High-Performance Computing Application Launching Environment Manual Version 1.0

Il-Chul Yoon, Norman Lo
Alan Sussman
Department of Computer Science
University of Maryland
College Park, MD 20742
{iyoon,normanlo,als}@cs.umd.edu

June 11, 2008

Contents

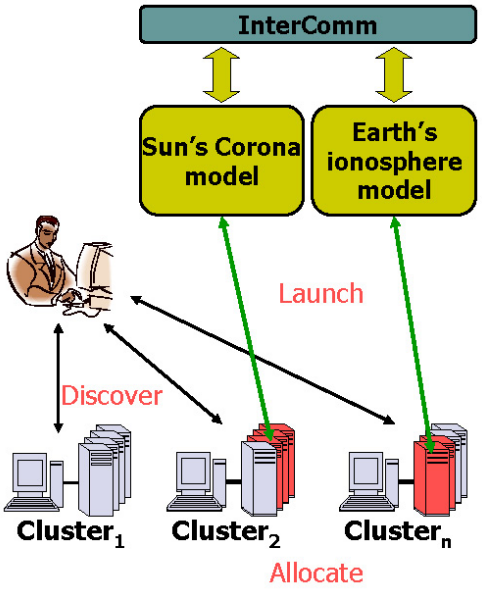

Figure 1: HPCA Launching Process

1 Overview

Many High-Performance Computing Applications (HPCA) need to be launched on multiple resources. One of the common scenarios to launch these applications, is allocating a set of computing resources via resource scheduler such as PBS [**?**]. Otherwise, the applications might run on a set of resources that a user specifies explicitly. The most complex situation is when the applications need to be launched on multiple resources, which is physically distributed, as shown in Figure 1.

In Figure 1, two separate applications run on multiple resources, in this case, clusters. While they are separate applications, they need to communicate each other to simulate the interaction between Sun's corona activity and the reaction of Earth's ionosphere. Thus, multiple resources must be allocated and runtime environment must be set up properly before launch for the applications, to ensure the successful execution. If users are not equipped with an automatic application launching environment, they must handle the whole process manually.

High-Performance Computing Application Launching Environment (HPCALE) is a convenient environment to handle this repetitive and burdensome work explained above. Specifically, it provides users with following functionalities based on the high-level job description written by users.

- Refine high-level job description specified by user
- Allocate multiple resources

- Prepare proper runtime environment for the applications
- Launch applications on specified resources
- Collect generated output files into client side
- Help resource administrators to manage resource and application information
- Help user to create customized job description semi-automatically

This manual is written for the HPCALE users and administrators. In Chapter 2, brief description on HPCALE working mechanism is described. Chapter 3 explains the input job description file for HPCALE, which is called XML Job Description (XJD) in detail. Chapter 4 explains how to install HPCALE, submit and terminate a job, and how to use the HPCALE Web interface to handle XJD and to manage repository, which contains all information on resources and software to be launched.

2 HPCALE Architecture

2.1 HPCALE At-A-Glance

Figure 2 shows how a job submitted by a user is handled by HPCALE. Based on the high-level job description, HPCALE handles all the low-level work listed in the figure necessary to launch the applications.

Figure 8 and 13 show Web interface to manage HPCALE repository and XJD. Resource administrator may use this interface interactively to add, modify or remove resource information in the repository, and user may use this interface to create or modify his job description.

The services shown in Figure 2 are implemented as Web-Services, and they are securely invoked from a HPCALE client. The Web server where the services are installed, is equipped with a certificate issued by a certificate authority. Since HPCALE trust a client only after the client passes 2-way SSL handshaking process with HPCALE server, a valid certificate must be issued to each client by the same authority that issues certificate of Web server. Each of the services are explained in the sections below. Figure 4 is a sequence diagram shows the basic interaction between HPCALE service entities to handle HPCALE services.

The services in Figure 3 are implemented with CGI scripts. Administrators can interactively add or update the resource and application information and users can use the information to build job descriptions. Current HPCALE version does not support dynamic resource status information. That is, up-to-date resource information is not maintained by HPCALE server. Instead, it is left to the dedicated resource schedulers.

2.2 Job Refinement

Users need to specify the applications to be launched and resources where the applications run on. However, they do not need to know all the details about the applications and resources. Job refinement service is

