecture

In the following, several steps are concerned with configuration. The first time through, you should go through all the steps. Subsequently, you can skip the configuration steps.

1. This process starts on your 32-bit workstation, where it is assumed that your cvs workspace resides, then continues on the 64-bit cspkserver.

   Starting in the source code directory, `r2/src/misc/parallel`, make a tarball, then copy it to your home directory on cspkserver:

   ```
   cd ..                           # go one directory above the source directory
   tar cvzf parallel.tgz parallel # make a tarball
   scp parallel.tgz cspkserver:   # copy it to cspkserver
   rm parallel.tgz                # clean up
   ```

2. Go to cspkserver.

   ```
   ssh cspkserver
   ```

3. If the command

   ```
   set | egrep '(PVM)|(SPK)'
   ```

   shows that PVM_ROOT, PVM_ARCH and SPK_SHARE are all defined, *skip to the next step*.

   Otherwise, edit `~/.bashrc` to add the following lines:

   ```
   PVM_ROOT=/usr/share/pvm3
   SPK_SHARE=/usr/local/spk/share
   export PVM_ROOT SPK_SHARE
   PVM_ARCH=$($PVM_ROOT/lib/pvmgetarch)
   export PVM_ARCH
   ```

   then log in again, to activate the new definitions:

   ```
   exit
   ssh cspkserver
   ```

4. Make sure that your user name is a member of the *spkshare* group. You can determine this by executing the command

   ```
   grep spkshare /etc/group
   ```

   and seeing whether or not your user name is in the list at the end of the entry that is displayed. If not, execute the following from a terminal window owned by your ordinary user name: *root* user:

   ```
   su
   gpasswd -a $USER spkshare
   ```

```
exit
```

5. Now you are ready to build and install the prototype.

   ```
   tar xvzf parallel.tgz
   cd parallel
   make clean
   make
   make install
   ```

Running the Parallel SPK Prototype

1. Log into cspkserver as pvm in three separate windows:

   ```
   ssh pvm@cspkserver
   ssh pvm@cspkserver
   ssh pvm@cspkserver
   ```

2. Start a real-time display of the log in one of the windows:

   ```
   tail -f $SPK_SHARE/log/spktest/messages
   ```

3. Start the pvm console in the second window. This will also start the master pvm daemon on cspkserver and add cspkserver to the pvm

   ```
   pvm
   ```

   At the pvm console command prompt, add additional hosts to the pvm:

   ```
   add rose
   add azalea
   add pasta
   add jambutty
   ```

4. Query the console for a list of hosts:

   ```
   conf
   ```

5. In the third window, start a job. The parameter specifies the number of instances (in this case 10 instances).

   ```
   newjob.sh 10
   ```

6. Query the console for a list of the tasks that are running:

   ```
   ps -a
   ```

7. Observer messages as they are added to the log.
8. When the first job completes, try another job. This time delete a node on which several instances of spkind are running

```
delete jambutty
```

and see what happens.

9. Run another job. This time delete the node that spkpop is running on. (If the node is cspkserver, don't delete it. Wait until the job is done and then start another.) Suppose that spkpop is running on rose. Delete that node

   ```
   delete rose
   ```

   and see what happens.

10. When you are tired of running jobs, halt the pvm by entering the following at the console prompt:

    ```
    halt
    ```

Copyright Notice

Notes

1. ../pvm-admin/pvm-admin.html
2. ../pvm-admin/pvm-admin.html
3. ../pvm-admin/pvm-admin.html
4. http://www.opencontent.org/openpub/