SSS-OSCAR Overview

SSS-OSCAR Team

May 24, 2006

1 Introduction

The Scalable Systems Software (SSS) project is a coordinated effort to produce scalable software that can be used to both run and manage scientific computing centers. The intent is to deliver a complete package of software that can meet the needs of a large computing facility. Currently most centers use a mix of vendor provided software and locally coded solutions that are not compatible across multiple sites. This is less than ideal, as it hinders collaboration in a time when increased coordination between scientists is the norm. Also, as larger and larger machines are produced, increased emphasis on effective software that can run and make use of these machines is needed, as current software that is in use cannot cannot work effectively on machines of this size.

To this end, the work of the SSS team has been combined with OSCAR and released to the public using OSCAR as the installation mechanism, and named SSS-OSCAR. Most of the components developed by the SSS project were made into OSCAR packages, so that they could be installed by the OSCAR system. While the SSS software has the goal of being run across different architectures, the SSS-OSCAR release is specifically for x86 hardware at this time. Also, since this release is limited in scope, many assumptions were made concerning the configuration of the SSS packages as installed by OSCAR. Thus, the SSS-OSCAR release is geared toward running on one Beowulf style compute cluster, having one headnode. Specific reference to the software included in this release can be found in the SSS Packages section below.

2 Background

OSCAR is open source and freely available for download via Sourceforge. The current version of OSCAR at the time of this writing is version 4.0, which can work with RedHat versions 9.0 and Fedora Core 2 on x86 hardware. SSS-OSCAR is still using OSCAR 3.0, so it is limited to Red Hat 9.0. OSCAR consists of several different logical entities. One way to think about OSCAR is to separate the packages from the installation mechanism itself. A package is a piece of software that runs on a cluster. Different pieces of the same package can be installed on the head node or the compute nodes or both. Fine grained control of where parts of a package go and running scripts for configuration of a package is the job of the installation mechanism. The packages are the individual bits of software that run on a cluster, such as the popular communication libraries like PVM and MPI. OSCAR has the goal of installing a cluster with minimal input from an administrator, and following commonly understood “best practices” for installing clusters.

The OSCAR working group was formed in 2000 under the Open Cluster Group (OCG) organization. The OCG was founded to address the need for a public forum to develop open source clustering solutions. The OSCAR working group produces a solution for High Performance Computing (HPC) clusters, and OSCAR should technically be labeled “HPC OSCAR”. The OCG forms a convenient umbrella organization for other types of clustering software. Currently there are three working groups [4]:

1. “HPC” OSCAR – previously mentioned, and commonly referred to as just OSCAR
2. Thin-OSCAR – disk-less approach to clustering
3. HA-OSCAR – group focusing on high availability

Membership in these groups is open to anyone that wants to contribute to the project. The more involved a member becomes, the more responsibility that organization/individual can have in the project’s direction. A good way to get

started in OSCAR is with testing the trial releases. Testing by expert users/developers is always needed and seems to be in short supply in most open source projects. [5]

The SSS center was formed in 2001 to meet the software needs of large computing centers. The software being developed has the mission of making terascale computing systems both manageable and usable, and is funded by the Department of Energy. To do this, working groups were formed in various areas in order to pool interest and expertise among the participating developers; the working groups include: process management, resource management, integration & verification, and build & configuration. These groups have focused, for the most part1, on software that runs at the application level and not at the kernel/operating system level [3]. To this end, the reference implementation that this document describes, has been developed first to run on RedHat Linux using x86 hardware. That should ensure that the software reaches the widest possible market initially with the expectation being that many will download the software for evaluation before deploying on production systems. The decision was also made to use OSCAR as the initial delivery vehicle, as this is a popular open source clustering toolkit.

The first step of the SSS project was to define a basic system architecture and agree upon standard interfaces for the system. It was decided that XML would be used for messages between components, and one of the first pieces of software that was developed was a communication library. Development commenced rapidly, and software is available today for testing/evaluation purposes in the form of an OSCAR release.