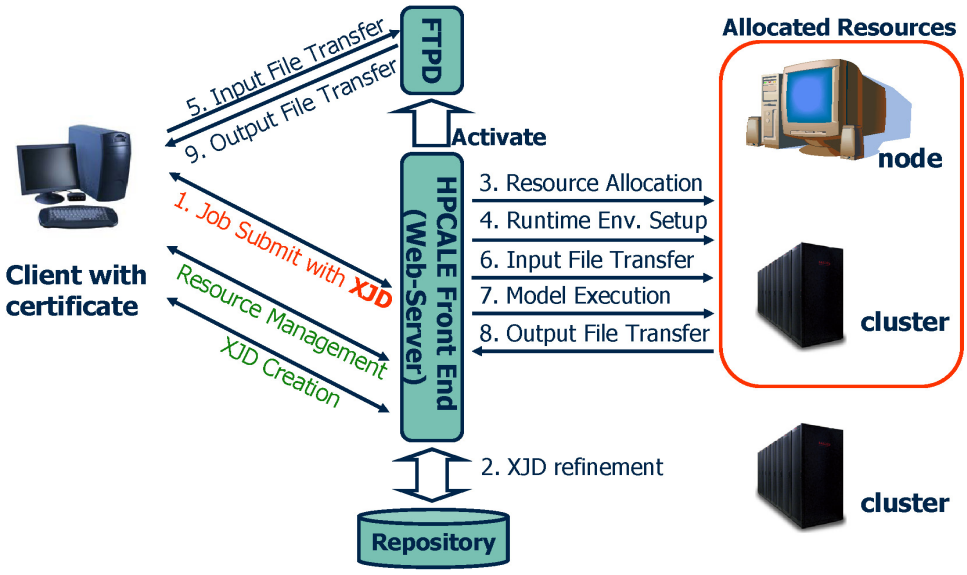

Figure 2: HPCALE Service Collaboration

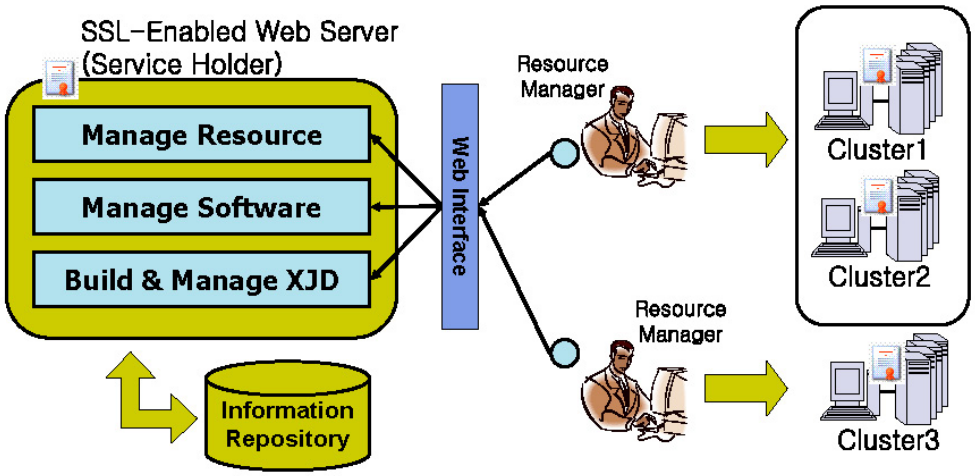

Figure 3: HPCALE Management Web Interface

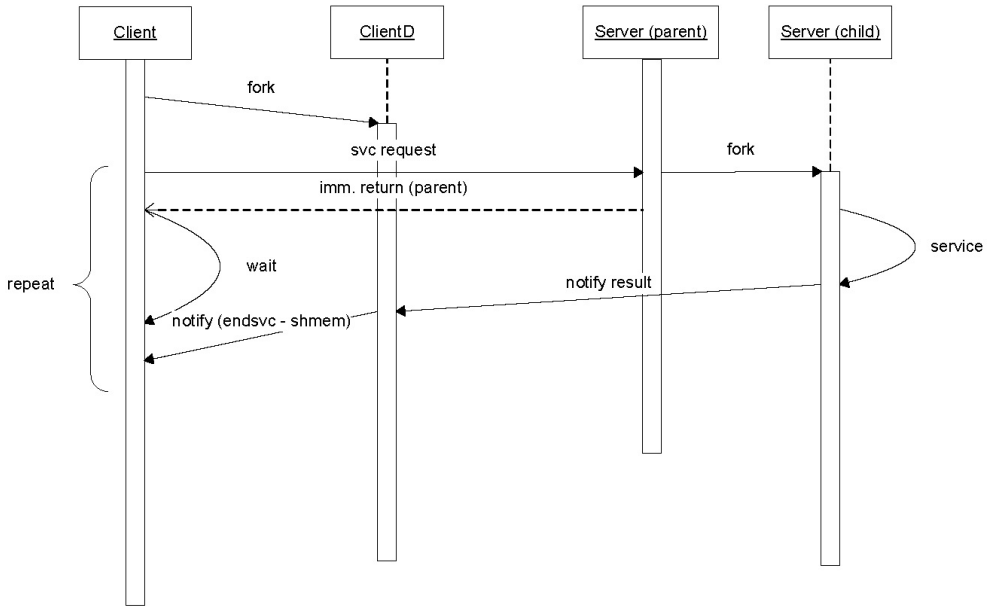

Figure 4: Basic service handling

to find detailed information on the specified application and resource and revise the given high-level job description.

High-level job description is the input to this service and refinement service tries to find matched information from HPCALE repository, and revises given job description into detailed job description. For example, user does not need to know the exact executable file location for an application and how to prepare runtime environment for the application. Instead, HPCALE can refer the repository to look for the detailed information. (The information must be registered into the repository by administrators.) Current version of HPCALE supports two resource types - *cluster* and *node*. Note that current version of HPCALE does not provide resource discovery itself.

2.3 Resource Allocation

To allocate resources manually, a developer must understand how the resources are managed, and must access the resource directly for allocation. This becomes more complex than this since each resource might have different allocation mechanism depending on the resource type.

In HPCALE, resource allocation is handled transparently by allocation service, and therefore, developer can focus on developing business logic. When a job description is submitted, HPCALE tries to allocate resources according to the resource type. For example, if a cluster must be allocated via PBS scheduler, HPCALE contacts to the PBS scheduler by sending proper resource allocation command to it, and holds the resources

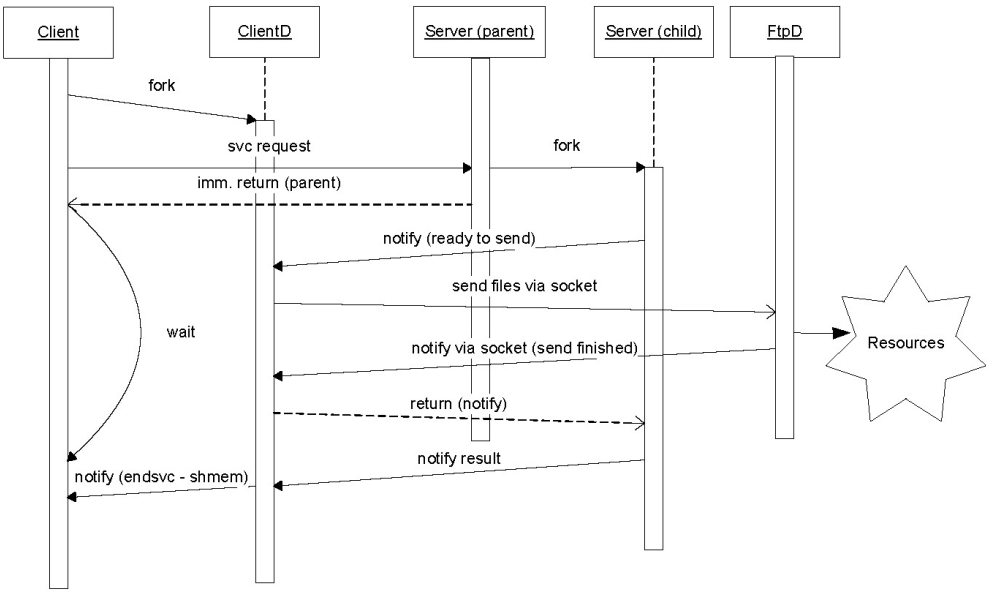

Figure 5: HPCALE File Transfer Handler Design

during the job execution.

HPCALE also provides limited resource co-allocation. If multiple applications in a job description asks for nodes from a single cluster, HPCALE tries to co-allocate the resources at once and distribute the allocated nodes to applications. However, the resources spanning multiple clusters or nodes are allocated one by one and HPCALE does not guarantee the co-allocation of the resources during application execution.

2.4 Runtime Environment Preparation

Resource allocation is only the starting point to launch applications. Each application might have own runtime requirements. A daemon might have to be launched before application or a runtime infrastructure might have to be ready for dynamic data exchange. The example applications in Figure 1 require InterComm [**?**] as a runtime environment since InterComm asks the resources to be joined into a single communication group using PVM [**?**, **?**].

Current version of HPCALE supports two handlers for such runtime requirements. First, PVM setup handler is provided to support InterComm applications. File transfer handler is to send/receive input and output files of an applications. STDOUT and STDERR streams generated from an application is also transferred to HPCALE client by this handler. Figure 5 is an example sequence diagram showing the HPCALE design to handle file transfer from a client to the remote sites.

2.5 Job Execution

Multiple applications may participate in a complex coupled simulation, and they are described in a job description. Since each of the applications may have different launching method, HPCALE needs to handle these different launching methods. For example, an application may be launched using MPI or another application may use only PVM.

HPCALE supports a number of launching mechanisms for the applications. Current version of HPCALE supports PVM application, MPI application via MPICH [**?**], and sequential application. HPCALE launches the applications one by one on the determined resources and records *stdout* and *stderr* streams to notify client later.

To decide application termination, HPCALE inspects the running process information on the assigned resources. Specifically, HPCALE sends standard UNIX *ps* command to each resource and analyzes the result. This is approach is very pragmatic but has limitations. For example, two different instances of an application cannot be discriminated if they are launched on a same resource. If a user asks for an application to run on a same resource, the requests will wait until both applications terminate. Moreover, if one execution falls into infinite loop while the other is running correctly, then both clients will not get the answer.

After job termination, the components in XJD may create output files. At least, they may generate *stdout* and *stderr* stream. HPCALE records these streams into separate files and transfers back to the client side with the output files from the components. The registered handler for runtime environment preparation is used for this process. The list of output files are listed in the file *recvfiles* tag specifies. Figure 6 is a sequence diagram showing the HPCALE design to transfer the output files back to the client side.

3 XML Job Description

XML Job Description(XJD) describes the applications a user wants to run for a (coupled) job, the resource for the applications and the connections between the application if necessary. In this section, we explain the structure of refined XJD in detail although user does not need to describe all the information in the XJD, the explained information might help users to understand the XJD and to customize it.

To generate an XJD, client may rely on the XJD Helper Web interface, which is a part of HPCALE distribution. Otherwise, he may manually write his own version of XJD. In either case, HPCALE will accept the description if it is a valid XJD, and HPCALE refines it if necessary. However, we recommend users to create XJD via the provided Web interface since it reduces potential conflicts by filling up XJD contents with the valid information retrieved from HPCALE repository.

An XJD consists of a list of components (applications) and a list of connections between the components. Following example shows a simplified XJD structure.

