Development of Resource/Commander Agents

For AgentTeamwork Grid Computing Middleware

Final Report

Enoch Mak
Professor Munehiro Fukuda
CSS497 Faculty Research Internship
June 20, 2005

Table of Content

I. Introduction

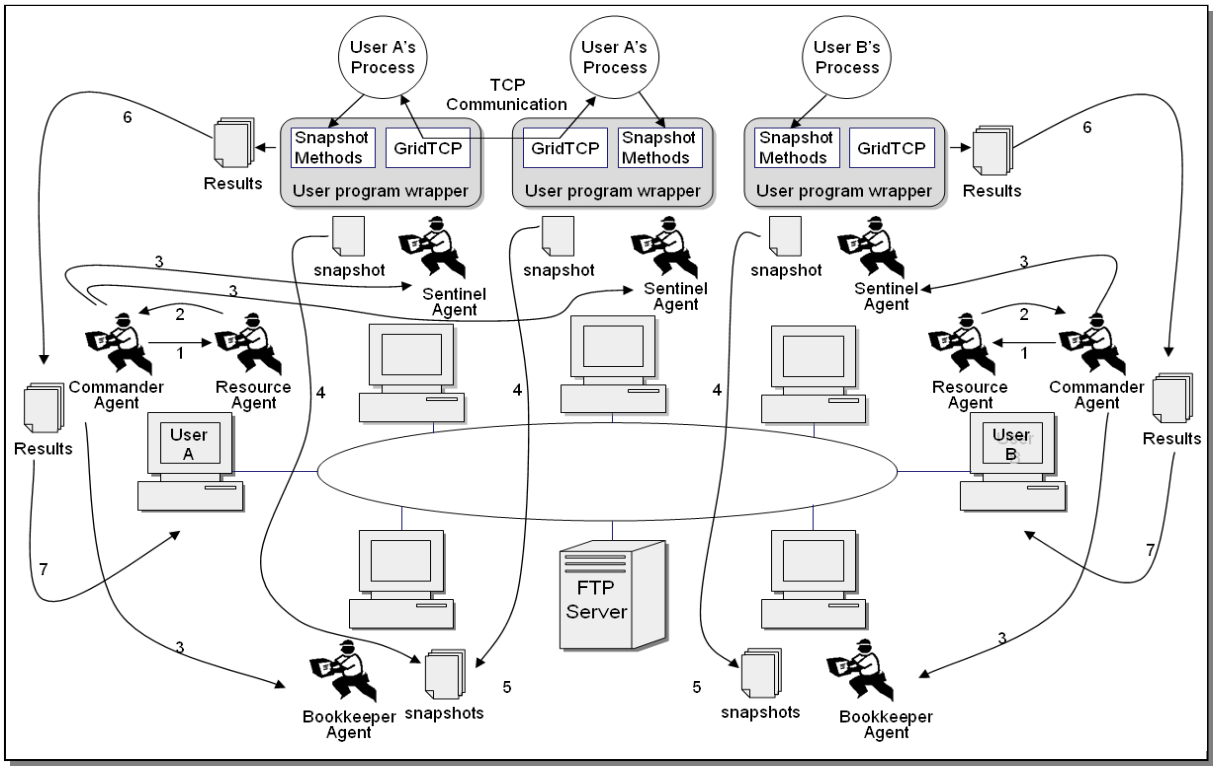
The UWB Distributed Systems Laboratory has been developing the AgentTeamwork grid computing middleware system that dispatches a collection of mobile agents to coordinate an execution of a user application over remote computing nodes in a decentralized manner. The ultimate focus is to maintain a high availability and dynamic balancing of distributed computing resources to a parallel-computing job.

The AgentTeamwork mobile-agent execution platform has been implemented, so that mobile agents are able to migrate over Internet and keep running on a different machine. All mobile agents are extended from the UWAgent class, instantiated as a Java thread, and executed on an UWPlace object which is part of the mobile-agent execution platform. The execution platform recognizes four types of mobile agents such as commander, resource, sentinel, and bookkeeper agents. Commander agent coordinates the work of other agents. Resource agent dynamically allocates distributed computing resource to a parallel computing job. Sentinel agent monitors and takes snapshot of the execution of user application at a different remote computer. Bookkeeper agent maintains the periodical execution snapshot from sentinel agent.

In this project, I implemented the first version of resource agent and enhanced part of the commander agent. AgentTeamwork is now able to perform a complete sequence of job execution:

1. an application is submitted,
2. a commander is spawned and send a resource query to the resource agent,
3. the resource accesses its local eXist DB, returns a list of remote computers, and performs periodic remote computers probing to update the local DB,
4. a commander dispatches sentinels and bookkeepers to launch and to monitor the application,
5. The commander receives the final computation results.

II. AgentTeamwork System Overview



AgentTeamwork targets a computational community agreed by a group of remote desktop owners and a common ftp server. The ftp server is used only to store a collection of XML-based user resource files. Once each computing node download and run the mobile agent execution platform in the background, each computing node is able to dispatch and to accept user jobs. In AgentTeamwork, each user can locally submit a job with a commander agent. This agent starts a resource agent that searches its local XML files for the computing nodes which are best fitted to the user job's resource requirements. The resource agent will send a list of remote computers' IP name to commander agent. Then, the commander agent will spawns as many sentinel and bookkeeper agents as the number of nodes required for the job execution. Each sentinel launches a user program wrapper that starts a user process and records its execution snapshots. Such snapshots are sent to and maintained by the corresponding bookkeeper agent for the purpose of retrieving a user process upon its accidental crash. Each agent can also migrate to an idle and faster machine individually. At the end, all results are forwarded to the commander agent that thereafter reports to the user.

