GPE Tutorial - Hands-On Session

Using the Grid Programming Environment

Ralf Ratering, Intel Corporation, ralf.ratering@intel.com

September 6, 2007

The only prerequisite: The free VMware Player

For the hands-on session we will use a pre-configured VM image that contains a full GPE installation including a UNICORE server. Therefore attendees have to install the VMware Player on their systems. The VMware Player is freely available from http://www.vmware.com/products/player. There are versions for Windows and Linux. The size of the download package is ~145MB, so please make sure to start the download in advance if possible.

If you have installed the VMWare Player you're all set for the tutorial. Everything else will be pre-installed in the VM image you will receive during the tutorial.

Getting Started

Copying the VM image from the Tutorial DVD

Once the VMware Player is installed, you will have to copy the VM image for the tutorial onto your system. The image will be distributed by DVD (or USB stick) during the tutorial. Its size is ~4GB, so please make sure there is enough space available on your file system. On the DVD you will find a file "Tutorial-VM.rar". Copy this file to your hard disk and unpack it. In case you do not have a packaging tool available on your system that can deal with the "rar" format, you will find free tools in the "Windows" or "Linux" sub directories of the DVD.

The archive will contain 3 files, that are needed by the VMware Player (SUSE Linux.vmdk, SUSE Linux.vmx, SUSE Linux-flat.vmdk). After you have copied the image to your file system you can open the file "SUSE Linux.vmx" in the VM Player and the tutorial OS will be started. We will use an OpenSUSE image that already contains all applications that are required for the tutorial.

At startup the VMWare Player will inform you that the VM image location has changed, since the last time it was used. Please confirm in the dialog that you want to create a new VM identifier.

The login name on the system is "unicore", password: "tutorial", root password: "gpedemo". On startup the system will automatically log in as user "unicore".

Starting the UNICORE 6 server

Once you have logged into the system, you will find the UNICORE server in the directory /home/unicore/Unicore6. The server will be started with the script "start.sh" in that directory. If the start was successful you will see the following message at the end of the console:

```
[…]
Starting UNICORE/X server…
Unicore atomic services starting.
```

Starting the GPE Application Client

The GPE client framework is installed in the directory "/home/unicore/GPE4Unicore Client". The Application Client will be started with the script "bin/application-client.sh".

After the client started you will be asked for the password to open the keystore that contains your private key and trusted certificates. The password is "tutorial".

If the keystore has been successfully unlocked, you should see the entry "localhost" in the registry list. If you select the registry entry, two new entries should appear in the list of target systems below. That's it! You're ready to run your first Grid job on your Unicore demo server.

Section 1: Using the GPE Application Client

Running a simple Grid job

This is how to run your first Grid job using the Generic GridBean that is capable of running arbitrary applications:

1. Connect to a target system: In the list of target system you will have to select the entry "localhost" and connect to it with the corresponding entry in the popup menu that appears on a right mouse click. (Behind the scenes a "connect" will create a target system resource for you, that you can now work with.) If the connection has been established, the entry should appear as "DEMO-SITE".
2. Select the application "Date 1.0" application in the combo box of the Generic GridBean.
3. Make sure that the "DEMO-SITE" target system is still selected in the target systems panel and submit the job with "File->Submit"
4. The job should appear in the job list. If it is still "RUNNING", do a right mouse click->"Refresh" to update the status.
5. Once the job appears as "SUCCESSFUL", do a right mouse click->"Fetch Outcome" on the selected job to retrieve standard out and standard error.

6. In the "Stdout" panel of the "Outcome" tab, the current date of the target system should be printed now, indicating that you just ran your first job on the Unicore demo server successfully!

Please note, that you just executed the abstract (pseudo-)application "Date 1.0" on the Grid. The application was incarnated to the command "/bin/date" on the Unicore server.

Using the POVRay GridBean

The next job we'll try is a little more complicated, since it involves file imports and exports between client and server. We'll also see how an application-specific GridBean works, taking the POVRay ray tracing application as an example.

1. Download the POVRay GridBean using "File->Download GridBean". The POVRay panel should appear in the Application Client now.
2. You will find an example POVRay scene description (Example.pov) in the folder "/home/unicore/GPE4Unicore Client/tutorial/files". Load this file into the POVRay text editor.
3. Submit the POVRay job to the target system. The POVRay scene file will now be transferred to the server and rendered there. This logic is implemented in the POVRay GridBean and hidden from the user.
4. If the job finishes as "SUCCESSFUL", fetch the outcome as in the previous examples. This time you will notice that the output image of POVRay is being displayed in the outcome panel in the client. This pre-processing functionality of displaying the image is also part of the POVRay GridBean, while stdout and stderr display are standard panels in the Application Client.