Example

```
<?xml version="1.0">
```

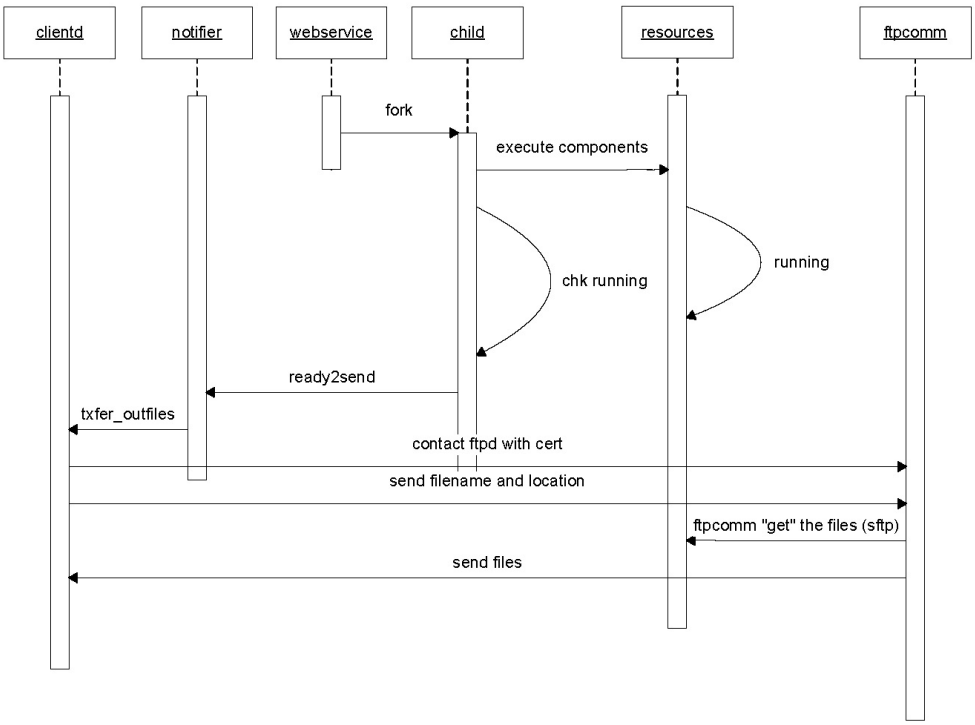

Figure 6: Back transfer output files to the HPCALE client

```
<ICXJD>
   <version>1.5</version>
   <components>
      <component>...</component>
      <component>...</component>
   </components>

   <connections>
      <connection>...</connection>
      <connection>...</connection>
   </connections>
</ICXJD>
```

3.1 Component

Component simply means an application deployed at a resource. HPCALE checks the validity of the component information in an XJD by comparing to the information retrieved from the repository. To run a component, detailed component information such as executable file name and location, must be specified in an XJD. However, HPCALE allows users to describe minimal component information in XJD as shown in the following example. HPCALE retrieves other information from the repository automatically.

Example component description

```
<component>
    <id>component1</id>
    <name>ring</name>
    <cluster>cluster1</cluster>
    <nNode>3</nNode>
</component>
```

The minimal description includes only an component identifier, a distinct name of a component, target resource name in XJD and the required resource amount if the resource type is *cluster*. Note that only the number of the required resources is described for a *cluster* resource type if it is managed by a local scheduler such as PBS. If the target cluster is not managed by a local resource scheduler, user may have to list explicit nodes in addition to the number of nodes. Below, we explain the items for a component. User may use this to write an XJD manually, or to customize an XJD generated semi-automatically via Web interface.

- ***id***
  A unique identifier for a component in a job description. Any string without blank characters is allowed.
- ***rcid***
  A unique identifier for a resource in the repository. HPCALE will identify this during refinement process, and user does not need to specify this.

- ***name***
  The registered component name. While registering a component, HPCALE repository administrator assigns a distinct name for each component. Thus, this must be unique in a resource boundary. HPCALE use this for XJD validation and refinement purpose.

- ***cluster***
  The registered cluster resource name. Each cluster resource is assigned a name by administrator during registration process. If a component is deployed on a cluster, the combination of *name* of the component and *cluster* is used as a key to find detailed information for the component.

  HPCALE supports two cluster types. One if a cluster managed by a PBS scheduler, and the other is a cluster just sharing user file system which users can access the member node via SSH without allocation.

- ***scheduler***
  In case of cluster resource, this means the type of local scheduler for the cluster. Current version of HPCALE supports *PBS* or *SSH* as scheduler type. If this value is *PBS*, it means that the cluster is managed by *PBS* cluster manager, and if *SSH*, it means that the cluster nodes share user file system, and the cluster does not have a specialized cluster manager.

- ***nNode***
  The number of necessary nodes to run a component. It must be used when a component is deployed on a cluster.

- ***nodeset***
  If *scheduler* value for a cluster is *SSH*, the cluster is not managed by a cluster manager as explained. Therefore, user must specify nodes for the component explicitly. *nodeset* is a comma-sperated list of cluster nodes. Be careful to specify only the node names. The domain for the nodes will be added automatically.

- ***path***
  The absolute path to an deployed component on a resource.

- ***file***
  The executable file name of a component on a resource.

- ***type***
  The executable type of a component. Current HPCALE version supports *MPI-executable*, *PVM-executable* and *Sequential*. HPCALE uses *type* to decide component launching mechanism.

- ***head***
  If a component is installed on a cluster, this is the front-end (or gateway) node of the cluster. Usually, it is the machine users login and invoke a local scheduler to allocate nodes. If the component is deployed on a node, this is the *hostname* matched with the node IP.

- ***parallel***
  The flag - *0* or *1* - indicating whether a component is parallel application.

- ***argument***
  The command-line arguments for a component. User can specify any arguments he wants, to launch the

component from a remote shell, except two reserved words, '__MFILE__' and '__RHOME__'. '__MFILE__' means automatically generated machine file by HPCALE, and '__RHOME__' means the remote job directory at the execution site. Whenever HPCALE see these reserved words in the *argument*, they will be replaced by proper value before launching.

- ***launchdir***
  By default, HPCALE tries to launch a component at the default remote job directory on resources, which is created by HPCALE automatically. However, due to implementation issue, some application must be launched at a specific directory the executable file resides or the directory where the necessary data resides.

  In these cases, user can set 'launch directory' to set the directory where he wants to launch the application. HPCALE will move to the directory before launching the component.

- ***launchprog***
  External launching tool for a component. Some of parallel components requires external tool such as *mpirun* to be launched. User can specify the absolute path to such external tool. If this is not specified, default launching tool found by HPCALE is used according to the component type. Note that default launching tool might not work for the component. For example, the *mpirun* of MPICH fails to launch component compiled with LAM/MPI.

- ***launcharg***
  User can specify additional arguments for the external launching tool. For example, *mpirun* allows user to provide optional arguments, which enables the target application be launched differently. User can use '__MFILE__' to describe the machine file.

- ***sendfiles***
  This item specifies a file name where input files for a component are listed. Thus, a file must be written manually for each component if necessary. HPCALE will transfer the listed files to the execution site automatically. It consists of the lines specifying the files to transfer, and each line must be in the following format.

  sendfiles entry format

  ```
  localhost:file1 , remotehost:file2
  ```

  In this format, *localhost* is a fixed value, but user can specify any remote host. HPCALE will forward the file to the specified *remotehost*. To locate the files to transfer, HPCALE supports both absolute and relative styles. If *file1* starts with the '/' character, obviously it means the absolute path to the file. If not, HPCALE tracks the file relatively from the directory the XJD resides. Similarly, if user specifies an absolute file path in *file2*, the file will be transferred to the directory. If not, it will be delivered to the relative path from the directory, which managed at remote execution site for each job. Lastly, '#' in the first column means that the line is a comment.

- ***recvfiles***
  If a user needs to receive output files generated by a component, he can list the files in a file and write the path to the file in this item. Similarly to the *sendfiles*, HPCALE transfers the files to localhost automatically. Each line in the *recvfiles* must be in the following format. The meaning of this format is intuitive and similar to the one for *sendfiles*.

recvfiles entry format

```
remotehost:file1 , localhost:file2
```

- ***node***
  If a component is deployed on a *node* instead of a *cluster*, user must use *node* and *domain* of the node to specify the resource. *node* is the first word of the full host name.

- ***domain***
  If a component is deployed on a *node* instead of a *cluster*, user must use *node* and *domain* to specify the resource. *domain* is the rest of the full host name except the *node*. So, if full host name is "test.umd.edu", *node* becomes *test*, and *domain* becomes *umd.edu*.

3.2 Connection

Connection is a mapping between data ports each component defines, and a mapping between *exportport* and *importport* works as a channel for data exchange between the connected components. The connection information is not directly related to launch components. That is, HPCALE launches components by only referring the *component* information in XJD.

However, the information on *connection* is necessary for XJD-aware components. If a component is XJD-aware, it accepts and parses *connection* information in addition to the *component* information, to set up the communication channel between components.

Currently, components using InterComm version 1.5 API, are XJD-aware, and such components must provide the information for *component* and *connection* in XJD. Below, we describe the elements for a connection in XJD.

- ***id***
  The unique identifier of a connection in a job description.

- ***type***
  The type of data to be exchanged. Current version of HPCALE allows following data types : *char, short, int, float, double*.

