Cobalt Manual

Narayan Desai

Rick Bradshaw

Cobalt Manual

Narayan Desai
Rick Bradshaw

Published April 2006

Table of Contents

Chapter 1. Installation

This section describes how to install Cobalt. Once these steps are completed, Cobalt will be completely functional on the system.

Prerequisites

Three prerequisites are required for Cobalt. Each of these, their functions and a download location are described below.

Python	Cobalt is written in python. It requires version 2.3 or greater.
DB2-python	This is a library for connecting to DB2 databases from python. This is only required for Cobalt on BG/L systems. It is available at ftp://ftp.mcs.anl.gov/pub/cobalt.
PyOpenSSL	PyOpenSSL provides python bindings for OpenSSL. It is required in order to support HTTPS on the server side. It is only needed on hosts where components execute.

Software Installation

Install python, db2-python, and pyopenssl on the server side. On SLES9, this can be accomplished by running:

```
# rpm -ihv \
ftp://ftp.mcs.anl.gov/pub/cobalt/rpms/sles9-ppc64/PyOpenSSL-0.6-1.ppc64.rpm
# rpm -ihv \
ftp://ftp.mcs.anl.gov/pub/cobalt/rpms/sles9-ppc64/cobalt-0.95-1.ppc64.rpm
```

On both the client and server sides:

```
# rpm -ihv \
ftp://ftp.mcs.anl.gov/pub/cobalt/rpms/sles9-ppc64/cobalt-clients-0.95-1.ppc64.rpm
```

Configuring the Cobalt Component Infrastructure

Cobalt uses https for data security between components and their clients. Each machine where components run must have their own ssl key. This can be generated by running:

```
# openssl req -x509 -nodes -days 1000 -newkey rsa:1024 \
-out /etc/cobalt.key -keyout /etc/cobalt.kek
```

Components can be located using static records in `/etc/cobalt.conf`, or by using a dynamic service location service. The service location component (slp.py) is bootstrapped similarly to dns; if a direct reference to a component isn't included in `/etc/cobalt.conf`, then it is looked up in the component listed as "service-location".

Copy the sample cobalt.conf file into place, and change the hostname in the service location component line to the one where cobalt components will run. Choose a secret password, and place this in the password field of the communication section. Once all of this is done, the cobalt component infrastructure is completely configured.

Cobalt Component Startup

Cobalt includes four components for resource management. Each of these components provides a specific type of functionality.

Process Manager	The process manager starts, manages, signals, and cleans up parallel processes. On BG/L, its functionality is implemented using the builtin process management system implemented by IBM. The program is `/usr/sbin/bgpm.py`. Bgpm requires several configuration parameters to be set in `/etc/cobalt.conf`. These parameters control environment setup for jobs executed. Incorrect parameters can cause process execution to fail on nodes. This process is started by the cobalt init.d script. Configuration file options are documented in the bgpm(8) man page. On clusters, the process manager uses MPD to start processes. The component is called `/usr/sbin/mpdpm.py` and is started by the sss-pm init script. Mpdpm doesn't currently take any configuration file parameters. On Blue Gene/L, the mpirun command must work for users on the host running bgpm.py. This is usually the service node. In most cases, rsh/ssh must be reconfigured to allow users to ssh from the service node to the service node. (This allows the mpirun frontend to properly contact the mpirun backend)
Service Location Protocol, Queue Manager	The service location component tracks the locations of active systems in the component. It uses a heartbeat mechanism to detect component failure or exit. It can be queried with the slpstat command. The queue manager handles all aspects of action aggregation related to jobs. For example, it uses the process manager interfaces to run user jobs, as well as prologue and epilogue scripts. It also handles job stdio handling on systems without a global shared filesystem. Cqm is the cobalt implementation of the queue manager. It is common to both BG/L and clusters, though it must be configured slightly differently for each. It uses a number of parameters in the `/etc/cobalt.conf` that control the behavior of jobs and which external systems are used. The

queue manager currently has support for file staging (for machines without global shared filesystems), and basic support for allocation management. This daemon is started by the cobalt init.d script.

All configuration file options are documented in the cqm(8) man page.

Scheduler

The scheduler controls resource allocation for job execution. It tells the queue manager when and where to run jobs. Due to differences in scheduling requirements, Blue Gene/L systems and clusters require different schedulers.

Bgsched is the scheduler for Blue Gene/L systems. It internally tracks partition state and performs DB/2 queries to ensure coherent partition usage in case of problems. Bgsched currently only accepts configuration options to control database connection parameters. These options are documented in the bgsched(8) man page. It is started by the cobalt init.d script.

Describe the cluster scheduler here.

Allocation Manager

The allocation manager tracks users, their project memberships, and time allocations. It is used by the scheduler to control resource allocation. A common allocation manager is used on cluster systems and Blue Gene/L systems. It currently has no configuration file options, and isn't started up by the cobalt init.d script yet.

Once each of these components is started, an entry will appear in the service location component. This can be displayed with another call to `/usr/sbin/slpstat.py`.

Each component can also be queried with a component specific tool. For example, the queue manager can be queried with the `cqstat` command. See the clients directory for other commands that can connect to cobalt clients.

Basic Component Testing

Need to rewrite.

Chapter 2. Component Operations

During normal operations, a variety of messages are produced. This allows for most state to be tracked through logs. All messages are logged to syslog facility LOG_LOCAL0, so ensure that these messages are captured.