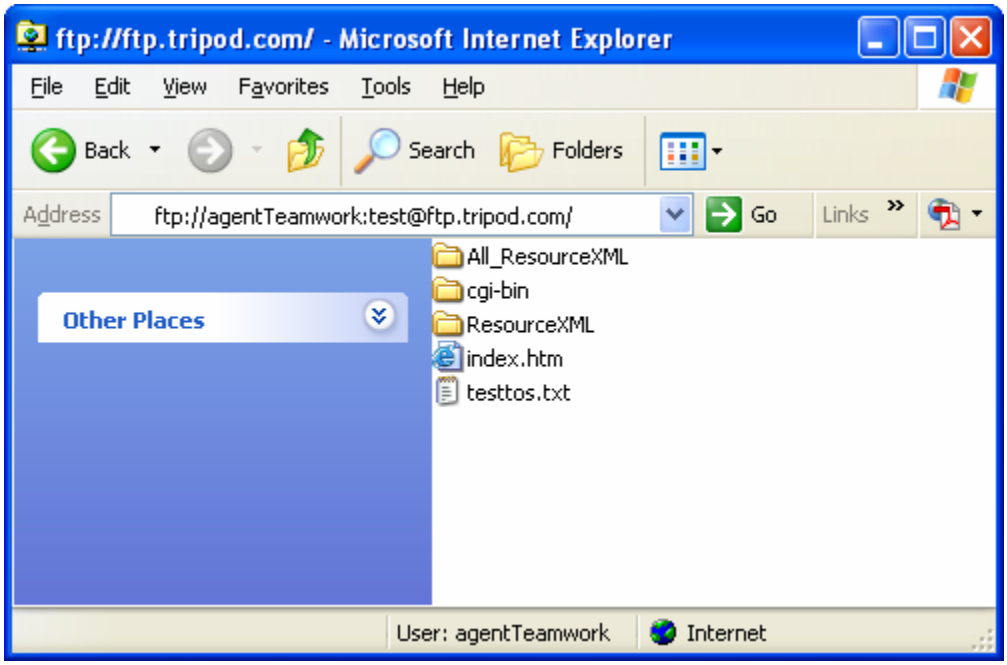
III. Ftp Server

AgentTeamwork need a common ftp server to store a collection of XML-based remote computing node resource files. Each XML file contains the resource information of a single remote computing node. For example, mnode0.xml contains the resource information of a remote computing node with IP name mnode0. Currently, the medusa cluster has 32 slave nodes, mnode0- mnode31. In order to let resource agent to identify all those 32 slave nodes, the FTP server need to maintain 32 xml files which are mnode0.xml – mnode31.xml. During the startup of a user application, the resource agent will download any new or updated XML files into the local database.

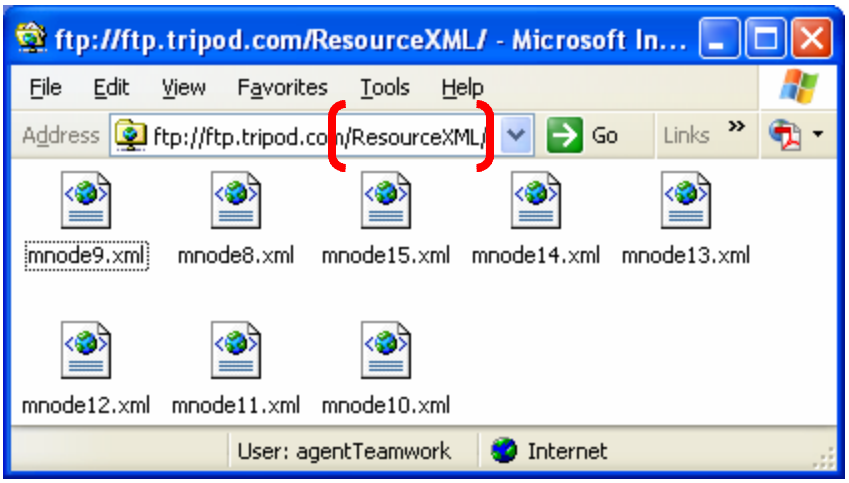
Currently, ftp.tripod.com is the FTP server being used to store the XML resource files. Below is the user name and password to access to the FTP server.

Host name: ftp://ftp.tripod.com
User Name: agentTeamwork
Password: test

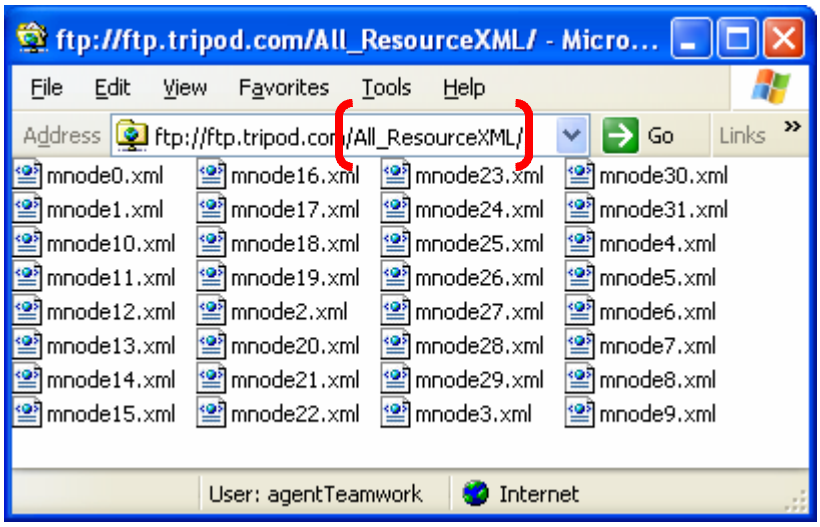
Below is the file listing of the FTP server.



Directory “ResourceXML” stores the XML files which will be downloaded by resource agent. Currently, it only stores 8 XML files, mnode8.xml – mnode15.xml, instead of all 32 xml files for testing purpose. Thus, only 8 remote computing nodes are available for user application.



A complete collection of 32 XML files can be found in directory “All_ResourceXML” as shown in below. In order to make all 32 remote computing nodes available to user application, one can copy all xml files from directory “All_ResourceXML” to directory “ResourceXML”.



IV. XML Database

A. eXist XML Database System

Resource agent stores the resource information of all remote computing node in it local database. Resource agent uses a DBMS (database management system) called eXist. eXist is a open source native XML database. For more information about eXist, please visit http://exist.sourceforge.net/ .

When writing Java application that accesses eXist database, it is important to understand on how XML database are organized. The concept of drivers, collection, resources and services are very important. Refer to the developer guide http://exist.sourceforge.net/devguide.html . In the developer guide section 3, there are some detail sample codes on how to write a java application with the XML:DB API to access eXist database. XML:DB API provides a common interface to native or XML-enabled databases. The javadocs of the XML:DB API can be found here http://exist.sourceforge.net/api/index.html , inside the org.exist.xmldb package.