- ***commtype***
  The communication type of a connection. HPCALE A connection can be used to redistribute data between parallel components, or used to broadcast data from a component to other component. $M \times N$ or $1 \times N$ is allowed for this tag.

- ***msgtag***
  The unique message tag for the data between the exporter and importer component.

- ***exporter***
  The identifier of the component which exports the data in XJD.

- ***exportport***
  The name of data port exposed by exporter component. The name specified here is the port name exporter component use internally.
- ***importer***
  The identifier of the component which imports the data in XJD.
- ***importport***
  The name of data port exposed by importer component. The name specified here is the port name importer component use internally.

4 Administrator/User Manual

HPCALE runs on user privilege. Thus, a normal user can use HPCALE to provide other people to use his working environment for launching provided components. He can install HPCALE server first on one of the machine that MySQL and secure Apache is available. Then, any authorized user can install and use HPCALE client program from any machine if HPCALE server is accessible from the machine using HTTPS protocol.

Note that the components are launched under the user privilege who is running the HPCALE server. That is, the user who submits a job is NOT same to the user actually running the components in the job description. The user privilege running HPCALE is used to allocate and launch components after receiving the job description from external client (user).

In this section, we explain how to install and use HPCALE server and client. The installation process for the HPCALE components requires configuration file, which contains environment variables with proper values depending on the machine.

4.1 Downloading and Installation

4.1.1 HPCALE Server installation

To install HPCALE server, following applications must be installed on your system. In addition, you have to install a few additional Perl modules and your own certificate authority using OpenSSL. HPCALE checks the module existences during installation process.

- Apache Web server with HTTPS support via mod_ssl module.
- MySQL server
- OpenSSL
- Perl

Module Name	Purpose
DBI, DBI::mysql	To access HPCALE repository in MySQL database
Data::Dumper	To print debug information.
XML::Simple	To parse and write XJD
SOAP::Lite	To provide HPCALE Web service
CGI, CGI::Carp	To provide HPCALE Web service
Net::SSL	To provide secure communication
Net::Socket::SSL	To provide secure communication
LWP with HTTPS support	To provide secure communication

Table 1: Perl Modules for HPCALE Server

Web server is necessary because HPCALE client accesses HPCALE Web Services via HTTPS. You can customize and use the Web server that your system administrator have installed on your system, or install your own version. For more information to install or customize Apache Web server, refer [**?**].

MySQL is used for the HPCALE repository. Similarly to Web server, you may use preinstalled MySQL server or install your own MySQL. For detail, refer [**?**].

Certificate authority is needed to issue certificates for the Web server and for HPCALE clients. Issued certificates are used for *two-way handshaking* between the HPCALE client and Web server to verify each other. Of course, you can use a certificate issued from a trusted third-party certificate authority. However, if so, for each HPCALE client (for you or your colleagues), you have to use a certificate issued from the same certificate authority. Thus, we strongly recommend you to create your own certificate authority. Refer OpenSSL [**?**].

The additional Perl modules are to handle XML documents and to handle Web services. The installer scripts for the HPCALE server and client check the existence of necessary Perl modules during installation process. You can use previously installed library or install Perl modules under the directories you can access. In either case, you have to add the path to the PERL5LIB environment variable in your shell environment. To install additional your own Perl modules, you may use the command '*perl -MCPAN -e shell*'. For detail, refer the Comprehensive Perl Archive Network (CPAN) [**?**]. Table 1 shows the necessary Perl modules used by HPCALE server.

If you install and prepare all the prerequisite packages, then you may install the HPCALE server according to the following procedure.

1. Download server code (hpcale-server.tgz) from HPCALE distribution site.
2. Uncompress and untar the downloaded file.
3. Edit environment variables defined in *setenv_server.cfg* considering your system setting. Table 2 describes the environment variables in the configuration file.
4. Apply the modified environment variables by the command '*source setenv_server.cfg*'.

5. Run '*perl install-server.pl*' to deploy the server files.
   Install script creates necessary directories, copy server files under the location specified in the configuration file. And, for the HPCALE Web services, the file, '.hpcale-libpath' will be created under the directory '*$ENV{APACHE_HOME}/cgi-bin/HPCALE*'. This file contains the runtime information that HPCALE services need to use to handle client requests.

6. Import HPCALE repository schema. HPCALE provides a SQL script *HPCALE_DB.sql* under the directory *$ENV{HPCALE_SERVER_HOME}/server/db*. It contains commands to create database schema for HPCALE repository. After importing the schema into the database HPCALE refers, use HPCALE Web interface to register software or resource information.

4.1.2 HPCALE Repository Management Web Interface installation

HPCALE Repository Management Web Interface consists of HTML documents and CGI scripts in Perl. It helps user to manage *resource* and *application* information stored in the repository. User can create, update and remove information in HPCALE database through this Web interface. To install HPCALE Repository Management Web Interface, follow the procedure below. The environment variable *HTDOCS_HOME* is assumed to be set before.

1. Download server code (hpcale_repman.tgz) from HPCALE distribution site.

2. Uncompress and untar the downloaded file.

3. Edit configuration variables.
   All the configuration variables are defined in the '*config*' file. User may modify this file according to his working environment. Table 3 describes the variables. Installing XJD Creation Helper Web interface has same information. Although APACHE_HOME and HTDOCS_HOME are not included in the configuration file, they must be defined as environment variable separately.

4. Run '*perl install-server.pl*' to deploy the server files.

Web interface accesses the same repository HPCALE does. Thus, we recommend to install this interface on the same server, which HPCALE is installed. In addition, user must set proper values for the variable in the configuration file.

4.1.3 HPCALE XJD Creation Helper Web Interface installation

HPCALE XJD Creation Helper Web Interface consists of HTML documents and Perl CGI scripts. It helps user to create XJD semi-automatically by retrieving information from the HPCALE repository. User can search for the components he wants to launch and also make connections between the components through this interface. To install HPCALE XJD Creation Helper Web Interface, follow the procedure below. The environment variable *HTDOCS_HOME* is assumed to be set before.

Environment variable	Description
APACHE_HOME	Apache home directory
HTDOCS_HOME	HTML document root directory. By default, it is a directory such as *public_html* under user's home directory. If he installed his own Web server, it might be *htdocs* under the APACHE_HOME.
HPCALE_PERL_PATH	Path to the Perl executable for HPCALE
HPCALE_PM_PATH	Colon-separated list of paths to the Perl modules.
HPCALE_SERVER_HOST	Full host name HPCALE server is installed.
HPCALE_SERVER_DOMAIN	Domain of HPCALE server. This is part of HPCALE_SERVER_HOST excluding the first word specifying the host name.
HPCALE_SERVER_PORT	Port number of the HPCALE Web service
HPCALE_SVC	Fixed to *soapsvc*
HPCALE_SERVER_HOME	Directory HPCALE server is installed
HPCALE_SERVER_PATH	Same to HPCALE_SERVER_HOST
HPCALE_SERVER_DB	Repository name maintained by MySQL
HPCALE_SERVER_DB_HOST	Host name of the HPCALE repository
HPCALE_SERVER_DB_PORT	Port number to access the HPCALE repository
HPCALE_SERVER_DB_USER	User name to access the HPCALE repository
HPCALE_SERVER_DB_PASS	Password to access the HPCALE repository
HPCALE_LOGPATH	Path to store HPCALE job execution log
HPCALE_LOGFILE	Name of the job execution log file
HPCALE_DEBUG_LEVEL	Debug level to decide how verbosely HPCALE print execution log (0,1,2,3)
OPENSSL_BIN	Path to the *openssl* binary file
HPCALE_FTPD_CADIR	Path to the directory HPCALE server certificate resides
HPCALE_FTPD_CAFILE	Path to the HPCALE server certificate file
HPCALE_FTPD_HOST	Host name running HPCALE FTP daemon. For current HPCALE version, this must be same to HPCALE_SERVER_HOST
HPCALE_FTPD_PORT	Port number to HPCALE FTP daemon
HPCALE_FTPD_HOME	Directory to store client certificate during job execution
HPCALE_FTPD_LOGPATH	Directory to store the HPCALE FTP daemon execution log
HPCALE_FTPD_LOGFILE	Name of the FTP daemon execution log
HPCALE_JOBDIR	Directory to store temporary files for each job execution by HPCALE server. Input files for each component are also stored under this directory temporarily before being transferred to proper remote execution site.
HPCALE_REMOTE_JOBDIR	Directory name that will be created at the remote execution site. Under this directory, all files related to the job execution are located.
HPCALE_RESERVE_RETRY	Number of retries to allocate resource. After asking resource allocation, HPCALE checks the reservation status and waits for two seconds if the resources are not ready until given retry number reaches.

