Design and Implementation of a Local Scheduling System with Advance Reservation for Co-allocation on the Grid

Hidemoto Nakada, Atsuko Takefusa, Katsuhiko Ookubo, Makoto Kishimoto
Tomohiro Kudoh, Yoshio Tanaka, Satoshi Sekiguchi

National Institute of Advanced Industrial Science and Technology (AIST)
Grid Technology Research Center
1-18-13 Sotokanda, Chiyoda-ku, Tokyo, 1010021, Japan,
{hide-nakada, atsuko.takefusa, kisimoto.m, ookubo-k, t.kudoh, yoshio.tanaka, s.sekiguchi}@aist.go.jp

Abstract

While advance reservation is an essential capability for co-allocating several resources on Grid environments, it is not obvious how it can co-exist with priority-based First Come First Served scheduling, that is widely used as local scheduling policy today. To investigate this problem, we 1) developed a scheduling API in Java for TORQUE, a variant of OpenPBS, that enables users to implement their own schedulers and replace the original scheduling module with them, 2) implemented a prototype scheduler module that has advance reservation capability with the API. We also provide an external interface for the reservation capability based on WSRF to enable co-allocation of resources over the Grid. Using this interface with the job submission module from Globus toolkit 4, users can make reservation for resources and submit jobs over the Grid.

Keywords: Grid, Advance Reservation, Scheduling, Batch Queuing System, Web Services Resource Framework

1 Introduction

One of the goals of Grid computing is to perform huge computation using numerous resources distributed on the network, simultaneously [13]. While there have been lots of experimental runs with resources from several sites[11], the resources are allocated by the human administrator via phones, e-mails, or faxes. They are just 'experiments', not the daily operations. To bring the huge computation with multi-site managed resources into reality, we really need to have fully-automated co-allocation mechanism.

To co-allocate computational resources, advance reservation capability is required for each local resource scheduler. A global scheduler (often mentioned as 'Super Scheduler') will find commonly available timeslot for all the required resources and make advance reservation on the timeslot, making sure that all the resources will be available for the time[14].

Advance reservation capability is supported by several local schedulers. To have advance reservation itself is relatively easy, but the problem is that each local site has its local users who expect queuing and priority based scheduling for their job. We do not know how to make the reservation based scheduling and the queuing based one consistent.

To study the problem, we need a scheduler that is capable of advance reservation and can be modified easily, as a study testbed. The problem here is that there is no such scheduler that is freely available, easily modifiable and redistributable.

Although some commercial scheduling systems, such as LSF[4] or PBS Professional[7] provide advance reservation capability, they are not for free and cannot be redistributed and modified. Maui scheduler[5] is a plug-in scheduler for OpenPBS[6] or TORQUE[8] that provides various features including advance reservation. The source code for the scheduler is freely available but the modified source code cannot be redistributed. Further more, the internal architecture is not well-defined and it is difficult for the third-party to modify the scheduling policy of Maui.

We implemented a scheduling module that enables advance reservation working with TORQUE[8], a descendant of OpenPBS, We designed and implemented a Java API that abstracts the TORQUE server module, and implemented the scheduling module using the API.

We also designed and implemented an interface to provide the reservation service via network, basing on WSRF (Web Services Resource Framework)[9]. The interface provides authentication and authorization based on Globus

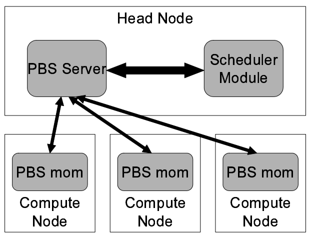

Figure 1. Overview of TORQUE.

Toolkit ver. 4[2, 12] and enables resource reservation and job execution on the Grid in secure manner working with GRAM, a job submission framework provided by Globus Toolkit.

The structure of the rest of paper is as follows. In 2, we introduce TORQUE local scheduler. In 3, we describe the design and implementation of the scheduler. In 4, we discuss on the WSRF based reservation service and GT4 GRAM interaction. In 5, we show results of preliminary evaluation. In 6, we conclude the paper showing future work.

2 Overview of TORQUE

2.1 Architecture of TORQUE

TORQUE is a variant of OpenPBS, an open source batch scheduling system. While the OpenPBS maintenance stopped for years, TORQUE is still well maintained by a company called CLUSTER RESOURCES INC. Under the license agreement, source modification and redistribution of the code are allowed.

TORQUE has following 3 modules(figure 1).