B. XCollection eXist-interfacing class

A general eXist-interfacing class, XCollection.java, was implemented. Resource agent can access the eXist database through this interface class. XCollection.java equipped with the following features:

1. Create and remove collections
2. Access to multiple collections
3. Retrieve, store, remove, query and update XML files

C. Database Server Deployment

eXist offers three alternatives to run the database. It may either run as a standalone server process, embedded into an application or in connection with a servlet engine. In the beginning of my implementation, the eXist database was embedded into our java application. In order to make the eXist accessible by multiple resource agents concurrently, eXist database was eventually deployed as a standalone server process. Either the eXist is run as a standalone server or is embedded into an application, our eXist-interfacing class (XCollection.java) is able to access the eXist through the standard XML:DB API. The XML:DB API uses URIs to locate a collection of XML resources on the server. The URI identifies the name of the collection and the way of the server deployment. For example, the URI

xmldb:exist:///db/resources

references the “resources” collection on an embedded instance of the database. When the eXist is deployed as a standalone server, the URI would change to

xmldb:exist://localhost:8081/db/resources (This is the current URI used in XCollection.java.)

which means the “resources” collection on a standalone server which is accessible at localhost through TCP port 8081.

D. Local Database Collections

Two database collections, “resources” and “probinginfo”, are maintained in the eXist database for resource agent. The “resources” collection stores multiple xml files (mnode0.xml – mnode31.xml) which contain the resource information of the corresponding computing nodes. The “probinginfo” collection only stores one xml file (pinfo.xml) which contains the probing frequency of the current primary resource agent. It helps the resource agents to determine which resource agent is the primary agent. Only the primary agent needs to probe the remote computing nodes. For more detail information, please read the Remote probing section of this document.

E. How to manually delete all XML file in the eXist DB

User can delete the entire data set stored inside eXist database for testing purpose, it could be done by the following steps.

1. find out the location of the eXist database, which is currently located at “/home/uwagent/eXist”
2. go to “/webapp/WEB-INF/data” under the directory of the eXist database. In this case, the complete path is “/home/uwagent/eXist/webapp/WEB-INF/data”.
3. delete all files inside this directory.

F. XML File Structure

1. “probinginfo” Collection

Below is the XML file structure of the pinfo.xml which is stored in “probinginfo” collection. Note that user does not need to manually create this XML file. Resource agent will create this file automatically if it is not exist in the “probinginfo” collection.

```
<?xml version="1.0"?>
<probing_status>
    <frequence>0</frequence>
</probing_status>
```

This XML file stores only the probing frequency of the current primary resource agent. The <frequence> element store the probing frequency in milliseconds.

2. “resources” Collection

Below is the XML file structure of the mnode8.xml which is stored in “resources” collection. User needs to manually create a XML file with the related information for each remote computing node and store it in the FTP server. For more information, please see the FTP Server section of this document.

```
0   <?xml version="1.0"?>
1   <resource>
2       <design_time>
3           <human_owner>uwb</human_owner>
4           <ip_name>mnode8</ip_name>
5           <ip_addr>216.186.73.31</ip_addr>
6           <cpu_speed>8000</cpu_speed>
7           <cpu_arch>i386</cpu_arch>
8           <cpu_count>1</cpu_count>
9           <memory>64</memory>
10          <os_type>linux</os_type>
11          <disk_space>200</disk_space>
12          <cpu_load>100</cpu_load>
13          <intra_net_band>100</intra_net_band>
14          <availability multiple="true">
15              <time>0000-0530</time>
16              <time>0600-1000</time>
17              <time>1030-1730</time>
18              <time>1800-2359</time>
19          </availability>
20          <time_zone>pacific</time_zone>
21          <inter_net_device>myrinet</inter_net_device>
22          <inter_net_band>100</inter_net_band>
23          <intra_net_device>myrinet</intra_net_device>
24          <libraries multiple="true">
25              <name>cexec</name>
26              <name>mpirun</name>
27          </libraries>
28      </design_time>
29      <run_time>
30          <cur_ip_addr>216.186.73.31</cur_ip_addr>
31          <cur_cpu_speed>8000</cur_cpu_speed>
32          <cur_cpu_arch>i386</cur_cpu_arch>
33          <cur_cpu_count>1</cur_cpu_count>
34          <cur_memory>64</cur_memory>
35          <cur_os_type>linux</cur_os_type>
36          <cur_disk_space>200</cur_disk_space>
37          <cur_cpu_load>100</cur_cpu_load>
38          <cur_intra_net_band>100</cur_intra_net_band>
39          <cpu_utilization>1</cpu_utilization>
40          <process_count>0</process_count>
41          <cur_inter_net_band>100</cur_inter_net_band>
42      </run_time>
43  </resource>
```

Below is the table to explain the above XML file structure line by line.

Line no.	Description
0	The XML declaration
1	Root element “resource”, all elements inside this element are resource information
2	All elements inside this element are resource information at design time
3	Currently not used by Resource agent, keep for future development
4	The actual IP name of the computer node
5	The actual IP address of the computer node
6	The cpu speed of the computer node in MHZ, should be the same as line 31
7	The cpu architecture, should be the same as line 32
8	The number of cpu of the computer node, should be the same as line 33
9	The number of available memory in MB, should be the same as line 34
10	The operating system type, should be the same as line 35
11	The number of available disk space in MB, should be the same as line 36
12	The percentage of CPU idle, should be the same as line 37
13	The bandwidth of the network in MB, should be the same as line 38
14	All elements inside this element is the available time slot of the computer node
15	This is the first time slot
16	This is the second time slot
17	This is the third time slot
18	This is the fourth time slot
19	All elements inside this element is the available time slot of the computer node
20	Currently not used by Resource agent, keep for future development
21	Currently not used by Resource agent, keep for future development
22	Currently not used by Resource agent, keep for future development
23	Currently not used by Resource agent, keep for future development
24	Currently not used by Resource agent, keep for future development
25	Currently not used by Resource agent, keep for future development
26	Currently not used by Resource agent, keep for future development
27	Currently not used by Resource agent, keep for future development
28	All elements inside this element are resource information at design time
29	All elements inside this element are resource information at run time. It will be updated by Resource agent during run time.
30	The actual IP address of the computer node
31	The cpu speed of the computer node in MHZ, should be the same as line 6
32	The cpu architecture, should be the same as line 7
33	The number of cpu of the computer node, should be the same as line 8
34	The number of available memory in MB, should be the same as line 9
35	The operating system type, should be the same as line 10
36	The number of available disk space in MB, should be the same as line 11
37	The percentage of CPU idle, should be the same as line 12
38	The percentage of CPU idle, should be the same as line 13
39	Currently not used and not be updated by Resource agent, keep for future development
40	Currently not used and not be updated by Resource agent, keep for future development
41	Currently not used and not be updated by Resource agent, keep for future development
42	All elements inside this element are resource information at run time. It will be updated by Resource agent during run time.
43	Root element “resource”, all elements inside this element are resource information

V. How to start AgentTeamwork

This section describes the procedures needed to start the AgentTeamwork.