Table 2: HPCALE Server Environment Variables

Environment variable	Description
APACHE_SERVER	Full host name running Apache Web server
APACHE_PORT	Port number to access the Apache Web server
PERL_PATH	Path to the Perl binary executable
DB_SERVER	Full host name running MySQL
DB_PORT	Port number to access MySQL
DB_NAME	HPCALE database name
DB_USER	User name to access the HPCALE database
DB_PASSWD	Password to access the HPCALE database

Table 3: HPCALE Web Interface Configuration Variables

1. Download server code (hpcale_xjdhelper.tgz) from HPCALE distribution site.
2. Uncompress and untar the downloaded file.
3. Edit configuration variables defined in '*config*' file, which is same to the one used to install the repository management Web interface.
4. Run '*perl install-server.pl*' to deploy the server files.

Similar to the Repository Management Web Interface, this Web interface accesses the same repository HPCALE does. It is recommended to install this interface on the same server, which HPCALE is installed.

4.1.4 HPCALE Client installation

HPCALE client consists of the service to ask for HPCALE Web services and daemon to interact with HPCALE server after job submission. Client can only submit a job to HPCALE server via HPCALE client.

To install HPCALE client, follow the procedure below. However, similarly to HPCALE server installation, the Perl modules - XML::Simple and SOAP::Lite - must be installed before. Contrary to the server code, HPCALE client installer does not copy files into other directory. Instead, it only checks the existence of necessary Perl modules.

1. Download client code (hpcale-client.tgz) from HPCALE distribution site.
2. Uncompress and untar the downloaded file.
3. Edit the environment variables
   The environment variables for the HPCALE client are defined in *setenv_server.cfg*, and user must modify the values according to his environment. Table 4 describes the variables in the configuration. To set up the variables related to the certificate, user must know the information on the valid private key and certificate before installing HPCALE client.

Environment variable	Description
HPCALE_CLIENT_HOST	Full host name HPCALE client is installed.
HPCALE_CLIENT_HOME	Directory HPCALE client is installed
HPCALE_CLIENT_PORT	Port number of the HPCALE client daemon
HPCALE_CLIENT_LOGPATH	Path to store the client-side job execution log
HPCALE_CLIENT_LOGFILE	Name of the client-side job execution log file
HPCALE_DEBUG_LEVEL	Debug level to decide how verbosely HPCALE client prints client-side execution log (0,1,2,3)
HPCALE_CLIENT_SSLCERT	Path to the client's certificate, signed by HPCALE certificate authority
HPCALE_CLIENT_SSLKEY	Path to the client's private key
HPCALE_SERVER_HOST	Full host name HPCALE server is installed.
HPCALE_SERVER_PORT	Port number to access HPCALE Web service.
HPCALE_SVC	Fixed to *soapsvc*
HPCALE_FTPD_HOST	Full Host name running HPCALE FTP daemon.
HPCALE_FTPD_PORT	Port number to access HPCALE FTP daemon

Table 4: HPCALE Client Environment Variables

4. Apply the modified environment variables by the command '*source setenv_client.cfg*'.

5. Run '*perl install-client.pl*' to deploy the client files.

4.1.5 Locating SSL Certificate

Every HPCALE client requires a valid SSL certificate to access the services provided by HPCALE server. The certificate may be issued by a third-party trusted certificate authority, or user's own certificate authority. In either case, HPCALE server certificate and HPCALE client certificate must be issued by same certificate authority, which enables the cross validation.

Whenever a client requests a HPCALE service such as resource allocation, the client's certificate must be signed by same certificate authority, which signed the server certificate. Then, a HPCALE client can pass two-way SSL handshaking with a HPCALE server.

Although HPCALE client requires only its private key and certificate, HPCALE server requires certificate of certificate authority to validate the client and server certificate. These files are necessary to make Apache Web server work with HTTPS support. Refer Apache Web server manual to configure the information on the the server certificate, server private key, and certificate of certificate authority.

4.2 Submitting a job

If HPCALE server and client is installed properly, job submission is relatively simple. Following is the job submission command from HPCALE client. Note that this command must be executed under the *client* sub-directory of HPCALE client installation. The option '-x' is to specify a job description file.

Job submission command

```
perl launch.pl -x XJD
```

If this command is submitted, HPCALE client instantiates (forks) a daemon at client side for communication with HPCALE server, and tries to access HPCALE Web service to handle the submitted job. The necessary services include services for XJD refinement, resource allocation, runtime environment setup and component execution.

For each request, HPCALE client invokes a Web service at server side, and waits while the service undergoes. When the service is finished, the service contacts to the client daemon and invokes appropriate notification function according to the service result.

Client daemon wrote small piece of information in the shared memory to notify the HPCALE client the result. Finally, HPCALE client decides the reaction to the result notified by the client daemon. Figure 7 shows the collaboration among HPCALE components for each service request.

4.3 Terminating a job

It happens that the components run indefinitely or hang before exiting normally at the resources. Since user does not have the right to access the resources, he have to use a job termination command provided by HPCALE to terminate a submitted job.

Terminating a job means that all components inside a job description are enforced to be killed from the allocated resources and that all the resources for the job must also be released.

Like job submission, job termination is simple. User may run following command in any directory at the client machine if all the environment variables are set up properly. The option '-j' is to specify the job identifier (an integer) returned by HPCALE server.

Job termination command

```
perl stopjob.pl -j jobid
```

If this command is submitted, HPCALE client instantiate (fork) a daemon at client side for communication with HPCALE server, and tries to access HPCALE Web service to terminate the specified job.

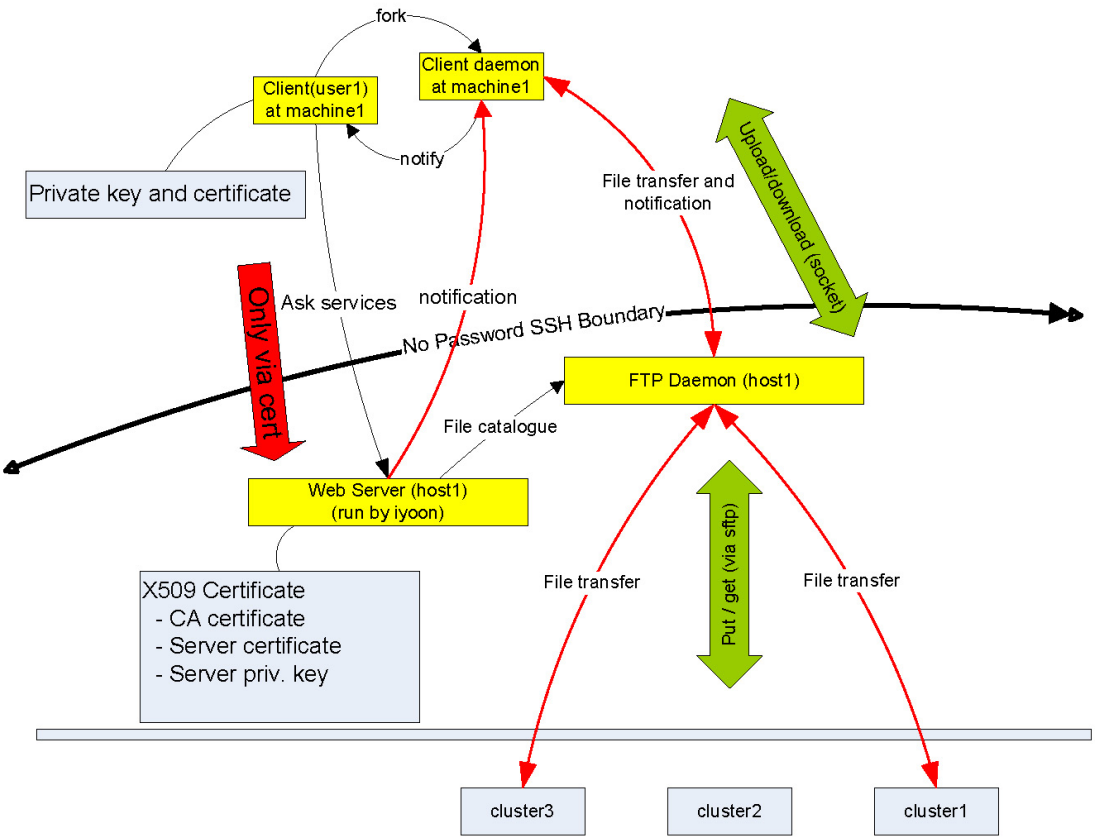

Figure 7: Collaboration of HPCALE components after job submission

The invoked Web service at server side tries to rebuild the job configuration. (The job identifier is used to reconfiguration process.) HPCALE server searches for the proper XJD in the HPCALE job directory linked to the job identifier, and parses the XJD to reconfigure the job. This reconfiguration includes the component and also resource information for each component.

