Optimizing resource management in supercomputers with SLURM

Dig into this introduction to the Simple Linux Utility for Resource Management

M. Tim Jones
Independent author
Consultant

22 May 2012

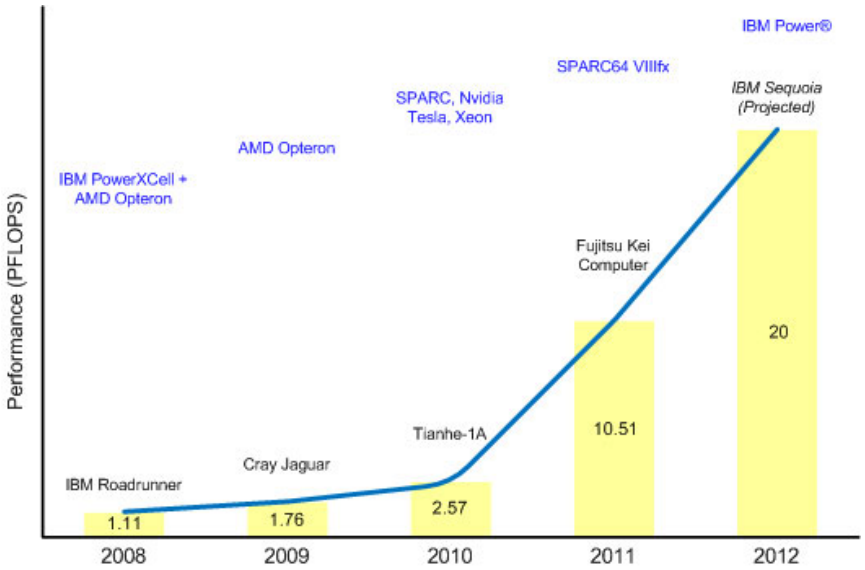
The arms race of supercomputers is fascinating to watch as their evolving architectures squeeze out more and more performance. One interesting fact about supercomputers is that they all run a version of Linux. To yield the greatest amount of power from an architecture, the SLURM open source job scheduler (used by Chinese Tianhe-IA supercomputer, and the upcoming Sequoia supercomputer from IBM®) optimizes resource allocation and monitoring. Learn about SLURM and its approach to parallelizing workloads in clusters.

Connect with Tim

Tim is one of our most popular and prolific authors. Browse all of Tim's articles on developerWorks. Check out Tim's profile and connect with him, other authors, and fellow developers in the developerWorks community.

Supercomputers are a classic example of an arms race. While the growing performance of modern supercomputers expands into new problem domains, these massive systems are providing platforms for solving new problems. Supercomputers are a source of national and corporate pride, as companies and nations work to improve LINPACK scores. Figure 1 illustrates the past five years of the supercomputing arms race, with the IBM Sequoia supercomputer the current projected leader in 2012. As shown, IBM Roadrunner was the first supercomputer to break the sustained petaflops barrier (and IBM Blue Gene®/L held the top spot from 2004 through 2008).

Figure 1. Supercomputer performance: 2008-2012



Early supercomputers were designed to model nuclear weapons. Today, their application is much more diverse, tackling massive computational problems in the fields of climate research, molecular modeling, large-scale physical simulations, and even brute force code breaking.

1964 to today

What is the LINPACK benchmark?

To compare the performance of competing supercomputers, the LINPACK performance benchmark was created. LINPACK measures the rate of execution for floating-point operations. In particular, LINPACK is a set of programs that solve dense systems of linear equations.

The first supercomputer is generally considered the Control Data Corporation (CDC) 6600, released in 1964 (designed by Seymour Cray). The 6600 filled four cabinets with hardware, a Freon cooling system, and a single CPU capable of 3 million floating-point operations per second (FLOPS). Although not lacking in aesthetics, its cabinets were visibly filled with colored wires tying its peripheral unit processors to the single CPU to keep it as busy as possible.

Fast-forward to today, and the current supercomputer leader is the Japanese Kei computer (built by Fujitsu). This system focuses on brute strength compute capacity, using more than 88,000 SPARC64 processors spread across 864 cabinets. The Kei supercomputer has the distinction of breaking the 10-petaflop barrier. Similar to the CDC 6600, the Kei uses water cooling in addition to air cooling.

What is a supercomputer?

