HPC Installation Guide

This guide will outline the steps to install the Web Service that will allow access to a remote resource (presumably a compute cluster). The Service runs within a Tomcat/Axis environment. The client application allows users to submit, retrieve job status, and save results from a remote job.

Step 1 – Prerequisites

Required Software

a. Java JDK

JDK 5.0 (recommended)
Download JDK 5.0 Update 11 (jdk-1_5_0_11-linux-i586.bin) from
http://java.sun.com/products/archive/

a) Install the SDK into a directory of your choice (/opt/java/jdk1.5.0_11, /opt/java/jdk1.6.0_02, or YOUR_JAVA_HOME_DIR)

b. Apache Tomcat 5.5 + JDK 1.4 Compatibility Package

The HPC cell runs under tomcat. Download 'apache-tomcat-5.5.23-compat.zip' and 'apache-tomcat-5.5.23.zip' from
http://archive.apache.org/dist/tomcat/tomcat-5/v5.5.23/bin/

a)Unzip into a directory of your choice (/opt/apache-tomcat-5.5.23 or YOUR_TOMCAT_HOME_DIR)

b) If default port 8080 is unavailable (another application is using this port), edit 'YOUR_TOMCAT_HOME_DIR/conf/server.xml' file to reconfigure the non-SSL HTTP/1.1 Connector to another port such as 7070 and the AJP 1.3 connector to another port such as 7009.

<!--Define a non-SSL HTTP/1.1 Connector on port 7070 -->

<Connector port="7070" maxHttpHeaderSize="8192"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
nableLookups="false" redirectPort="8443" acceptCount="100"
connectionTimeout="20000" disableUploadTimeout="true" />

<!--Define an AJP 1.3 Connector on port 7009 -->

<Connector port="7009"
nableLookups="false" redirectPort="8443"
protocol="AJP/1.3"/>

c. Apache Ant 1.6.5

Download 'Apache Ant version 1.6.5' (apache-ant-1.6.5-bin.zip) from
http://archive.apache.org/dist/ant/binaries/

a) Unzip into a directory of your choice (/opt/apache-ant-1.6.5 or YOUR_ANT_HOME_DIR)

d. Apache Axis2 1.1

Download 'Apache Axis2 version 1.1', from
http://ws.apache.org/axis2/download/1_1/download.cgi and select the download type

WAR (Web Archive) Distribution.(axis2.war)

a) Create folder axis2 inside 'YOUR_TOMCAT_HOME_DIR/webapps'

b) Unzip axis2.war inside 'YOUR_TOMCAT_HOME_DIR/webapps/axis2' folder.

c) In axis2/WEB-INF/conf/axis2.xml, set this parameter 'true' (it is false by default):

```
<parameter name="enableRESTInAxis2MainServlet"
locked="true">true</parameter>
```

e. Update your environment variables

Be sure to set the JAVA_HOME, ANT_HOME and CATALINA_HOME variables to the JAVA, ANT and Tomcat home directories you set up in steps a-c respectively. Examples are shown below.

```
# Sample environment variables
JAVA_HOME=/usr/java/jdk1.5.0_11
ANT_HOME=/opt/java/apache-ant-1.6.5
CATALINA_HOME=/opt/apache-tomcat-5.5.23
PATH=$PATH:$ANT_HOME/bin:$JAVA_HOME/bin
export CATALINA_HOME
export JAVA_HOME
```

Step 2 - Installing HPC software

Install HPC Web Service

Whether you intend to access a remote cluster via the i2b2 HPC cell or via the standalone java applicaton, you must install the web service into tomcat.

a) Download Web Service ClusterService.aar, which communicates with the remote resource (presumably a compute cluster) from http://i2b2.org.

b) Ensure Tomcat is not running: $CATALINA_HOME/bin/shutdown

c) Copy ClusterService.aar to $CATALINA_HOME/webapps/axis2/WEB-INF/services

d) Start Tomcat: $CATALINA_HOME/bin/startup

e) Verify webservice is running; substitute 'yourHost' and 'yourPort' with your local settings.
   a) Check url 'http://yourHost:yourPort/axis2/services/listServices' in a browser. Verify that HPC Service is listed as active.