- **PBS Server:** The central server of the whole system. One PBS pool has just one PBS Server. This module takes job queuing requests from users and manages the job queue. It manages the status of all the compute nodes communicating with the PBS mom on each node.
- **Scheduler Module:** This module is responsible for resource allocation for each job. This makes a pair with the PBS Server and often runs on the same machine with it. Just like PBS Server, the scheduler module is just one for one PBS pool.

  When an event occurs, such as job submission or job termination, the PBS Server sends trigger to the scheduling Module. The scheduling module queries status of the job queue and the nodes, decides what job run on which node, and orders PBS Server to perform the decided operation.
- **PBS mom:** This module manages each compute node. It monitors status of the node and reports it to the PBS Server. It is also capable of job invocation and process management of the job.

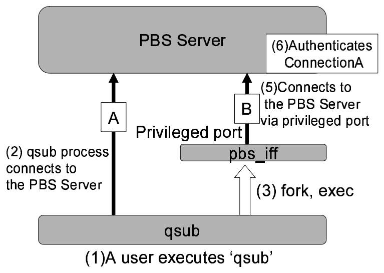
PBS Server
(6)Authenticates ConnectionA
A
B
(5)Connects to the PBS Server via privileged port
Privileged port
(2) qsub process connects to the PBS Server
pbs_iff
(3) fork, exec
qsub
(1)A user executes 'qsub'

Figure 2. Authentication in TORQUE.

2.2 User authentication in TORQUE

The PBS Server operates responding requests from users. When it receives requests, proper authentication has to be performed. PBS server authentication process is as follows.

1. A user executes queuing command, such as 'qsub'.
2. The command program connects to the PBS Server with a socket connection, shown connection A in figure 2. As of this time, the connection is NOT authenticated, and the request through the connection will be refused.
3. The command program forks and execs a program called *'pbs_iff'*, that has setuid flag and owned by root. It means that the program will run with root privilege. Pbs_iff is given the file descriptor number for the connection A.
4. Pbs_iff get the port number of connection A, from the given file descriptor number. Then, it get real user id to know who did invoke the original command. Note that the effective user id will be root here, but the real user id will be kept as the invoking user.
5. Pbs_iff connects to the PBS Server from privileged port (connection B in figure 2) and sends the port number and the user name.
6. PBS Server now can trust the Pbs_iff, because it comes from privileged port, and authenticate the user who is on the other peer on the connection A.

2.3 TORQUE customization

TORQUE scheduler module can be easily customized to a certain extent with a simple customization language called BaSL(Batch Scheduling Language). Although BaSL provides basic functionalities to manage job allocation based

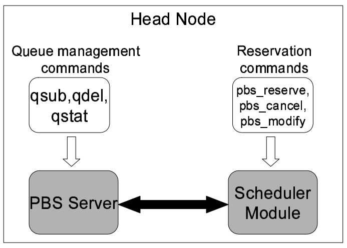

Figure 3. Overview of the proposed system.

on queue and nodes status, the simplicity of BaSL restricts the ability. The scheduler module also can be customized by writing scheduling routine in C. While you can do anything you want with this way, it is really hard to do since there is no well-defined API and requires detailed knowledge of the implementation of the scheduling module.

Another possibility to customize the scheduling is to replace the whole scheduler module. The scheduler module and the PBS server communicate each other with a relatively simple text-based protocol. The Maui scheduler is replacing the original scheduler, talking the protocol

We employed the last approach and replace the whole scheduler with our own Java written scheduler. Also, we implemented an API that provides functions to get queue and node status from the PBS server and perform operations on jobs, to make it easy to implement and try possible scheduling policies.

3 Design and Implementation of Scheduler Module

3.1 The proposed system

We define an API to implement scheduling policies in Java, and implement a scheduling module that is capable of advance reservation, with the API. The Scheduler module maintains reservation table to manage reservation entries. Each reservation entry keeps a unique reservation ID, reservation start time, end time, number of nodes, and reserved nodes.

While job management commands such as qsub communicate with PBS Server, reservation related commands have to communicate with the scheduling module. For the communication, we implemented an interface using Java RMI on the scheduling module. Figure 3 shows the overview of the system.