Job Execution

Job execution is the most common operation in cobalt. It is a procedure that requires several components to work in concert. All jobs go through the same based steps:

Initial Job Queueing	A request is sent to the queue manager describing a new job. Aspects of this request are checked both on the server side, and in `cqsub`, for better user error messages. Whenever a job is created or changes state, appropriate events are emitted. These events can be seen using the `eminfo.py` command. Any client that has subscribed to this sort of event will receive a copy.
Job Scheduling	The scheduler periodically pools the queue manager for new jobs, and can also receive events as an asynchronous notification of queue activity. At these times, it connects to the queue manager and fetches information about current jobs. This process results in a set of idle partitions and idle jobs. If both sets are non-empty, then the scheduler attempts to place idle jobs on idle partitions. This cycle culmunates in the execution of suitable jobs, if they can be scheduled.
Job Execution	Once the queue manager gets a job-run command from the queue manager, it can start the job on those specified resources. At this point, the job state machine is activated. This state machine can contain different steps depending on the underlying architecture and which queue manager features are enabled. For example, enabling allocation management functionality causes jobs to run several extra job steps before completion. These extra steps will not be discussed here; our main focus is generic job execution.
Process Group Execution	The queueing system spawns some number of parallel processes for each job. The execution, management, and cleanup of these processes is handled by the process manager. It, like the queue manager, emits a number of events as process groups execute.
Process Group Cleanup	Parallel process management semantics are not unlike unix process semantics. Processes can be started, signalled, killed, and can exit of their own accord. Similar to unix processes, process groups must be reaped once they have finished execution. At reap time, stdio and return codes are available to the "parent" component.
Job Step Execution	As the job executes, some number of process groups will be executed. These will result in a number of cycles of the previous two steps. Note that process groups can be serial as

	well, so steps like job prologue and epilogue are executed in an identical fashion.
Job Completion	Once all steps have completed, the job is finished. Cleanup consists of logging a usage summary, job deletion from the queue, and event emission. At this point, the job no longer exists.
Scheduler Cleanup	When the job no longer exists in the queue manager, the scheduler flags it as exited and frees its execution location. It then attempts to schedule idle jobs in this location.

Job Log Trace

The following is a set of example logs pertaining to a single job.

```
Jun 29 20:27:14 sn1 BGSched: Found new job 4719
Jun 29 20:27:14 sn1 BGSched: Scheduling job 4719 on partition R000_J108-32
Jun 29 20:27:14 sn1 cqm: Running job 4719 on R000_J108-32
Jun 29 20:27:14 sn1 cqm: running step SetBGKernel for job 4719
Jun 29 20:27:14 sn1 cqm: running step RunBGUserJob for job 4719
Jun 29 20:27:14 sn1 bgpm: ProcessGroup 84 Started on partition R000_J108-32. pid:
Jun 29 20:27:16 sn1 bgpm: Running /bgl/BlueLight/ppcfloor/bglsys/bin/mpirun mpirun
  -np 32 -partition R000_J108-32 -mode co
  -cwd /bgl/home1/adiga/alumina/surface/slab_30/1x1/300K/zerok
  -exe /home/adiga/alumina/surface/slab_30/1x1/300K/zerok/DLPOLY.X
Jun 29 21:05:28 sn1 bgpm: ProcessGroup 84 Finshed. pid 29368
Jun 29 21:05:28 sn1 cqm: user completed for job 4719
Jun 29 21:05:28 sn1 cqm: running step FinishUserPgrp for job 4719
Jun 29 21:05:29 sn1 bgpm: Got wait-process-group from 10.0.0.1
Jun 29 21:05:29 sn1 cqm: running step Finish for job 4719
Jun 29 21:05:29 sn1 cqm: Job 4719/adiga on 32 nodes done. queue:9.18s user:2294.08
Jun 29 21:05:35 sn1 BGSched: Job 4719 gone from qm
Jun 29 21:05:35 sn1 BGSched: Freeing partition R000_J108-32
Jun 29 21:28:37 sn1 BGSched: Found new job 4720
```

In the event that this job ran out of time or was cqdelled, additional log messages would appear to that effect.

Data Persistence

Each of these components must store persistent data, for obvious reasons. Each of the components present in cobalt store data using a common mechanism. These functions are implemented in common code. Each component has some data that needs to be persistent. Periodically, each component marshalls this data down to a text stream (using Python's cPickle module), and saves this data in a file in the directory `/var/spool/sss`. The filenames in this directory correspond to the component implementation name. This is the name that appears in syslog log messages (ie cqm, bgpm, BGSched).

This data can be manipulated from a python interpreter using the `cddbg.py`. This should not be attempted unless you really know what you are doing.

Chapter 3. Component Specific Notes

This chapter describes component specific issues.

bgsched

bgsched keeps an internal representation of the partitions it has to schedule. These partitions can be queried with the `partadm.py` command. These partitions must be manually defined. By default, no partitions are definied. Partition definitions contain information including partition name, size, dependencies (contained partitions), and a list of valid queues. Other information is also tracked about partitions. An overall state is maintained (idle or busy). Note that partition dependencies cannot contain nonexistant partitions; that is, if a partition is deleted, it must be removed from any dependency lists it is in.

The standard operations of bgsched are fairly simple. The scheduler queries the queue manager, compares the list of jobs it received with the list of jobs that it already knew about, and appropriately deals with any discrepancies. When partitions are free and idle jobs are in the queue, it attempts to schedule. When jobs disappear, the partition previously occupied by the job is freed.

Potential Problems

Several sorts of problems can occur to cause problems with the scheduler. Some of its data is maintained in other components or in the DB2 database for BG/L. Failures in this other software can render the scheduler unable to function properly. In case of either failure, odds are good that scheduling will be the least of your worries. There are error messages bgsched will report upon connection failures.

Chapter 4. Troubleshooting