A supercomputer is not about any particular architecture but simply a design that is on the bleeding edge of computational performance. Today, this means a system that operates in the performance range of petaflops (or quadrillions of FLOPS) as measured by the LINPACK benchmark.

Regardless of how the supercomputer achieves those FLOPS, a low-level goal of any supercomputer architecture is to optimally keep the compute resources busy when there's work to do. Similar to the CDC 6600 peripheral processors, which existed to keep its single CPU busy, modern supercomputers need the same fundamental capability. Let's look at one such implementation of compute node resource management, called the *Simple Linux® Utility for Resource Management* (SLURM).

SLURM in a nutshell

SLURM is a highly scalable and fault-tolerant cluster manager and job scheduling system for large clusters of compute nodes. SLURM maintains a queue of pending work and manages the overall utilization of resources by this work. It also manages the available compute nodes in an exclusive or nonexclusive fashion (as a function of the resource need). Finally, SLURM distributes jobs to a set of allocated nodes to perform the work in addition to monitoring the parallel jobs to their completion.

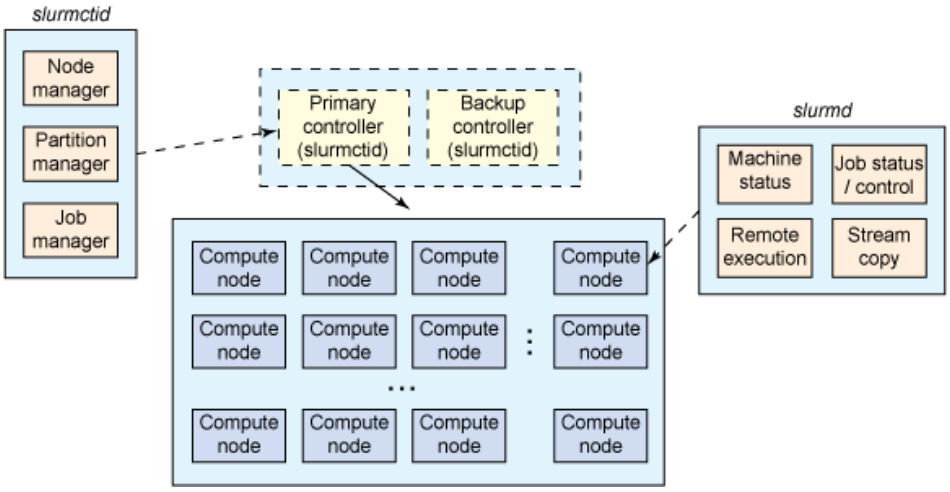
Under the covers, SLURM is a robust cluster manager (focusing on need over feature richness) that is highly portable, scalable to large clusters of nodes, fault tolerant, and most importantly, open source. SLURM began as open source resource manager, developed by several companies (including the Lawrence Livermore National Laboratory) in a collaborative effort. Today, SLURM is a leading resource manager on many of the most powerful supercomputers.

SLURM architecture

SLURM implements a fairly traditional cluster management architecture (see Figure 2). At the top is a redundant pair of cluster controllers (though redundancy is optional). These cluster controllers serve as the managers of the compute cluster and implement a management daemon called `slurmctld`. The `slurmctld` daemon provides monitoring of compute resources, but most importantly, it maps incoming jobs (work) to underlying compute resources.

Each compute node implements a daemon called `slurmd`. The `slurmd` daemon manages the node on which it executes, including monitoring the tasks that are running on the node, accepting work from the controller, and mapping that work to tasks on the cores within the node. The `slurmd` daemon also can stop tasks for executing, if requested from the controller.