Looping the resources, HPCALE server kills the job process until all the processes for the job is stopped in the resources. Then, HPCALE server contacts to the client daemon to notify the job termination.

4.4 Using XJD Creation Helper Web Interface

XJD Creation Helper Web Interface is developed to help users to create XJD in semi-automatic way, thereby reducing potential errors from user mistakes in XJD. The Web interface is provided separately to HPCALE, and consists of HTML documents and Perl CGI scripts. Thus, HPCALE administrator has to install it separately following the instruction in 4.1.3.

To use Web interface, user must enable the use of Javascript and Cookie in Web browser configuration because it uses Javascript to handle the information retrieved from the repository dynamically and a cookie to keep automatically-generated identifier for each XJD at client side.

Note that user also needs to store the XJD explicitly after editing. This is required since new identifier is generated and assigned to an intermediate XJD at server side whenever a user upload an XJD file to Web server. (The intermediate file is stored under the directory *$ENV{HTDOCS_HOME}/HPCALE_XJDHELPER/sessions* with the assigned identifier as file name.

4.4.1 Create and modify Component

Figure 8 shows the main screen of Web interface to create XJD or modify existing XJD. If user select 'Create new XJD', Figure 9 will be displayed. Two hyperlinks located in the left frame of Figure 9 show two methods to add a component into an XJD. User can also specify existing XJD stored at client side for modification. In this case, HPCALE parses the information on the components and connections in the XJD and displays in the browser.

After choosing semi-automatic creation (newcomp(semi-auto)), user may search appropriate component he wants to launch. For effective search, two wild card characters, '*' and '?', can be used. (In fact, these are the wild-card characters MySQL uses.) The former corresponds to any character string and the latter corresponds to any single character. The search result is shown as a table of components matched with given search string. For each component, it shows the component name in the repository, executable type of the component, and other default values for the application. Same application name may appear multiple times if they are installed on different resources.

Figure 10 shows the screen-shot to customize the selected component from search result. User may provide additional information necessary for the component. The description on the items listed in Figure 10 can be found in Section 3.