The scheduling module allocates and returns a unique ID for each reservation request. User specifies the timeslot when a job should be executed by specifying the reservation ID as an optional argument to the submit command. The specified reservation ID is stored as an extra attribute of the job and passed

```
public interface PBSInterface{
   void   setSocket(Socket socket);
   Socket getSocket();
   void   authenticateUser(String userName,
                           int    localPort);
   void   disconnect();
   ServerStatus           statusServer();
   BatchReplyStatusNode   statusNode();
   BatchReplyStatusQueue  statusQueue();
   BatchReplyStatusSelect selectStatus(
                            String queueName);
   void runJob   (String jobId,
                  String destination);
   void runJob   (String jobId,
                  Collection<NodeInfo> nodes);
   void deleteJob(String jobId);
   void holdJob  (String jobId,
                  HoldJobType holdType);
   void rerunJob (String jobId);
   void modifyJob(String jobId,
                  String attr,
                  String value);
}
```

Figure 4. PBSInterface.

to the scheduler module along with the job, like follows.

```
qsub -W x=rsvid:XXXXXXX
```

The scheduler module looks up the reservation time table with the ID and when it finds that the reservation time has come it allocates reserved nodes to the job and execute the job.

The reservation timetable has to be persistent, since it has to survive crashes and reboots of the PBS head node. We use an object database written in Java, called db4objects[1]. Thanks to the simple interface provided by db4objects, implementation was really easy, compared with writing it with JDBC and RDB.

3.2 Overview of the Scheduling API

To make it easy to implements scheduling policies with Java, we provide an API. In figure 4, we show the primal interface of the API, PBSInterface. This interface is an abstraction of the PBS server and provides methods to get information from PBS Servers, such as node and queue status, and methods to operate jobs, such as job execution, removal and stop. Figure 5 shows a simple FIFO scheduler with the API, demonstrating the easiness of implementation of scheduling policy with the API.

3.3 Command Line Interface for reservation

Table 1 shows command to handle reservation. The commands are implemented as shell scripts that invoke corresponding Java programs. The Java programs communicate with the scheduling module with Java RMI(Remote Method Invocation). The scheduling module has to authenticate the user who invoke the commands. We enabled this by implementing customized ServerSocket and Socket classes and modifying the RMI SocketFactory. For authentication mechanism, we employed the technique described in 2.2.

```
public class SimpleFifoScheduler {
  public static void main(String[] args) {
    // start scheduling server
    PBSServerConfig servConf =
      new PBSServerConfig();
    ScheduleStarter starter =
      new ScheduleStarter(servConf);
    PBSInterface pbs = new TorqueImpl();

    // get scheduling order, and run
    ScheduleOrder order;
    PBSSchedulerCommandType cmd =
      PBSSchedulerCommandType.NULL;
    do {
      order = starter.waitOrder();
      Socket socket = order.getPBSServerSocket();
      pbs.setSocket(socket);
      cmd = order.getSchedulerCommand();

      if (cmd.mustRunSchedule()) {
        try {
          schedule(pbs);
        } catch (PBSException e) {}
      }
      socket.close();
    } while (cmd!=PBSSchedulerCommandType.QUIT);
  }
  private static void schedule(PBSInterface pbs)
        throws PBSException {
    ServerStatus server = pbs.statusServer();
    if (!server.isReadyToUse() ||
        server.getQueuedJobs() == 0)
      return; // no jobs to schedule

    Collection<NodeStatus> nodes =
      pbs.statusNode().getAllStatus();

    for (QueueStatus queue : pbs.statusQueue()) {
      if (!queue.isReadyToRun() ||
          queue.getQueuedJobs() == 0)
        continue;    // no jobs to run

      for (JobStatus job :
           pbs.selectStatus(queue.getName())) {
        if (!job.isReadyToRun())
          continue;    // cannot run now

        for (NodeStatus node : nodes) {
          if (!job.isRunnableOn(node))
            continue;    // node is down

          String jobId = job.getJobId();
          String destination = node.getName();
          pbs.modifyJob(jobId, "comment",
                "Job started on " + new Date());
          pbs.runJob(jobId, destination);
          return;
        }
      }
    }
  }
}
```

Figure 5. A Simple FIFO Scheduler implemented in the Proposed API.

4 WSRF based interface for advance reservation

To enable co-allocation over several sites, we have to have an inter-site advance reservation interface. We designed and implemented a interface based on WSRF (Web Service Resource Framework) with Globus Toolkit 4 (GT4), that is a de facto standard in Grid.

WSRF is a standard being specified in OASIS, one of the international standard bodies in the Web Services field. While Web Services does not have status in general, WSRF introduce status as 'resources' of services. GT4 provides not only WSRF implementation but also several components implemented on WSRF, including GRAM service for invoking jobs, and MDS service for gathering and publishing information.