A. A list of all necessary file

This section briefly describes all files which are related to this project. Since I spent a lot of time to start this project, I think it is a good idea to document every single file that I have been used or updated.

Files Description	
File Name:	ResourceAgent.java
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the source file of the ResourceAgent
File Name:	XCollection.java
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the source file of the XCollection class
File Name:	CommanderAgent.java
Location:	/home/uwagent/enoch/agents
Description:	This is the source file of the CommanderAgent
File Name:	cr.sh
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the shell script to compile the ResourceAgent, all the necessary classpaths for compilation are included in this script. After the compilation, the file ResourceAgent$RemoteRscProbeTask.class will be copy to /home/uwagent/enoch/agents for CommanderAgent.
File Name:	cx.sh
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the shell script to compile the XCollection, all the necessary classpaths for compilation are included in this script
File Name:	cc.sh

Location:	/home/uwagent/enoch/agents
Description:	This is the shell script to compile the CommanderAgent, all the necessary classpaths for compilation are included in this script
File Name:	compile
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the shell script to compile the entire UWAgent directory and create an update UWAgent.jar, all the necessary classpaths for compilation are included in this script
File Name:	compile
Location:	/home/uwagent/enoch/agents
Description:	This is the shell script to compile the entire agents directory, copy the new ResourceAgent.class and sub class from UWAgent directory, and create an update agents.jar, all the necessary classpaths for compilation are included in this script
File Name:	run.sh
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the shell script to start the RMI server and UWPlace. A port number need to be provided when running this script file. For example, “run.sh 35353”
File Name:	runittry.sh
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the shell script to inject the CommanderAgent and start the ResourceAgent without any user application. This shell script is for testing only.
File Name:	runtestnow.sh
Location:	/home/uwagent/enoch/UWAgent
Description:	This is the shell script to inject the CommanderAgent and start the ResourceAgent with user application Series.
File Name:	RTruntestnow.sh

Location:	/home/uwagent/enoch/UWAgent
Description:	This is the shell script to inject the CommanderAgent and start the ResourceAgent with user application RayTracer.
File Name:	ResourceAgent$RemoteRscProbeTask.class
Location:	/home/uwagent/enoch/agents
Description:	This is the sub class of ResourceAgent. In order to start the AgentTeamwork correctly, this file should be resided in the same directory with CommanderAgent.
File Name:	ttcp
Location:	/home/uwagent/enoch/UWAgent and /home/uwagent/enoch/agents
Description:	This is the ttcp program for the ResourceAgent to run the bandwidth testing.
File Name:	ttcp
Location:	/home/uwagent/enoch/ttcp
Description:	This is the source code (C++) of the modified ttcp program.
File Name:	server.sh
Location:	/home/uwagent/eXist/bin
Description:	This is the shell script to start the eXist database standalone server.
File Name:	shutdownserver.sh
Location:	/home/uwagent/eXist/bin
Description:	This is the shell script to shutdown the eXist database standalone server.
File Name:	eXist database directory
Location:	/home/uwagent/eXist/webapp/WEB-INF/data
Description:	This is the directory which eXist store all the xml file. To clear all xml file inside the eXist, simply just delete all file in this directory. Please read the section IV E in this document.
File Name:	mnode0.xml – mnode31.xml

Location:	ftp://agentTeamWork:test@ftp.tripod.com/ResourceXML
Description:	All the xml files are located in the ftp server

B. Make sure FTP server have the necessary XML files

Check to make sure the FTP server (ftp://ftp.tripod.com/ResourceXML) has the necessary XML files. If not, simply copy it from (ftp://ftp.tripod.com/All_ResourceXML) which includes all XML files for mnode0 to mnode31. For more detail, please read section III of this document.

C. Start and Shutdown eXist Standalone Database Server

1. To start the eXist Server

   Go to the following directory "/home/uwagent/eXist/bin" and use the shell script server.sh to start the standalone server. Please note that it is a good idea to start the database server in a new terminal because the eXist server will print out a huge amount of information and mix up with the output of the AgentTeamwork. For example:

   ```
   [uwagent@medusa bin]$ pwd
   /home/uwagent/eXist/bin
   [uwagent@medusa bin]$ server.sh &
   ```

2. To Shutdown the eXist Server

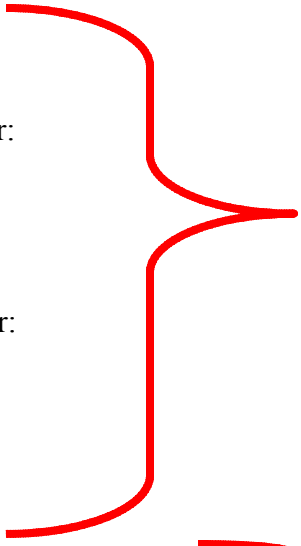
   Go to the following directory "/home/uwagent/eXist/bin" and use the shell script shutdownserver.sh to shutdown the standalone server. For example:

   ```
   [uwagent@medusa bin]$ pwd
   /home/uwagent/eXist/bin
   [uwagent@medusa bin]$ shutdownserver.sh
   ```

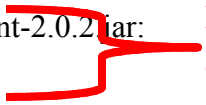
D. Setting Class Path

This Java class path is needed to be set correctly to prevent any compile time and run-time errors. Below is the example class path to compile and run AgentTeamwork. Please note that this example class path only valid for a particular machine and file directory. Use it as a guideline when switch to different machine or file directory.