Figure 2. High-level view of the SLURM architecture

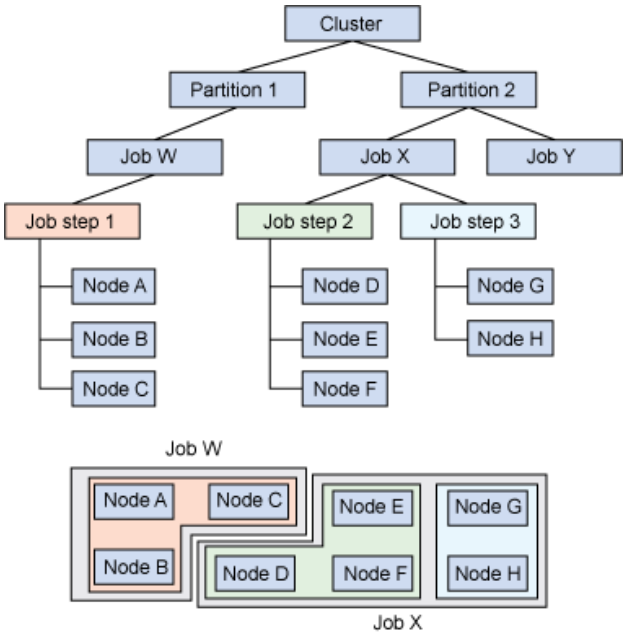


Other daemons exist within the architecture—for example, to implement secure authentication. But a cluster is more than just a random collection of nodes, as some of these nodes can be logically related at points in time for parallel computation.

A set of nodes can be collected into a logical group called a *partition,* which commonly includes a queue of incoming work. Partitions can be configured with constraints about which users can use it or the job size of time limit the partition supports. A further refinement of a partition is the mapping of a set of nodes within a partition to a user for a period of time for work, which is called a *job.* Within a job is one or more job *steps,* which are sets of tasks executing on that subset of nodes.

Figure 3 shows this hierarchy, which further illustrates the SLURM partitioning of resources. Note that this partitioning includes awareness of resource proximity to ensure low-latency communication among cooperating nodes.

Figure 3. Resource partitioning in SLURM



Installing SLURM

How you install SLURM ultimately depends on your particular Linux environment, but the process can be as simple as using a package manager. SLURM has been fully packaged, making it easy to install and configure. For my favorite distro, Ubuntu, I use the Advanced Packaging Tool (APT) to install the SLURM package and all of its dependencies:

```
$ sudo apt-get install slurm-llnl
```

This operation consumes less than 40MB of space and includes not just SLURM but each dependency, basic plug-ins, and other necessary packages.

Configuring SLURM

Before starting SLURM, you must configure it for your particular environment. To create my configuration file, I used the online SLURM configurator, which generates the configuration file for me based on form data. Note that the file needs to be massaged in the end to remove options that were no longer supported. Listing 1 shows my resulting configuration file (stored at /etc/slurm-llnl/slurm.conf).

Listing 1. SLURM configuration file for a single-node cluster

```
# slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ControlMachine=mtj-VirtualBox
#
AuthType=auth/none
CacheGroups=0
```

```
CryptoType=crypto/openssl
MpiDefault=none
ProctrackType=proctrack/pgid
ReturnToService=1
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmctldPort=6817
SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/tmp/slurmd
SlurmUser=slurm
StateSaveLocation=/tmp
SwitchType=switch/none
TaskPlugin=task/none
#
# TIMERS
InactiveLimit=0
KillWait=30
MinJobAge=300
SlurmctldTimeout=120
SlurmdTimeout=300
Waittime=0
#
# SCHEDULING
FastSchedule=1
SchedulerType=sched/backfill
SchedulerPort=7321
SelectType=select/linear
#
# LOGGING AND ACCOUNTING
AccountingStorageType=accounting_storage/none
ClusterName=cluster
JobCompType=jobcomp/none
JobCredentialPrivateKey = /usr/local/etc/slurm.key
JobCredentialPublicCertificate = /usr/local/etc/slurm.cert
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/none
SlurmctldDebug=3
SlurmdDebug=3
#
# COMPUTE NODES
NodeName=mtj-VirtualBox State=UNKNOWN
PartitionName=debug Nodes=mtj-VirtualBox Default=YES MaxTime=INFINITE State=UP
```

Note that in a real cluster, `NodeName` would refer to a range of nodes, such as `snode[0-8191]`, to indicate 8192 unique nodes (named `snode0` through `snode8191`) in the cluster.

The final step is to create a set of job credential keys for my site. I chose to use `openssl` for my credential keys (which are referenced in the configuration file in Listing 1 as `JobCredential*`. I simply use `openssl` to generate these credentials, as in Listing 2.

Listing 2. Creating credentials for SLURM

```
$ sudo openssl genrsa -out /usr/local/etc/slurm.key 1024
Generating RSA private key, 1024 bit long modulus
.................++++++
.........................................++++++
e is 65537 (0x10001)
$ sudo openssl rsa -in /usr/local/etc/slurm.key -pubout -out /usr/local/etc/slurm.cert
writing RSA key
```

With these steps complete, I have everything I need to tell SLURM about my configuration. I can now start SLURM and interact with it.

