

# **Cobalt Manual**

**Narayan Desai**

**Rick Bradshaw**

## **Cobalt Manual**

by Narayan Desai

by Rick Bradshaw

Published June 2005

Copyright © 2005 Argonne National Laboratory

This manual is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This is distributed in the hope that it will be useful, but *without any warranty*; without even the implied warranty of *merchantability* or *fitness for a particular purpose*. See the GNU General Public License for more details.

### Revision History

Revision 0.93 \$Date: 05/07/01 15:52:29-05:00 \$

\$Id: manual.xml 1.4 05/07/01 15:52:29-05:00 desai@topaz.mcs.anl.gov \$

# Table of Contents

<b>1. Installation.....</b>	<b>1</b>
1.1. Prerequisites .....	1
1.2. Configuring SSSlib .....	1
1.3. Cobalt Component Installation .....	2
1.4. Basic Component Testing .....	3
<b>2. Component Operations .....</b>	<b>5</b>
2.1. Job Execution .....	5
2.2. Job Log Trace.....	6
2.3. Data Persistence .....	6
<b>3. Troubleshooting .....</b>	<b>8</b>

# Chapter 1. Installation

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

## 1.1. Prerequisites

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

### ElementTree

ElementTree is an XML parsing library for Python. It is available from <http://www.effbot.org>. Debian packages are available from debian. Rpm packages can be built using the distutils configuration shipped with the code.

### SSSlib

SSSlib provides a component architecture infrastructure used by all Cobalt components. It is available from <ftp://ftp.mcs.anl.gov/pub/sss>. This source will product several RPM packages: ssslib, ssslib-python and sss-infrastructure are particularly important.

### Xerces

Xerces is a validating XML parser. It is optional, however, it is needed for XML validation. No validation can occur without it. It is available from <http://xml.apache.org/xerces-c/>. Debian packages are available directly.

### 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>.

## 1.2. Configuring SSSlib

SSSlib provides a set of component infrastructure services. Wire protocol abstraction is included, as is service location and asynchronous messaging. The service location protocol is used to determine component location as well as connection and protocol information. When connecting to a component, each client performs a lookup of the component name, and attempts to connect to this location using the specified protocol. Each client knows how to locate the service directory a priori, in order to find other components.

Configuration of SSSlib consists of two parts. First, communication protocol preferences must be established. Also, a correct service directory location must be specified. This value is used to connect to

the service directory, and thus locate other components. Communication protocol preferences are specified in `/etc/ssslib.conf`. The password supplied in the `[challenge]` section should be changed from its default value. Other settings in this file should remain the same. Static component locations are set in `/etc/sss.conf`. The "service-directory" entry in this file should be modified such that the hostname is where the service directory is run.

Once these steps are complete, the sss infrastructure can be started by calling the sss init script. Two components, `sd.py` and `emng.py`, will be started. Once these have been started, the event manager and service directory components will be returned from `/usr/sbin/sdstat.py`.

```
[0:86] /usr/sbin/sdstat.py
```

Service	Protocol	Location	Version	Tier
event-manager	challenge	topaz:32788	1.0	1
service-directory	challenge	localhost:7000	1.0	1

Once these steps are complete, and those two components are returned from the service directory, the SSS infrastructure is completely functional. This means that other components can be started.

## 1.3. Cobalt Component Installation

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.

### Queue Manager

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 directory. This can be displayed with another call to `/usr/sbin/sdstat.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.

## 1.4. Basic Component Testing

Each of the included components can be independently tested, as each provides disparate functionality. Clients are included that can invoke individual component functionality. `sssjob.py` can submit processes to directly to the process manager for execution. Once the process management system have been verified, the queueing system should be tested next. `cqsub` can submit jobs to the queueing system.

If the scheduler isn't running, these jobs won't start. Jobs can be manually started using `qrun.py` to test the queue manager. Finally, scheduler state can be examined by using the various scheduler client commands. It won't be able to work without all other components active, so scheduling policy can't be tested independently of jobs.

# 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.

## 2.1. 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 culminates 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.

## 2.2. 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: 29368
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:2294.08s
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 cqeddelled, additional log messages would appear to that effect.

## 2.3. 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 `cddb.py`. This should not be attempted unless you really know what you are doing.

## **Chapter 3. Troubleshooting**