/home/uwagent/mpiJava/lib/classes:
/home/uwagent/MA/benchmark/MPJv1.0:
/home/uwagent/eXist/exist.jar:
/home/uwagent/eXist/lib/core/xmldb.jar:
/home/uwagent/eXist/lib/core/resolver-20030708.jar:
/home/uwagent/eXist/lib/core/jakarta-oro-2.0.6.jar:
/home/uwagent/eXist/lib/core/antlr.jar:
/home/uwagent/eXist/lib/core/xmlrpc-1.2.jar:
/home/uwagent/eXist/lib/core/commons-pool-1.1.jar:
/home/uwagent/eXist/lib/endorsed/xerces-2.6.1.jar:
/home/uwagent/eXist/lib/endorsed/xalan-2.5.2.jar:
/home/uwagent/eXist/lib/endorsed/xml-apis.jar:
/home/uwagent/eXist/lib/core/log4j.jar:



/home/uwagent/eXist/commons-httpclient-2.0.2/commons-httpclient-2.0.2.jar:
/home/uwagent/eXist/commons-net-1.3.0/commons-net-1.3.0.jar:



/home/uwagent/enoch/UWAgent/UWAgent.jar:
/home/uwagent/enoch/GridTcp/GridTcp.jar:
/home/uwagent/enoch/agents:
/home/uwagent/enoch/UWAgent:
/home/uwagent/eXist:
/home/uwagent

Please note that some JAR file names might be different for a newer distribution.

E. Steps to compile AgentTeamwork

1. compile the java files inside UWAgent directory

   Go to the following directory "/home/uwagent/enoch/UWAgent" and use shell script "compile" to compile all java files. The shell script includes all necessary class paths. Here is the example:

   ```
   [uwagent@medusa UWAgent]$ pwd
   /home/uwagent/enoch/UWAgent
   [uwagent@medusa UWAgent]$ compile
   ```

2. Then, compile the java files inside agents directory

   Go to the following directory "/home/uwagent/enoch/agents" and use shell script

"compile" to compile all java files. The shell script includes all necessary class paths. Here is the example:

```
[uwagent@medusa agents]$ pwd
/home/uwagent/enoch/agents
[uwagent@medusa agents]$ compile
```

F. Starting the RMI Server and UWPlace

Go to directory "/home/uwagent/enoch/UWAgent" and use the shell script "run.sh" to start the RMI Server and UWPlace. Simply just type in the shell script and the port number. For example, I want to start the RMI Server with port 20000:

```
[uwagent@medusa UWAgent]$ pwd
/home/uwagent/enoch/UWAgent
[uwagent@medusa UWAgent]$ run.sh 20000&
```

G. Injecting the CommanderAgent with user application

1. To run test with Series user application

   Use the shell script runtestnow.sh to injecting the CommanderAgent with Series user application. Just type in the script name like below:

   ```
   [uwagent@medusa UWAgent]$ pwd
   /home/uwagent/enoch/UWAgent
   [uwagent@medusa UWAgent]$ runtestnow.sh
   ```

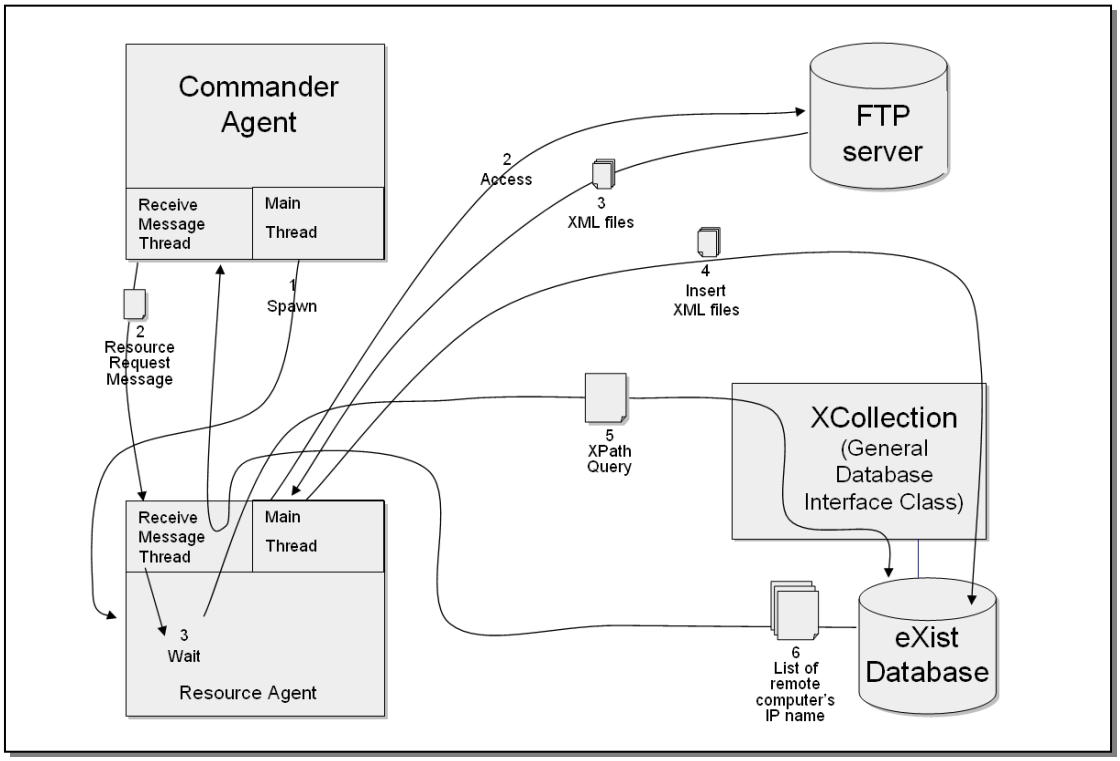
2. To run test with RayTracer user application

   Use the shell script runtestnow.sh to injecting the CommanderAgent with Series user application. Just type in the script name like below:

   ```
   [uwagent@medusa UWAgent]$ pwd
   /home/uwagent/enoch/UWAgent
   [uwagent@medusa UWAgent]$ RTruntestnow.sh
   ```

VI. Round-Trip Communication

This section briefly describes the round-trip communication between commander agent and resource agent.



After a job is submitted with a Commander agent, this commander agent will spawn a Resource Agent immediately. During the startup of resource agent, it accesses to the remote FTP server and download the new or updated XML files and then insert XML files into the local eXist database through the XCollection interface class. At the same moment, the commander agent will send a Resource Request Message, which consists of the resource requirement of the user application, to the resource agent to ask for a query. Since the resource agent is still in its startup process and is not ready to perform query, the receive message thread of the resource agent will wait until the resource agent is ready for query. When the resource agent is ready for query, its main thread will notify the Receive Message Thread to wake up and query the local database by sending a XPath query. The eXist database will generate a list of remote computers' IP name and send back to the Resource agent. The list will further forward to the commander agent. This is a complete round-trip communication between commander and resource agent.

