Grapevine User's Manual

Kunal Agrawal and Kevin Chu

May 20, 2002

1 Introduction

In general a grapevine application is a Job, which may contain many smaller tasks. The application developer creates a job and defines all the different tasks within it. The client then sends this job to the grapevine server. The grapevine server then sends all the tasks to different volunteer nodes. The volunteer nodes run these tasks and sends the results back to the server. The server then sends the results back to the client. After the client gets back all the results, the client removes the job from the server.

2 Installation

The developer needs to include the java archive grapevine.jar in the class path. The developer can then start writing the applications.

3 Writing Applications

The applications developer is provided with two templates for writing his applications. One is JobLauncher.java and the other is TaskImp.java.

For every Job the Client has to fill in the template of one JobLauncher. The template is given below.

```
import java.rmi.*;
import java.util.*;
import grapevine.common.*;
import grapevine.client.*;
import grapevine.server.*;
```

```
public class JobLauncher
{
  public static void main(String args[]) {

    // Read in command-line arguments.
    String serverHost = args[0];

    // Set up the RMI SecurityManager.
    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new RMISecurityManager());
    }

    // Create the job definition.
    JobDefinition jobDef = new JobDefinition();

    // Set the local data and executable directories for the job.
    jobDef.setDataDirectory("data");
    jobDef.setExecDirectory("exec/");

    // Add the data elements required for the job.
    DataHandle dh1 = jobDef.addDataElement("Test.data");
    DataHandle dh2 = jobDef.addDataElement("Test2.data");
    DataHandle dh3 = jobDef.addDataElement("Test3.data");
    DataHandle dh4 = jobDef.addDataElement("Test4.data");
    DataHandle dh5 = jobDef.addDataElement("Test5.data");

    // Add executable elements required for the job.
    ExecHandle eh = jobDef.addExecElement("HelloWorld.class");


    // Define Task1
    TaskDefinition taskDef1 = new TaskDefinition("HelloWorld");
    ArrayList data = new ArrayList();
    data.add(dh1);
    data.add(dh2);
    data.add(dh3);
    taskDef1.setMainExecElem(eh);
    taskDef1.addDataElem(data);
```

```
// Define Task2
TaskDefinition taskDef2 = new TaskDefinition("HelloWorld");
taskDef2.setMainExecElem(eh);
taskDef2.addDataElem(dh4);
taskDef2.addDataElem(dh5);

// Add TaskDefinitions to JobDefinition
jobDef.addTask(taskDef1);
jobDef.addTask(taskDef2);

// Print the job and task definitions for inspection
System.out.println(jobDef.toString());
System.out.println(taskDef1.toString());
System.out.println(taskDef2.toString());


// Start the Grapevine system
Grapevine grapevine = null;
try {
  grapevine = new Grapevine(serverHost);
} catch (Exception e) {
  System.err.println("Error creating Grapevine. Exiting...");
  System.exit(1);
}

// Submit the job
Job newJob = null;
try {
  newJob = grapevine.submitJob(jobDef);
  System.out.println("Job ID = " + newJob);
  System.out.println("Job Status = " + newJob.getJobStatus());
  System.out.println("Job Progress = " + newJob.getProgress());
  if (newJob.isDone()) {
    System.out.println("Job is done.");
  } else {
    System.out.println("Job is not done.");
  }
} catch (Exception e) {
  System.err.println("JobLauncher exception: Failed to create new job - "
    + e.getMessage());
```

```
      e.printStackTrace();
    }

    // Start the job
    if (newJob != null) {
      try {
        newJob.start();
      } catch (Exception e) {
        System.err.println("JobLauncher exception: Failed to start job - "
          + e.getMessage());
        e.printStackTrace();
      }
    }

    // Remove the job
    if (newJob != null) {
      try {
        grapevine.removeJob(newJob.getJobID());
      } catch (Exception e) {
        System.err.println("JobLauncher exception: Failed to remove job - "
          + e.getMessage());
        e.printStackTrace();
      }
    }

  }

}
```

4 Status

For every job, the client can check the status of the job using the methods provided in the job class. The isDone method returns a true if all the tasks of the job are done and results are ready. The client can also check the status of the job by using getProgress for the remote job method. This tells us how many final results are available on the server.

5 Results

The results for all running jobs are stored on the server in the form of hashtables. The client can access these using the methods provided. The results can be recovered at any time and not only after the job is complete. The server will return the results that are currently available. There is also a facility for getting intermediate results. It is similar to the final result reporting.

6 Task

The main task class is be defined in a class that extends the Task.java abstract base class.

The template for the main task class is given below

```
import java.util.*;
import grapevine.common.*;

public class <class name> extends Task
{
  static public void main(String args[])
  {
    <class name> hw = new <class name>();
        Thread thread = new Thread(hw);
        thread.run();
  }

  public <class name>()
  {

  }

  public void run()
  {
    /* add all your code here */
  }

  public void stop(){
    /*Add stopping code here */
  }
```

```
}
```

There are some functions provided in the abstract base class(Task) which can be used for communication with the server and with the other tasks. These include getTaskID, Send, Receive, SendResults, SendInterResults etc. The details can be seen in The JavaDOC for the TaskClass.