To expand on some of the thinking for using OSCAR, one must consider the portability question. The notion of software portability can extend down to hardware instruction sets up to runtime/operating system environments. The portability of a clustering toolkit such as OSCAR is bound by the runtime environments it attempts to configure and manage. A goal of OSCAR is to be Linux distribution agnostic. The distributions spend a great deal of time doing integration and testing for a large set of applications and OSCAR tries to leverage this work whenever possible. The current implementation of OSCAR does make some assumptions about the underlying distribution but the objective is to have particular applications, in this case SSS software, port to OSCAR and then leverage the toolkits portability. The portability of the OSCAR framework is bound by the set of supported Linux distributions, and the set of SSS OSCAR packages is bounded by the portability of the framework itself.

3 SSS Packages

The overall design of the SSS system can be summarized quickly in a diagram, commonly known as the “meatball” diagram (Figure 1) to the SSS developers. Each piece of the SSS system has been specified in general, with developers creating components that meet the general specification. This was done to allow different implementations of the same component (with possibly differing feature sets) to exist and be used so long as the general requirements were met.

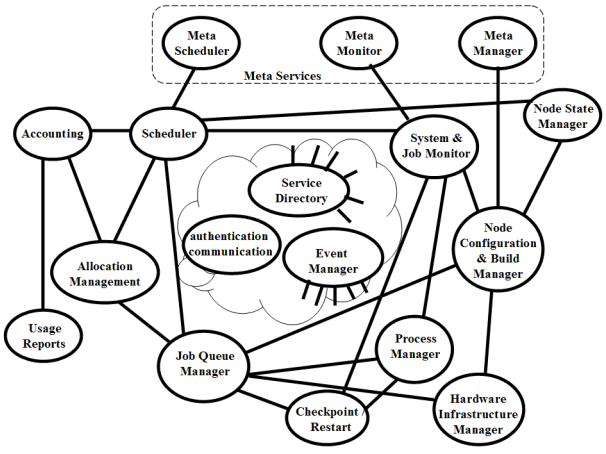

Figure 1: Diagram of the Scalable System Software architecture.

The current software that is specific to SSS-OSCAR is listed below. In some cases a component correlates directly to an OSCAR package, but sometimes one package can contain multiple components. Some of this software is not configured to meet its full potential, but for early software releases the decision was made to publish software that uses a bland configuration in order to speed integration and deployment. As SSS-OSCAR evolves, more of the component's capabilities will be harnessed. All of the SSS components fit together into one integrated system with a well defined communication protocol. The following subsections outline the currently available SSS packages with details regarding their default configuration.

3.1 Communication Infrastructure

SSSlib

The SSSlib is a communication library is used by the SSS components for inter-component communication. The library provides a range of default protocols for communication, e.g., Challenge/Response, and is extensible to allow for new additions. Library bindings exist for several languages including Perl, Python, C and C++. The Service Directory and Event Manager are also key aspects of this communication infrastructure.

Service Directory (SD)

The SD is a centralized routing service that runs as a daemon on the head node. It serves as a central phone book for the system. All SSS components register themselves with the SD so that others can query the SD for component connection and location information. This allows for multiple implementations of the components to be running simultaneously for different communication protocols and capabilities.

Event Manager (EM)

The EM is a process that runs on the head node of the cluster, and collects events or signals that other SSS components generate. The use of this centralized asynchronous event system is used throughout the SSS communication infrastructure. This assists inter-component communication and helps organize the management of the cluster. Events are passed to the EM via the SSSlib. Also, error messages and other types of events from the SSS system can be logged here.

3.2 Scheduler

Maui

Maui is popular scheduling system that is widely used in the HPC community. The SSS version has the additional benefit of being able to work with SSS components and understand the SSS communication infrastructure. Maui interacts with Warehouse (system monitor) and Bamboo (queue manager) to schedule jobs across the cluster. Basically, Maui polls both Bamboo and Warehouse to get a list of jobs to run and some information about the cluster. Then it schedules the next job in the list at an appropriate time, if authorization to run the job is obtained from Gold (allocation manager).

3.3 System Monitor

Warehouse

This is a monitoring system that can provide information about both individual jobs and the entire system. Currently, Warehouse interacts with Maui (scheduler), providing information about the system upon request. Individual job information is not currently used at the time of this writing. Warehouse has a daemon that runs on the head node that serves as an information repository and one information collector daemon per compute node. Currently, the information repository polls the collector daemons on the nodes for job and system information. Maui is the primary consumer of Warehouse's information repository, polling occasionally for the monitoring data to use when scheduling jobs.