VII. Resource Agent Startup

A. Two different modes

Resource agent can run in two completely different modes which are regular mode and remote probing mode. In the regular mode, resource agent download XML files from FTP server, update local database, accept resource request, query local database, return list of computer IP name and spawn child resource agent to perform remote probing. In remote probing mode, it is a child resource agent which is spawned by another parent resource agent in regular mode. It immediately migrates to remote computer and performs a series of testing and sends the status result of the remote computer back to the parent resource agent.

The arguments received by the constructor of the resource agent determine which mode a resource agent should run. If the constructor received an argument with exactly 5 pieces of information, which are ftp server name, ftp user name, ftp user password, frequency of periodic probing, and directory of the eXist database, then the resource agent will run in regular mode. On the other hand, if the constructor received an argument with only two pieces of information, which are a string "PerformTest", and a remote computer IP name, then the resource agent will run in remote probing mode.

B. Startup in regular mode

1. Connect to remote FTP server and calculate the time offset

   Since the remote FTP sever(ftp://ftp.tripod.com) locate in east coast with a time zone 3 hours faster than the local computer and the FTP client (apache FTP client) can not recognize the time zone different, resource agent have to calculate the time offset between the FTP server and the local computer. With the time offset calculation algorithm, resource agent is able to work with any FTP server located anywhere in the world.
2. Download the FTP file listing

   Get the file listing which contains the timestamp of each XML file from the FTP.
3. Get the file listing from the local database

   Get the file listing which contains the timestamp of each XML file from the local database.
4. Download new and updated XML file from the FTP

   Download any new XML files from the FTP server. Compare the timestamp of each XML file between the FTP server and the local database, download the updated file from FTP if necessary.

5. Ready for query
   Ready to perform query request from Commander agent.

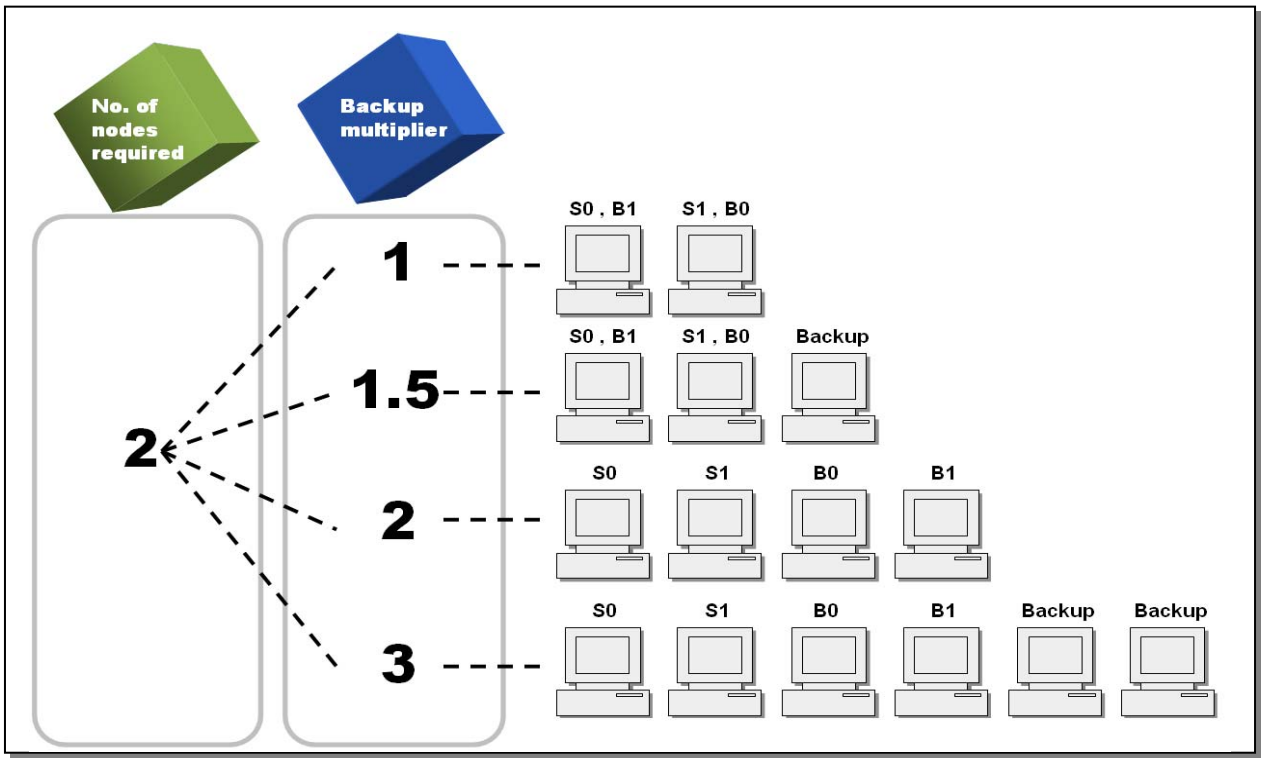
C. Startup in remote probing mode

1. migrate to the remote computer node
2. Find out the OS type of the computer node
3. Find out the available memory size in MB
4. Find out the percentage of CPU idle
5. Find out the available disk space in MB
6. Find out the bandwidth between this remote computer node with the parent resource agent
7. Send all the result back to the parent resource agent

For more detail information about the remote probing, please read section X of this document.

VIII. Resource Distribution

This section briefly describes how the commander agent distribution the remote computing resource to sentinel and bookkeeper agent.



When the Commander agent was started by the user, two pieces of information pass to its constructor. Those two pieces of information are number of computer nodes required and number of backup multiplier. If the number of nodes required is 2, which mean 2 pair of sentinel and bookkeeper agents (totally 4 agents S0, S1, B0, B1) will be spawned to the remote computing nodes. For example, if the backup multiplier is 1, then the resource agent will return 2 remote computing node IP name to the commander agent. In this case, Sentinel agent S0 share a computing node with Bookkeeper agent B1 and Sentinel agent S1 share a computing node with Bookkeeper agent B0. For example, if the backup multiplier is 3, then the resource agent will return 6 remote computing node IP name to the commander agent. In this situation, each sentinel agent and bookkeeper use a single computing node. The reminding two computing nodes are used for backup purpose in case of any computing node suddenly offline.

Please note that the commander agent only accept 1, 1.5, 2, 3 or any integer number bigger than 3 as backup multiplier. If the multiplier is not an acceptable number, the default value 1 will be used. Also note that all excessive computing nodes will be used as backup purpose.