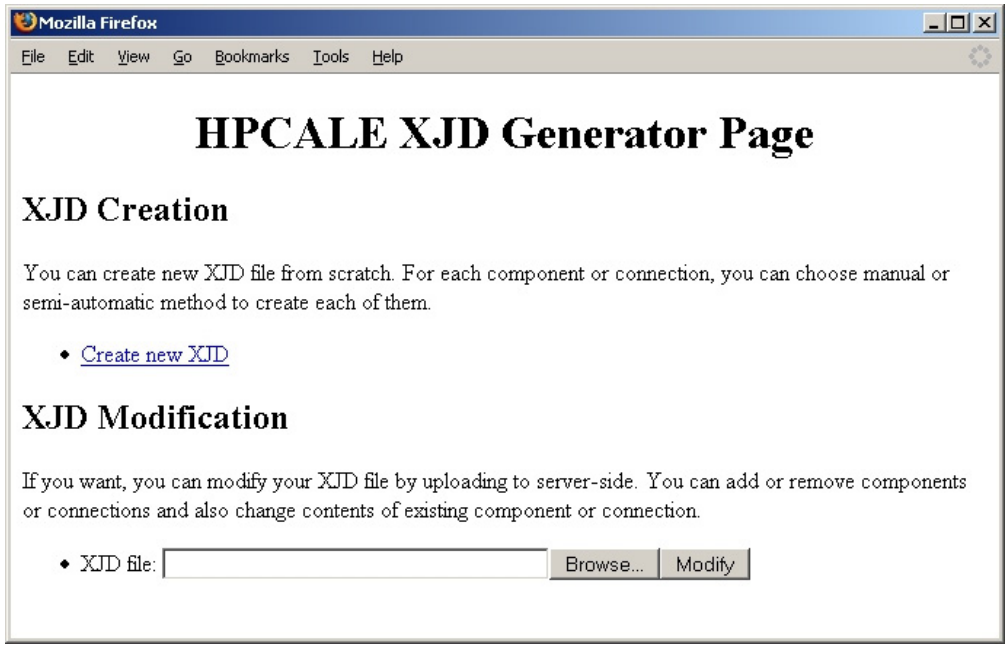

Figure 8: HPCALE XJD Management Homepage

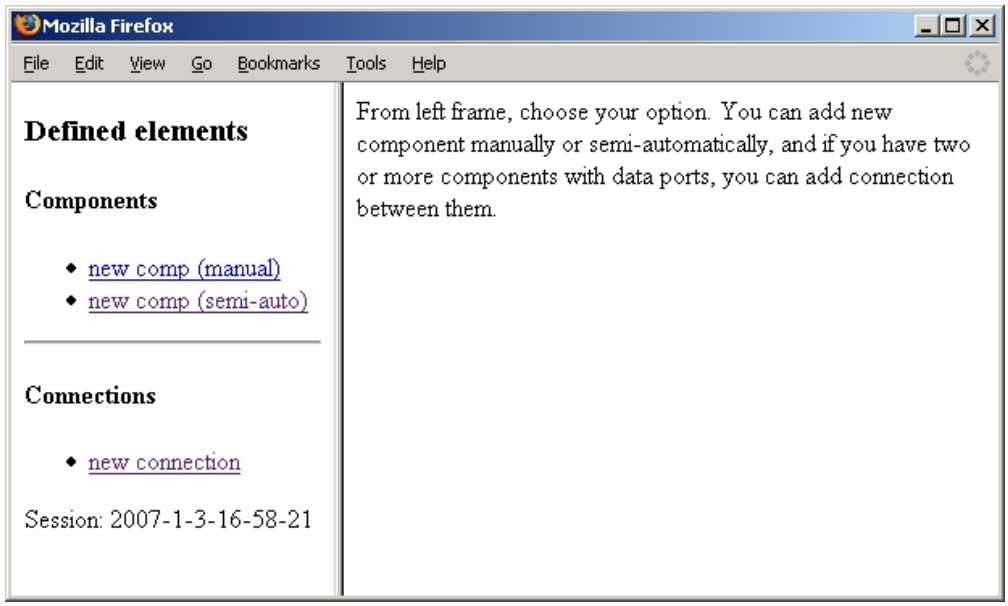

Figure 9: Create Component or Connection

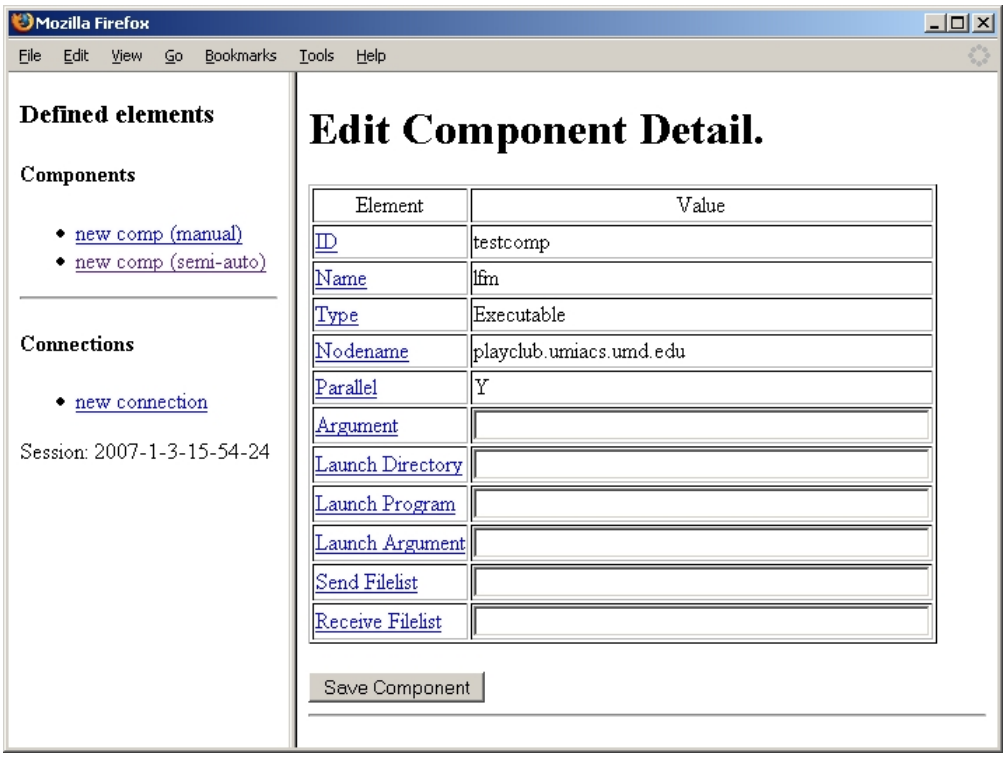

Figure 10: Component Detail

Note that not all items are same for all components. For example, *nodeset* may be displayed additionally if the component is installed on a cluster that user can access to its member node via SSH without restriction without allocation by a local scheduler. However, if a cluster is managed by a local scheduler, the item may not listed because the scheduler chooses and allocates nodes for the user.

Although semi-automatic component addition is recommended method, user can also add a component by providing all required information. If user chooses 'newcomp(manual)' link in Figure 9, user may see Figure 11, is similar to the Figure 10 but must be filled up manually by the user. Note that the information user provides must be matched with the information in the repository.

4.4.2 Modify/Remove Component

Added components are listed in the left frame of the Web browser. For example, two components are already added to the Figure 12. They are represented as a pair of user-specified component identifier and component name retrieved from repository, separated by a colon.

A customization screen similar to Figure 10 may show up when user clicks one of the listed components.

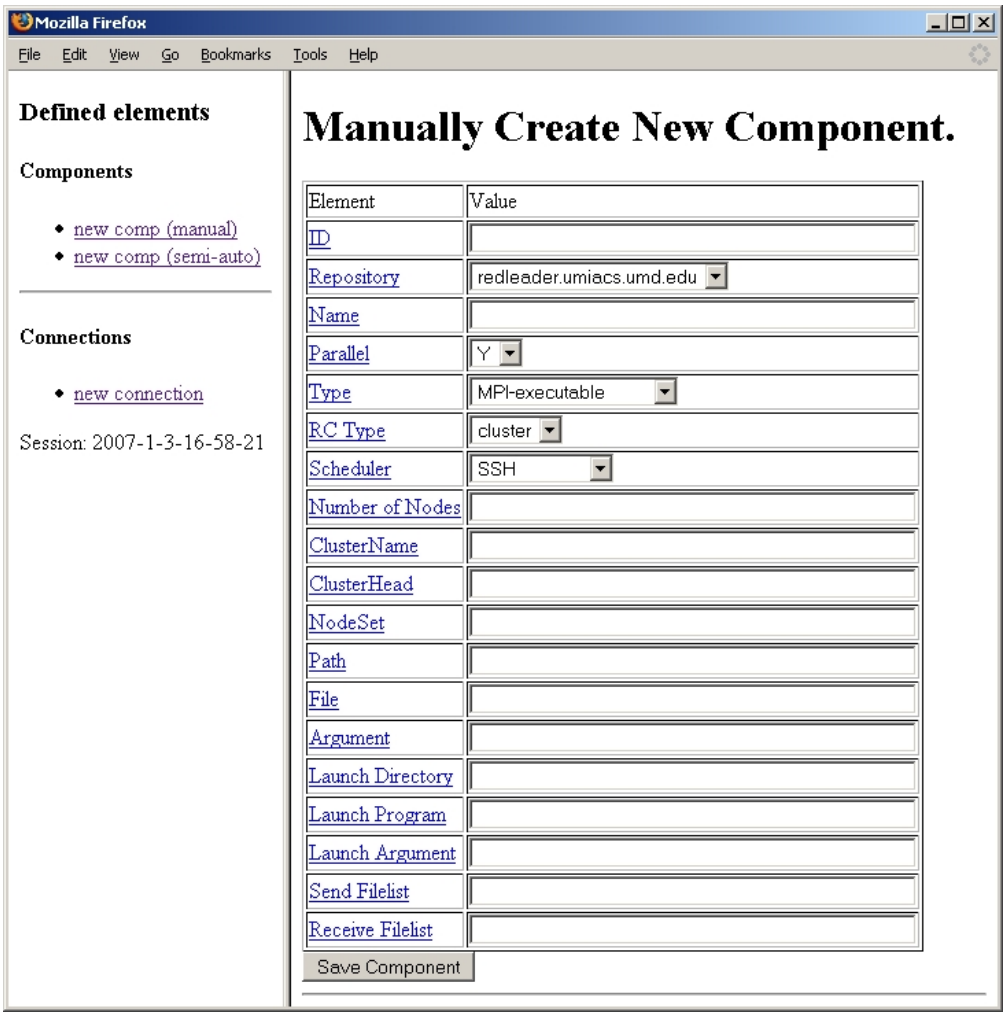

Figure 11: Manually Add Component

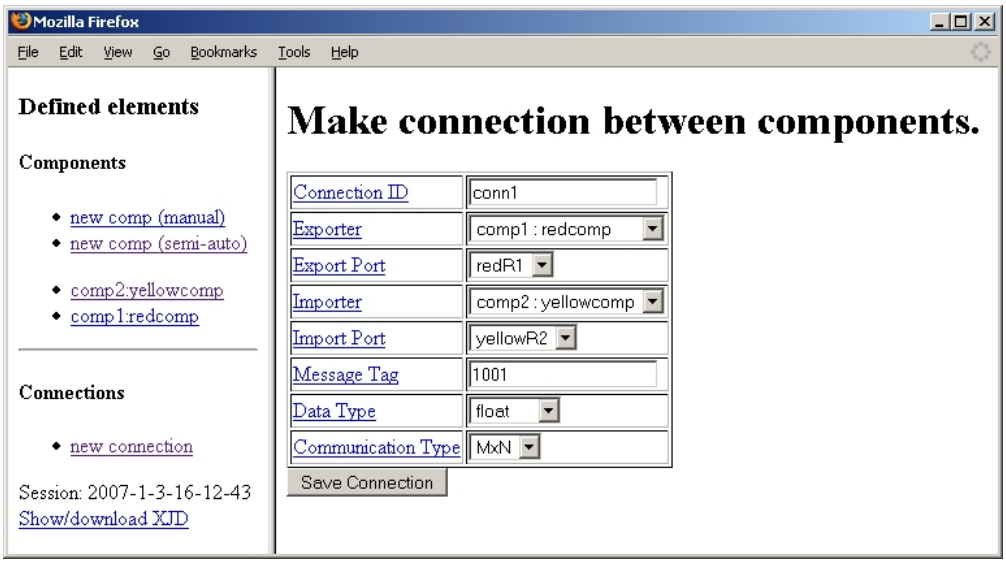

Figure 12: Add Connection

User may save the component after modification or remove from the XJD by choosing options at the bottom of the screen.

4.4.3 Insert Connection

A connection is used when multiple components need to exchange data with each other for coupled simulation. Currently, this information is only meaningful for the component which understand and uses XJD as component input (e.g. command-line argument). For example, InterComm 1.5 component is XJD-aware. It gets an XJD in initialization to figure out the data exchange pattern between participating components in computation.

As shown in Figure 12, user assigns a unique connection identifier and maps import and export port between components. Obviously, port information for each component must be preregistered when the component is added to the repository.

4.4.4 Modify/Remove Connection

As we explained before for the component modification, added connection will be listed in the left frame of the Web browser. By clicking a connection in the list, user may modify or remove a connection in similar fashion as we explained for adding new connection.