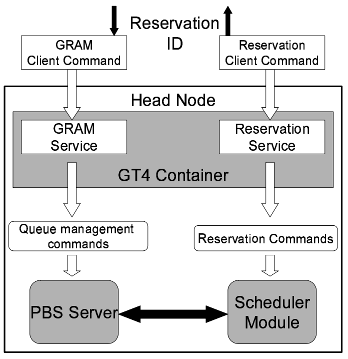

Figure 6. Integration with GT4.

We implemented reservation service based on WSRF with GT4. Using the service and GRAM mentioned above, we enabled authenticated and authorized reservation and execution of jobs on the Grid.

4.1 The design of the reservation service based on WSRF

We implemented two services, one is PBSReservation that stands for one reservation, and PBSReservationFactoryService, that creates the PBSReservation, based on the factory service design pattern, that is commonly used in WSRF. Table 2 shows operations for these services.

The PBSReservationFactoryService is a factory service that has just one operation; 'createPBSReservation' to create the PBSReservation service and return the reference to the service (EPR: End Point Reference). Note that although the operation takes parameters to make reservation and stores it to the PBSReservation service, it just creates the service and does not actually make reservation.

The PBSReservation service has 4 operations, that are corresponds to the reservation commands shown in table 1, and actually, invokes the commands on the server. These operations do not have output. All the outcomes from the invocation of commands will be reflected to the resource properties shown below.

Table 3 shows resource properties for the PBSResrvation. When operations are invoked on a PBSReservation, corresponding reservation commands will be performed on the server side. The commands will update the resource properties when they finish.

4.2 Authentication and authorization for the reservation service

The reservation service has to identify the user who made request and map him/her into a local user. We implemented this so that it is compliant with the GT4 GRAM service; i.e. we use PKI base X.509 certificate to authenticate the

Table 1. Reservation Commands.

Command Name	Function	Input		Output
pbs_reserve	make reservation	[-R SCHEDULER_HOST] -e END_TIME	-s START_TIME -n NUM_OF_NODES	RESERVATION_ID
pbs_rsvcancel	cancel reservation	[-R SCHEDULER_HOST]	-r RESERVATION_ID	
pbs_rsvmodify	modify reservation	[-R SCHEDULER_HOST] [-s START_TIME] [-n NUM_OF_NODES]	-r RESERVATION_ID [-e END_TIME]	
pbs_rsvstatus	show reservation status	[-R SCHEDULER_HOST]	[-r RESERVATION_ID]	RESERVATION_STATUS

Table 2. Reservation operations.

Operation Name	Function	Input	Output
PBSReservationFactoryService			
createPBSReservation	create PBSReservation service	start/end time, No. of Nodes	EPR of the created service
PBSReservation			
reserve	make reservation	N/A	N/A
getStatus	renew the status	N/A	N/A
cancel	cancel reservation	N/A	N/A
modify	modify reservation	start/end time, No. of Nodes	N/A

Table 3. Resource property of PBSReservation.

Prop. Name	Meaning
StartTime	Reservation Start Time
EndTime	Reservation End Time
NodeNum	No. of Reserved Nodes
Caller	Distinguished Name on the Cert.
LocalUsername	Local user name on the site
ReserveId	Reservation ID
ResultStatus	Command Result
ResultStdout	Command stdout
ResultStderr	Command stdin
IsBusy	Command exec. status

incoming user and map the distinguished name on cert into local user using a mapping file, called grid-mapfile.

4.3 Working with GRAM

GRAM is a job submission interface provided by Globus toolkit. GRAM uses modules called Job Managers that is written in Perl, to submit a job into backend batch queuing system, such as TORQUE.

To submit a job at the reservation time, the 'qsub' command has to receive the reservation ID from the upstream module. To enable this, we used GRAM RSL extension mechanism that provides a generic way to pass extra information to the Job Manager. [1].

We leveraged this extension mechanism and slightly modified the Job Manager for TORQUE, so that it takes RSL with extension entry shown below and pass the reservation ID to 'qsub' command.

```
<extensions>
   <schedulerAttrs name="reservationID">
        xxxxxxxxxx
   </schedulerAttrs>
</extensions>
```

4.4 Advance reservation and execution via GT4 container

The following shows the steps to execute a job with advance reservation.

1. A client creates a PBSReservation service providing reservation timeframe and number of nodes, and gets an EPR for the service.