IX. Arguments for Commander Agent Constructor

Below is the content of the shell script "runtestnow.sh". This section briefly describes how to pass the arguments for commander agent by configuring the shell script.

```
java -classpath
/home/uwagent/mpiJava/lib/classes:/home/uwagent/MA/benchmark
/MPJv1.0:/home/uwagent/eXist/exist.jar:/home/uwagent/eXist/l
ib/core/xmldb.jar:/home/uwagent/eXist/lib/core/resolver-2003
0708.jar:/home/uwagent/eXist/lib/core/jakarta-oro-2.0.6.jar:
/home/uwagent/eXist/lib/core/antlr.jar:/home/uwagent/eXist/l
ib/core/xmlrpc-1.2.jar:/home/uwagent/eXist/lib/core/commons-
pool-1.1.jar:/home/uwagent/eXist/lib/endorsed/xerces-2.6.1.j
ar:/home/uwagent/eXist/lib/endorsed/xalan-2.5.2.jar:/home/uw
agent/eXist/lib/endorsed/xml-apis.jar:/home/uwagent/eXist/li
b/core/log4j.jar:/home/uwagent/eXist/commons-httpclient-2.0.
2/commons-httpclient-2.0.2.jar:/home/uwagent/eXist/commons-n
et-1.3.0/commons-net-1.3.0.jar:/home/uwagent/enoch/UWAgent/U
WAgent.jar:/home/uwagent/enoch/GridTcp/GridTcp.jar:/home/uwa
gent/enoch/agents:/home/uwagent/enoch/UWAgent:/home/uwagent/
eXist:/home/uwagent UWInject localhost CommanderAgent -u
"/home/uwagent/enoch/agents" -p UWPlace -n 20000 -c myClient -s
ResourceAgent\$RemoteRscProbeTask,SentinelAgent,BookkeeperAg
ent -m 100 -j agents.jar,Series.jar
RQ_cpuspeed_456_cpucount_1_memory_2_os_linux_disk_5_total_2_
time_110000_cpuarch_i386_cpuload_40_bandwidth_20 RN_3
RA_ftp.tripod.com_agentTeamWork_test_5_/home/uwagent/eXist
RC_ResourceAgent\$RemoteRscProbeTask U_Series_2_a_3 C_Timer
```

A. The text highlighted in red

It is the class path.

B. The text highlighted in bright green

It is the arguments for UWInject. For more information about UWInject, please read Professor Fukuda's UWAgent User's Manual version 1.01 http://depts.washington.edu/dslab/AgentTeamwork/doc/uwagent.pdf .

C. The text highlighted in blue

It is the resource query requirement of the user application. It starts with RQ_ for commander agent to identify. It will pass to resource agent to generate an xPath XML query to query the local database to find the best fit remote computing nodes for the user application. The

follow resource request options are permissible in the commander agent:

-ip	Directly specify where a user wants to run his/her application.
-cpu_speed	Specify the cpu speed in MHz.
-cpu_arch	Specify the cpu architecture.
-cpu_count	Specify how many cpus a user program needs.
-memory	Specify how many MB a user program needs for memory.
-os	Specify under which OS a user program must be executed.
-disk	Specify how many MB a user program needs for temporary disk storage.
-total	Specify how many candidates the commander wants in an itinerary.
-cpuload	Specify how many percentage of cpu idle a user program needs
-bandwidth	Specify the MB bandwidth a user program needs
-time	Specify at which time a user program will be executed. Time should be given in the form of: 205534 (8:55pm 34second) or 0 (now).

D. The text highlighted in pink

This is the backup multiplier. It starts with RN_ for commander agent to identify. Commander agent only accept the number of the option to be 1, 1.5, 2, 3, or any integer number bigger than 3. Otherwise, default value 1 will be used. This option helps the commander agent to determine how to distribute the remote computing resource to sentinel and bookkeeper agent. And, it also helps the resource agent to determine how many remote computing node IP name should return to commander agent. For more information, please read section VIII of this document.

E. The text highlighted in orange

This is the argument for the resource agent constructor. It starts with RA_ for commander agent to identify. This argument will pass to the resource agent by commander agent. Below is the meaning of each argument.

ftp.tripod.com	Specify the address of the FTP server
agentTeamWork	Specify the user account of the FTP server
test	Specify the password
5	Specify the probe frequency in minutes
/home/uwagent/eXist	Specify the where the eXist database was installed

F. The text highlighted in sky blue

This is the list of sub class for the resource agent. It starts with RC_ for commander agent to identify .Currently, resource agent only have one sub class "ResourceAgent$RemoteRscProbeTask". Please note that in Linux, "$" sign need to add a "\" in front of it. Therefore, sub class "ResourceAgent$RemoteRscProbeTask" becomes "ResourceAgent\$RemoteRscProbeTask".