Install HPC Standalone Java Jar

Perform this step if you plan on accessing remote resources through the command line, independent of i2b2 workbench.

a) Download AccessClusterStandalone.exe from http://i2b2.org, which will install a java application that allows access to the remote resource via calls to ClusterService.aar.
b) Run AccessClusterStandalone.exe to extract and install the java wrapper application. This will be installed in c:\Program Files\AccessCluster.

Install i2b2 HPC cell

Currently, the HPC client cell must be installed in an eclipse IDE where the i2b2 workbench is installed. These steps are described in **HPC Developer's Guide**.

Step 3 – Using the Web Service

From the command line

Create an xml file of commands to describe the task to be performed on the remote resource. A sample is provided in Appendix A.

a) Execute the following from the command line, with various options defined:
   -u = a valid user name on the remote resource
   -p = the corresponding password
   -h = host name of the remote resource
   -f = the filename of xml directives to run on the remote resource.

java.exe –jar "c:\Program Files\AccessCluster\AccessCluster.jar" -u <user> -p <password> -h <host> -f remote_application_file.xml

From i2b2 workbench

See **HPC Developer's Guide** for an example of usage within i2b2 workbench.

Appendix A

Example XML file to run as standalone input to AccessCluster.jar .

```
<?xml version="1.0" encoding="UTF-8"?>
<App>
<!--
"ClusterApp" is what will be submitted to the remote resource; specify
a meaningful name for the application, and a type of SUBMIT to indicate
you are submitting a job to the cluster
-->
<ClusterApp name="MyApp" type="SUBMIT" logging="true">
<!-- web service data, provide your local tomcat info here -->
 <local-resource>
  <server>localhost</server>
  <port>8080</port>
 </local-resource>

 <!-- "preprocess" is an optional element indicating one or more
commands to run on the remote machine before the execution of your
remote job -->
 <preprocess>
```

```
<!-- “concat” simply runs commands, in this example creating two
directories on the remote machine -->
  <concat>
   <command>mkdir testdata</command>
   <command>mkdir OA_output</command>
  </concat>
<!-- “ftp” will copy files from your local directory
(“c:\joe\mydata\*txt” in this case) to a directory “testdata” on the
remote machine; “testdata” is created under your home directory on the
remote machine -->
  <ftp action="put" remotedir="testdata">
   <fileset dir="c:\joe\mydata">
    <include name="*.txt"/>
   </fileset>
  </ftp>
 </preprocess>


<!--  at this point, you could submit a job to run on the remote
scheduler, or, submit a simple command -->
<!-- “submit” of type bsub (to submit a job to the LSF queue),
providing a fictitious java jar to run with parameters -->
 <submit type="bsub">
  <app name="java -Xmx1600M">
   <param>-jar app.jar</param>
   <param>config.xml</param>
  </app>
 </submit>

<!-- instead of a submit command, you could send a command and retrieve
its output, here asking to retrieve the status of the LSF queues on the
cluster -->
<!-- <command>bqueues</command> -->


</ClusterApp>
</App>
```

Once you have submitted a presumably long-running job on the cluster, you could periodically submit this XML to retrieve status:

```
<?xml version="1.0" encoding="UTF-8"?>
<App>
<ClusterApp name="MyApp" type="STATUS" logging="true">
```

```
</ClusterApp>
</App>
```

Then, when the STATUS command returns information that your job is done, you could save the output from the job to your local machine:

```
<?xml version="1.0" encoding="UTF-8"?>
<App>
<ClusterApp name="MyApp" type="SAVE" logging="true">
<output>c:\joe\savedata\result.txt</output>
</ClusterApp>
</App>
```