2. The client invokes operation 'reserve' on the EPR.

3. The client gets resource properties from the EPR to make sure that the reservation successfully completed.

4. The client gets reservation ID from the ReservationID resource property. reservation ID.

5. The client submits a job using GRAM client program with an RSL that embed the reservation ID in it.

5 Preliminary Evaluation

As a basic data of the proposed scheduler and the reservation service, we conducted a measurement for some basic reservation operations with 1) direct access through command line programs, 2) indirect access via the reservation service. Note that the reservation service is using the command line programs behind the scene.

[1]As of now, the latest release of the Globus Toolkit (4.0.2) does not includes this capability and administrators have to install update packages to have this. It will be included in the next release.

Table 4. Reservation operations latency.

	reserve [s]	cancel [s]
Direct Access	0.78	0.68
Via GT4 container	1.7	1.3

The testing set-up is as follows. We used a PC with dual Pentium III 1.4 GHz, 2Gbytes. All the participating components; service client, service container, PBS scheduling module, are all on the PC. We measured time spent to make reservation, and cancel it.

Table 4 shows the result of the measurement. Even with the direct access, it takes almost one second. We have investigated the reason and found that most of time is spent to load (and JIT compile) Java library modules.

The reservation service with GT4 container is posing extra half to one second. This is due to the authentication and authorization by the container.

The implication of these numbers varies depending on the usage of the service. It might be considered fast enough for one-shot reservation. It might be too slow if the co-allocation protocol requires multiple reserve/cancel operations on each resource.

6 Conclusions

We have designed and implemented a scheduler module as a testbed to study scheduling policies based on queuing based system with advance reservation. We provide easy to use API for the scheduler module and implemented advance reservation capability. Furthermore, we designed an external interface for advance reservation based on WSRF with GT4, that enabled reservation based job submission on a Grid, coordinated with GT4 GRAM.

The followings are our future work.

- **Grid Engine support:** We are now implementing a version that works with GridEngine[3]. The goal for this is that one scheduling policy written in Java API works with both TORQUE and GridEngine.

- **Study of scheduling policy:** In the current implementation, the reserved jobs are always have higher priority than the usual jobs, i.e. when the reservation time comes, the jobs already running on the reserved nodes will be just preempted. And there is no priority concept for making reservation. All the reservations from valid users will be accepted in First Come First Served base. It means one user can occupy all the resources forever by making a bunch of reservations. Obviously, this is not acceptable for the real usage. We will study this issue to find acceptable allocation policy for users and administrators.

- **Deployment and Operation in the real environment:** We are planning to deploy this framework with our super scheduler [14] on the PRAGMA[10] testbed to provide inter cite co-allocation for them. With the feedback from real operation experience, we will improve the software and scheduling policy.

Acknowledgement

This work is partly funded by the Science and Technology Promotion Program's "Optical Paths Network Provisioning based on Grid Technologies" of MEXT, Japan.

References

[1] db4objects. http://www.db4o.com/.
[2] Globus project. http://www.globus.org.
[3] Grid Engine. http://gridengine.sunsource.net.
[4] LSF. http://www.platform.com/Products/-Platform.LSF.Family/.
[5] Maui cluster scheduler. http://www.clusterresources.-com/pages/products/maui-cluster-scheduler.php.
[6] OpenPBS. http://www.openpbs.org/.
[7] PBS Professional. http://www.altair.com/software/pbspro.htm.
[8] TORQUE Resource Manager. http://www.-clusterresources.com/pages/products/torque-resource-manager.php.
[9] WSRF. http://www.oasis-open.org/committees /tc_home.php?wg_abbrev=wsrf.
[10] P. Arzberger and P. Papadopoulos. PRAGMA: A Community-Based Approach to Using the Grid. June 2004.
[11] S. Brunett, K. Czajkowski, S. Fitzgerald, I. Foster, A. Johnson, C. Kesselman, J. Leigh, and S. Tuecke. Application experiences with the globus toolkit. In *Proceedings of 7th IEEE Symp. on High Performance Distributed Computing*, July.
[12] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pages 2–13, 2005.
[13] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. *Grid Computing: Making the Global Infrastructure a Reality*, chapter The Physiology of the Grid. John Wiley & Sons Ltd, March 2003.
[14] A. Takefusa, M. Hayashi, N. Nagatsu, H. Nakada, T. Kudoh, T. Miyamoto, T. Otani, H. Tanaka, M. Suzuki, Y. Sameshima, W. Imajuku, M. Jinno, Y. Takigawa, S. Okamoto, Y. Tanaka, and S. Sekiguchi. G-lambda: Coordination of a grid scheduler and lambda path service over gmpls. In *Future Generation Computing Systems*, 2006.