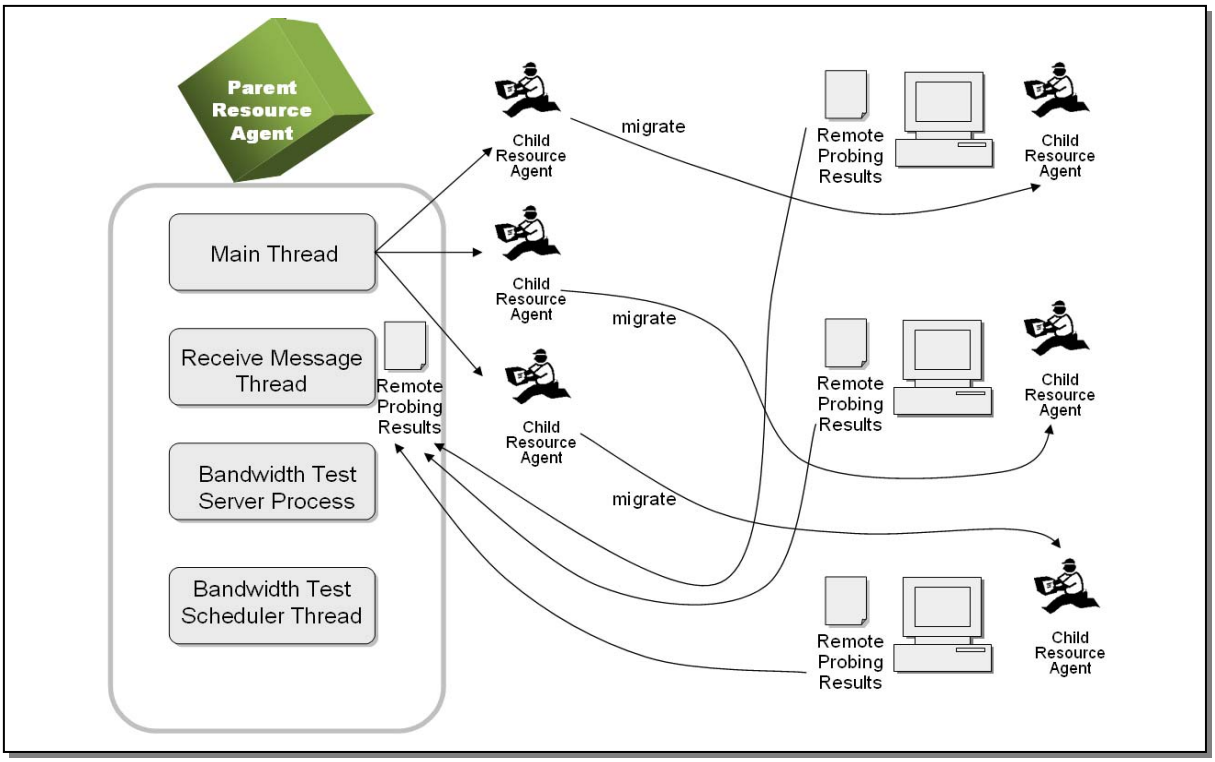
G. The text highlighted in violet

This is the user program's class name and the following are the arguments for its constructor. It starts with U_ for commander agent to identify

H. The text highlighted in brown

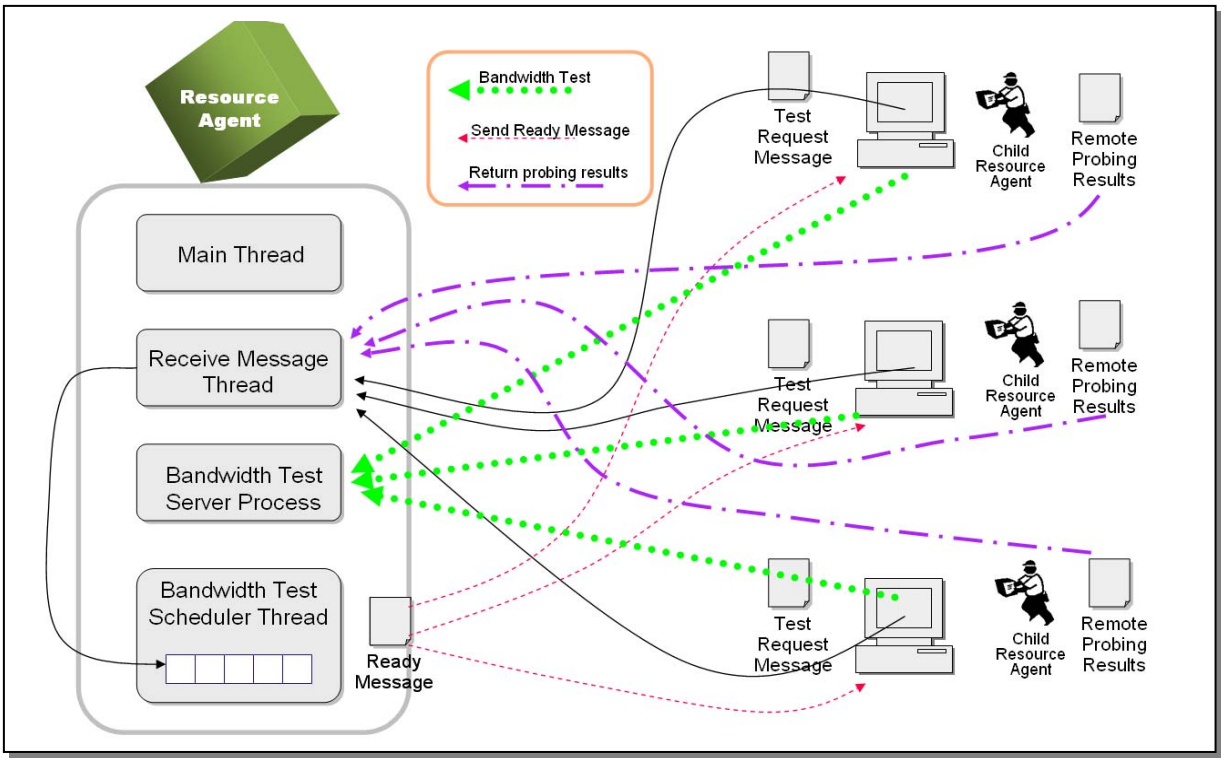
This is the class name which is needed by the user program. It starts with C_ for commander agent to identify

X. Probing Remote Computing Node



A. Child Resource Agent

Above is the overview diagram of how resource agent performs periodic remote probing. When the resource agent performs remote probing, it spawns child resource agents which running in remote probing mode (see section VII A for more information). If the parent resource agent needs to probe 3 remote computers, then 3 child resource agent will be spawned. The child resource agent will migrate automatically to the corresponding remote computing node. A series of test, including OS type, memory size, CPU idle, disk space, and bandwidth, will perform in the remote computers. Then, the result will send back to the parent resource agent.



B. Bandwidth Test

The bandwidth test need to specially handle because inaccurate result will be generated when all child resource agent attempt to connect to parent resource agent to perform bandwidth test at the same time. Thus, create a bottleneck to the network. In order to get an accurate bandwidth test result, only one child resource agent can perform bandwidth test at the same time. A Bandwidth Test scheduler Thread was added to the parent Resource Agent. Before the child resource agent performs the bandwidth test, it has to send a Test Request Message to the Parent resource agent. The parent resource agent will store all the Test Request Messages into a queue. Then, the Bandwidth Test Scheduler Thread will dequeue the first Request Message from the queue and send a Ready Message to the corresponding child resource agent. The Bandwidth Test Scheduler Thread will wait and not send out any Ready Message until the corresponding child send in the Remote Probing Results. In order to prevent deadlock, the Bandwidth Test Scheduler Thread will send out another Ready Message if a child cannot return the Remote Probing Result within the timeout period.

XI. Appendix

The following is a list of resources that are useful for the development of resource agent.

1. eXist Home Page
   http://exist.sourceforge.net/

2. eXist API Specification
   http://exist.sourceforge.net/api/

3. xPath tutorial
   http://www.w3schools.com/xpath/default.asp

4. xUpdate tutorial
   http://www.xmldatabases.org/projects/XUpdate-UseCases/
   http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html