3.4 Queue Manager

Bamboo

This component is the queue manager and this is the place where a user submits a job in the SSS system. Bamboo has the responsibility of keeping track of jobs until they are scheduled to run by Maui. Currently, Bamboo responds to requests from Maui to send it job information.

3.5 Process Manager

SSS Process Manager (PM)

The Process Manager is that is provided is a frontend to MPD that exists to handle the SSS communication protocols that have been developed. It performs this service on behalf of MPD, and sends to MPD other types of messages to control its actions. MPD does the work, but the Process Manager interacts with the rest of the SSS system.

Multi Purpose Daemon (MPD)

MPD is a process manager that Argonne National Laboratory (ANL) provides with its MPI implementation, MPICH. MPD serves as the backend to the process manager component of the SSS system, and is a ring of daemons that run across the cluster. MPD is responsible for starting and managing processes across the cluster. MPD also has the ability to redirect the I/O of the process and pass signals back and forth between processes that it manages.

3.6 Checkpoint/Restart

Berkeley Labs Checkpoint/Restart (BLCR)

BLCR is a checkpoint/restart system that works on Linux kernel based systems. It can be used as a standalone library to checkpoint a process on an individual system and in conjunction with LAM/MPI to checkpoint an MPI job across a cluster. BLCR is compiled against a specific kernel, therefore some adjustments must be made if the kernel version is changed [1].

LAM/MPI

LAM/MPI is an implementation of the Message Passing Interface (MPI) standard. In SSS-OSCAR it is compiled to work with BLCR so that MPI jobs can be check-pointed across the cluster [6].

3.7 Resource Allocation Management

Gold

An accounting and allocation manager that basically keeps track of how many system resources a user is allotted, and when/where they can use these privileges. Gold keeps track of information in a PostgreSQL database, so it has a large capacity for information storage. Gold gets queried by Maui (scheduler) to see if a process has permission to run.

4 Acknowledgments

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

4.1 Contributors

The SSS-OSCAR release could not have happened without the following folks:

- Ames National Laboratory: Brett Bode and David Jackson.
- Argonne National Laboratory: Rick Bradshaw, Rusty Lusk, and Narayan Desai.
- Department of Energy: Fred Johnson.
- Indiana University: Jeff Squyres.
- Lawrence Berkeley National Laboratory: Paul Hargrove.
- National Center for Supercomputing Applications: Craig Steffen, Michael Showerman, and Chad Lowe.
- Oak Ridge National Laboratory: Al Geist, Stephen Scott, Thomas Naughton, and John Mugler.
- Pacific Northwest National Laboratory: Scott Jackson and Kevin Walker.
- Sandia National Laboratory: Eric DeBenedictus, William McLendon, Ron Oldfield, Jim Laros, and Neil Pundit.

References

[1] Berkeley Lab Checkpoint/Restart. http://ftg.lbl.gov/checkpoint.

[2] Resource Management and Accounting. http://sss.scl.ameslab.gov/home.shtml.

[3] Al Geist et al. Scalable Systems Software Enabling Technology Center, March 7, 2001. http://www.scidac.org/ScalableSystems/.

[4] John Mugler, Thomas Naughton, Stephen L. Scott, Brian Barrett, Andrew Lumsdaine, Jeffrey M. Squyres, Benot des Ligneris, Francis Giraldeau, and Chokchai Leangsuksun. OSCAR Clusters. In *Proceedings of the* 5^{th} *Annual Ottawa Linux Symposium (OLS'03)*, Ottawa, Canada, July 23-26, 2003.

[5] Thomas Naughton and Stephen L. Scott. Tutorial: Clustering with OSCAR. In *Proceedings of the* 4^{th} *Annual Ottawa Linux Symposium (OLS'02)*, Ottawa, Canada, June 26-29, 2002.

[6] Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, Andrew Lumsdaine, Jason Duell, Paul Hargrove, and Eric Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. In *Proceedings, LACSI Symposium*, Sante Fe, New Mexico, USA, October 2003.