Starting SLURM

To start SLURM, you simply use the management script defined in /etc/init.d/slurm. This script accepts `start`, `stop`, `restart`, and `startclean` (to ignore all previously saved states). Starting SLURM with this method causes the `slurmctld` daemon to begin (as well as the `slurmd` daemon on your node, in this simple configuration):

```
$ sudo /etc/init.d/slurm-llnl start
```

To validate that SLURM is now running, use the `sinfo` command. The `sinfo` command returns information about the SLURM nodes and partitions (in this case, a single node makes up your cluster), as in Listing 3.

Listing 3. Using the sinfo command to view the cluster

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug*       up   infinite      1   idle mtj-VirtualBox
$
```

More SLURM commands

You can get more information about your SLURM cluster from the variety of commands available in SLURM. In Starting SLURM, you saw the `sinfo` command used to learn about your cluster. You can get more information with the `scontrol` command, which allows you to view detailed information about the various aspects of the cluster (as in Listing 4, the partition and the node).

Listing 4. Getting more detailed information on a cluster with scontrol

```
$ scontrol show partition
PartitionName=debug
   AllocNodes=ALL AllowGroups=ALL Default=YES
   DefaultTime=NONE DisableRootJobs=NO Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=1
   Nodes=mtj-VirtualBox
   Priority=1 RootOnly=NO Shared=NO PreemptMode=OFF
   State=UP TotalCPUs=1 TotalNodes=1

$ scontrol show node mtj-VirtualBox
NodeName=mtj-VirtualBox Arch=i686 CoresPerSocket=1
   CPUAlloc=0 CPUErr=0 CPUTot=1 Features=(null)
   Gres=(null)
   OS=Linux RealMemory=1 Sockets=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1
   BootTime=2012-03-07T14:59:01 SlurmdStartTime=2012-04-17T11:10:43
   Reason=(null)
```

To test your simple SLURM cluster, you can use the `srun` command. The `srun` command allocates a compute resource and launches a task for your job. Note that you can do this separately (through `salloc` and `sbatch`), as well. As shown in Listing 5, you submit a simple shell command as your job to demonstrate `srun`, and then submit a `sleep` command (with an argument) and demonstrate the use of the `squeue` command to show the jobs that exist in the cluster.

Listing 5. Submitting jobs to the cluster and checking queue status

```
$ srun -l hostname
0: mtj-VirtualBox
$ srun -l sleep 5 &
[1] 24127
$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
     15     debug    sleep      mtj   R       0:03      1 mtj-VirtualBox
$
[1]+  Done                    srun -l sleep 5
$
```

Note in Listing 5 that the job you submit to the cluster can be a simple Linux command, a shell script file, or a proper executable.

As a final example, look at how to stop a job. In this case, you start a longer-running job and use `squeue` to identify its ID. Then, you use the job ID with the `scancel` command to terminate that job step (see Listing 6).

Listing 6. Terminating a job step

```
$ srun -l sleep 60 &
[1] 24262
$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
     16     debug    sleep      mtj   R       0:03      1 mtj-VirtualBox
$ scancel 16
srun: Force Terminated job 16
$ srun: Job step aborted: Waiting up to 2 seconds for job step to finish.
0: slurmd[mtj-VirtualBox]: error: *** STEP 16.0 CANCELLED AT 2012-04-17T12:08:08 ***
srun: error: mtj-VirtualBox: task 0: Terminated

[1]+  Exit 15                 srun -l sleep 60
$
```

Finally, you can stop your cluster using the same `slurm-llnl` script, as in Listing 7.

Listing 7. Stopping the SLURM cluster