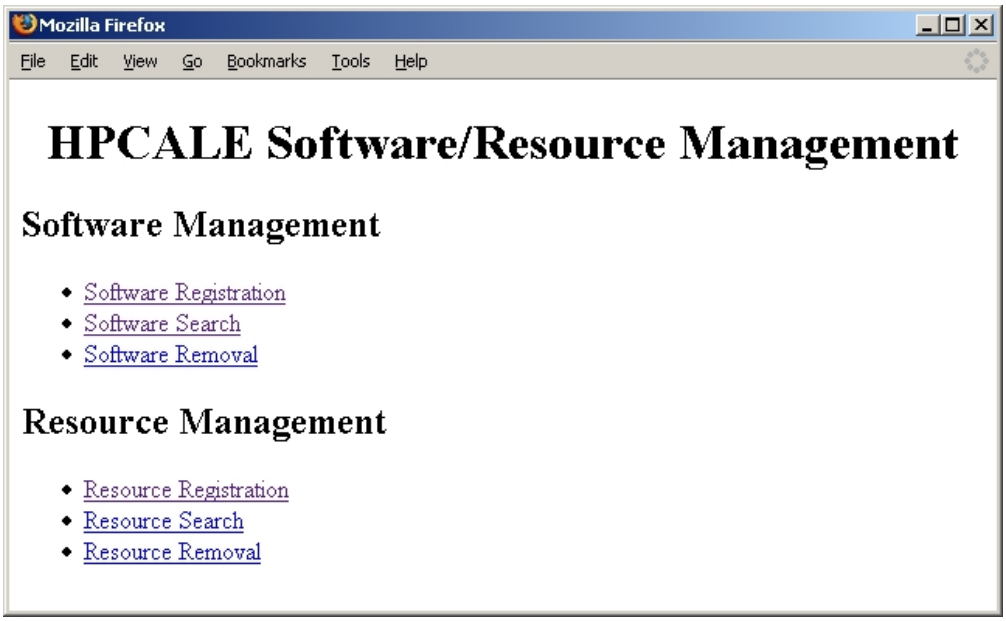

Figure 13: HPCALE Repository Management Homepage

4.5 Using XJD Repository Manager Web Interface

Figure 13 shows the main screen for HPCALE repository management. HPCALE administrator can manage the information on the software and resource inside the HPCALE repository through this Web interface.

4.5.1 Register Component

Figure 14 shows screen-shot to register a component into repository. Basic information to launch a component such as component name, executable path and executable file name need to be always decided for all components. 'Path' means the canonical path to the directory which the executable file of a component resides. 'file' is the executable name of the component.

For the XJD-aware components such as InterComm 1.5 component, 'ImportPortList' and 'ExportPortList' also need to be provided. Such components may define and expose its port names. Administrator provides the list of exposed import and export port names in comma-separated form without space. Other items in Figure 14 are described below.

'Runtime Environment' means other tools that must be set up before launching a component. For example, if a component is compiled by LAM/MPI library, LAM/MPI daemon need to be started before launching the component. And, it a component utilizes InterComm library, all participating nodes in the computation must be joined into PVM. During runtime environment setup phase, HPCALE handles this tedious and

Figure 14: Register Software Component

low-level work. Currently, HPCALE only supports InterComm and PVM as runtime environment.

'Resource' is a nickname of a resource and used in XJD. 'Manager Full Name' means the full name of the host which manages resource. Lastly, the 'DescriptionFile' is used to upload a HTML description for the component. Uploaded user-specified file is stored under the directory *$HTDOCS_HOME/swdesc* of Web server and is poped up to user when he selects the component in the component search result.

4.5.2 Modify/Remove Component

To modify component information, first, administrator must search component by component name, by choosing the 'Search Software' link in the Figure 13. The search result is in table format as shown in Figure 15 and administrator selects a component from the table for modification. The modification screen is same to Figure 14. After modification, he can commit the changes.

To remove a component from repository, follow the 'Software Removal' link in Figure 13, and search components. Removal is done by selecting components in the search result and pressing confirmation button. Multiple components can be removed at once by selecting multiple components in the search result.

4.5.3 Register Resource

HPCALE provides three types of resource to register - *node, cluster and manager*. *node* mean a machine the user running HPCALE server can access without allocation. A SMP machine might be a node example. HPCALE regards *cluster* type as a set of nodes which share user file system. Sometimes, they are managed by a local scheduler such as PBS. *Manager* is a node which is responsible for the other two types of resources. In case of a *node* resource type, the node itself is its manager. In case of *cluster*, the front-end node of the cluster becomes its manager. HPCALE always tries to access the cluster through the manager node for allocation.

Required information is different according to the resource type. For example, scheduler type has to be provided to register a cluster resource into repository as shown in Figure 16.

Note that resource manager must be registered before registering a resource. When administrator registers a node into repository, HPCALE uses provided node information to register the node also as resource manager first. However, when he registers a cluster, he must add the resource manager separately before registering the cluster. HPCALE checks whether the resource manager is valid, before registering the cluster information into repository.

4.5.4 Modify/Remove Resource

Resource modification and removal is implemented in similar fashion to that of component modification and removal. Refer Section 4.5.2.

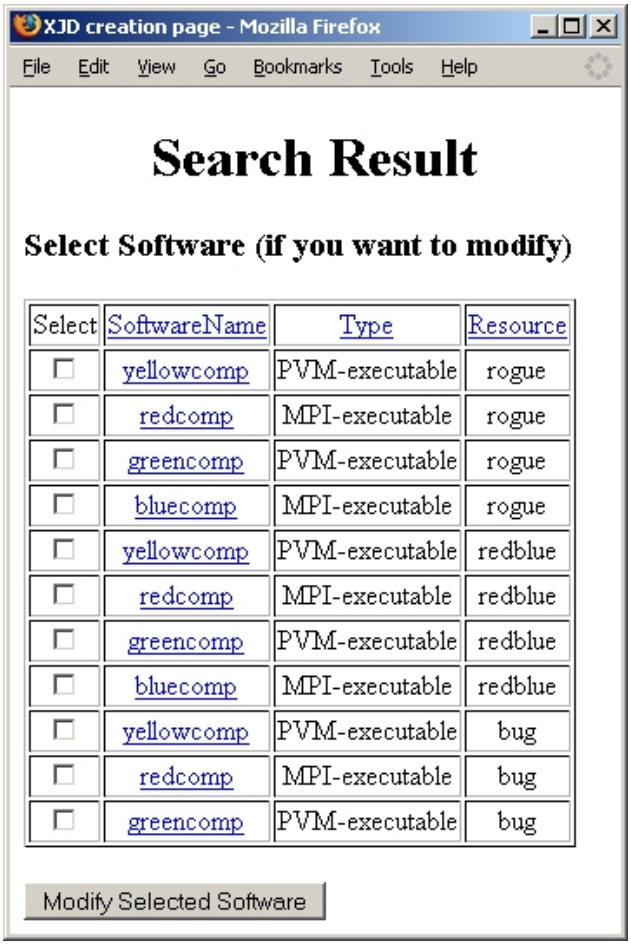

Figure 15: Component Search Result

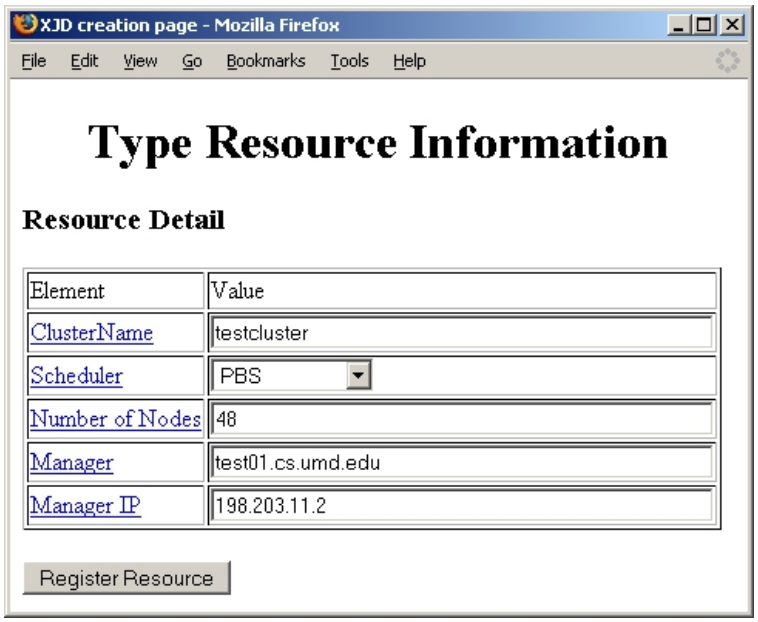

Figure 16: Register Resource