```
$ sudo /etc/init.d/slurm-llnl stop
 * Stopping slurm central management daemon slurmctld                          [ OK ]
 * Stopping slurm compute node daemon slurmd                                   [ OK ]
slurmd is stopped
$
```

Unlike Apache Hadoop, SLURM has no concept of a distributed file system. As such, it requires a bit more processing to distribute data to the nodes for a given computation. SLURM includes a command called `sbcast`, which is used to transfer a file to all nodes allocated for a SLURM job. It is possible—and potentially more efficient—to use a parallel or distributed file system across the nodes of a SLURM cluster, thus not requiring `sbcast` to distribute data to process.

In this simple demonstration of SLURM, you used a subset of the available commands and an even smaller subset of the available options to these commands (for example, see the options available in the `srun` command). Even with the minimal number of commands available, SLURM implements a capable and efficient cluster manager.

Tailoring SLURM

SLURM isn't a static resource manager but rather a highly dynamic one that can incorporate new behaviors. SLURM implements a plug-in application programming interface (API) that permits run time libraries to be loaded dynamically at run time. This API has been used to develop a variety of new behaviors, including an interconnect fabric, authentication, and scheduling. Plug-in interfaces support a variety of other capabilities, such as job accounting, cryptographic functions, Message Passing Interface (MPI), process tracking, and resource selection. All this allows SLURM to easily support different cluster architectures and implementations. Check out the SLURM Programmer's Guide in Resources for details.

What's ahead for SLURM

In 2011, SLURM was updated with a variety of new features, including partial support for the IBM Blue Gene/Q supercomputer and the Cray XT and XE computers. Support for Linux control groups (cgroups) was also added, which provides greater control over Linux process containers.

In 2012, Blue Gene/Q support will be fully implemented, along with improved resource selection as a function of job need and resource capabilities (for example, node feature AMD). A new tool is planned to report scheduling statistics, and in the near future, a web-based administration tool. Another future plan for SLURM is in the context of cloud bursting, which involves allocating resources in a cloud provider and migrating overflow work from a local cluster to the cloud (also running SLURM daemons). This model can be quite useful and supports the idea of elasticity within certain supercomputer workloads.

Finally, the SLURM developers are considering the use of power and thermal data to more effectively distribute work in a cluster—for example, placing jobs that will consume high power (and thus generate more heat) in areas of the cluster that are more likely to shed heat.

Going further

This short introduction to SLURM illustrates the simplicity of this open source resource manager. Although modern supercomputers are beyond the price range of most people, SLURM provides the basis for a scalable cluster manager that can turn commodity servers into a high-performing cluster. Further, SLURM's architecture makes it easy to tailor the resource manager for any supercomputer (or commodity cluster) architecture. This is likely why it's the leading cluster manager in the supercomputer space.

Resources

Learn

- SLURM: A Highly Scalable Resource Manager has a useful website administered at the Lawrence Livermore National Laboratory. This site provides an overview of SLURM, a great list of publications and presentations, documentation, and a useful FAQ. You can also find professional support by the developers of SLURM at SchedMD.
- A Brief History of Supercomputing from Gordon Bell of Microsoft presents an interesting (though slightly dated) history of supercomputing, from the CDC 6600 to Beowulf clusters.
- The LINPACK Benchmark: Past, Present, and Future (Jack Dongarra, Piotr Luszczek, and Anttoine Petitet) provides a useful introduction to LINPACK history and internals. It also demonstrates LINPACK over a set of common hardware (AMD Athlon, Intel® Pentium®, and so on).
- SLURM Elastic Computing implements the capability called *cloud bursting*, which allows a cluster to grow and shrink as a function of the needs placed in it. In particular, growth of the cluster uses public cloud resources to cost-effectively grow the capacity of the cluster.
- The Top500 project maintains a ranking of the world's fastest supercomputers, along with details on current and historical systems. Top500 releases its updated list twice per year. Not surprisingly, the last 10 positions in the Top500 all ran a form of Linux.
- IBM supercomputers have a long and distinguished history. Check out this reference from Wikipedia for a list and details of some of the supercomputers developed by IBM.
- The SLURM Programmer's Guide provides a readable introduction to build plug-ins and generally get to know the layout of the SLURM source and documentation.
- The SLURM Configuration Tool provides a simple way to generate a slurm.conf file from a simple form of input. After entering the required set of information, the website emits a slurm.conf that is ready to drop into your cluster.
- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth of other resources for Linux developers and administrators.
- The Open Source developerWorks zone provides a wealth of information on open source tools and using open source technologies.
- developerWorks Web development specializes in articles covering various web-based solutions.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools, as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow Tim on Twitter. You can also follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.
- Get involved in the developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

M. Tim Jones

M. Tim Jones is an embedded firmware architect and the author of Artificial Intelligence: A Systems Approach, GNU/Linux Application Programming (now in its second edition), AI Application Programming (in its second edition), and BSD Sockets Programming from a Multilanguage Perspective. His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a platform architect with Intel and author in Longmont, Colo.