Solving Sudoku with Ruby

Now that we've seen how comfortable an application-specific GridBean like the POVRay GridBean handles all file imports and exports as well as the program parameters, we'll go back to a generic GridBean that allows executing arbitrary scripts: the Script GridBean. We will show how it can be used to solve a Sudoku game by running a Ruby program on a Grid system:

1. Load the Script GridBean in the Application Client
2. Select Ruby as application in the Script GridBean combo box
3. In the folder "tutorial/files", you will find the main Ruby file "main.rb". Load this file in the text editor of the Script GridBean. If you're roughly familiar with Ruby, (or programming such), you will notice that main.rb requires additional files to execute: The statement <requires "sudoku"> imports an additional Ruby file and the constructor <Sudoku.new("board1.txt") > takes a

file named "board1.txt" in the current directory as input. This means that we have to transfer the files to the job directory before we actually run "main.rb".

4. The files "sudoku.rb" and "board1.txt" are also located in the directory "tutorial/files". Add these files to the "InputFileset" in the "Files" tab.
5. Submit the job and fetch the outcome as usual. Were you able to solve board1? Can you do the same with "board2.txt" in the "tutorial/files" folder? What happens if the Sudoku has multiple solutions?

Exercise: Run a POVRay job using the Generic GridBean

Instead of using the comfortable POVRay GridBean, it is also possible to render the "Example.pov" file using the Generic GridBean. Therefore you have to specify Example.pov as input file in the "Files" tab and the image file "Example.png" as output file in the "Output files" section of the Generic GridBean. Please note that output files are defined by name/value. This allows to refer to files in workflows by their variable name. In our case it is save to set both name and value to "Example.png".

Can you make this work?

Section 2: GPE Programming

Introduction

In this section we will learn how to implement your own Grid client using the GPE API. You will learn how to run and monitor Grid jobs, how to specify input files and parameters and how to retrieve the outcome. We do not start with GridBean programming directly, but instead use a simple Java client for the programming exercises. We believe that it is important to understand the underlying GPE concepts by looking at the code that is actually hidden by the GridBean API.

The main idea behind the GPE API is that details of the underlying Grid middleware are hidden from the developer. Currently two implementations of the GPE API exist, one for UNICORE 6 and one for the Globus Toolkit 4. All code that is written in this section will run on both middleware implementations without modification.

Task 1: Running a Grid job from a simple Java application

For this exercise we will use the class com.intel.gpe.tutorial.SimpleClient1. You will find the corresponding source files in the "tutorial/src" folder of the GPE4Unicore Client installation directory.

To start the application you will use the ant build script "tutorial/build.xml". Try running the application by invoking the command "ant runSimpleClient1". If the job executed successfully, open your GPE Application Client and fetch the outcome of the job.

Exercise 1.1: Which Application was executed and what is the result? Does everyone receive the same result here and if yes, why?

Exercise 1.2: The job in the GPE Application Client appears without a name in the job list. Set the job name in the source code of SimpleClient1. Hint: A job name in GPE is an "Id".

Exercise 1.3: SimpleClient1 allows you to specify arbitrary command line parameters for the application that will be stored in the String applicationParameters. Set the application parameters as a field in the GPE job. What should be the name of this field and why? Hint: Lookup the server-side incarnation of the applications in the example IDB under "files/example.idb". What is the common parameter of all applications? Execute the COMMAND application with "date" as parameter to see if your program works.

Task 2: Adding file imports

For this exercise we will use SimpleClient2 ("ant runSimpleClient2"). This client provides a method importFilesFromLocal() to import files from the local file system to the job's working directory.

Exercise 2.1: Start the file imports by adding a line importFilesFromLocal(jobClient); in the code. Where should this line be added and why?

Exercise 2.2: Verify your implementation by importing a file from your local file system and executing the "ls" command on the job's working directory.

Task 3: Specifying and fetching the job outcome

For this exercise we will use SimpleClient3 ("ant runSimpleClient3"). This client provides a method fetchOutcome() for fetching standard output of your job to your local file system.

Exercise 3.1: Modify fetchOutcome() so that also the standard error will be fetched. Hint: In a good implementation, you should only send one request to the server to retrieve the complete outcome.

Exercise 3.2: Create a job that will fail to see if fetching the standard error works.

You have learned now how to add a file transfer for standard error from the job working directory to your local file system. The same mechanism will also work for arbitrary file exports. For instance you can add a transfer that exports the image file of a POVRay execution to your local file system. (If you execute the POVRay application on the input file "./files/example.pov" an output file named "example.png" will be created in the job's working directory.)

Exercise 3.3: Add a file transfer that exports the image file of a POVRay execution to your local file system. Set the application parameters +W and +H to change the image size.

Task 4: Destroying the job after execution

We almost have a fully working GPE client now. An important step that is still missing is the job destruction.

Exercise 4.1: After the job has successfully finished and all outcome has been fetched, destroy the job. Is there another possibility of removing a job than directly destroying it?

Task 5: Creating a multi-step job

We have now created a fully functional client that allows to stage in files from the local file system to the job's working directory, execute an application with parameters, retrieve the outcome and destroy the job. In this exercise will enhance this functionality to create a client-side workflow that consists of two jobs.

Exercise 5.1: Create a client-side workflow that imports the Java file "./files/Factorization.java". This Java class will compute the prime factors of a large numbers. Compile the java class on the remote target system using the JAVAC application. The resulting class file "Factorization.class" should be transferred back to the client file system and then back to the working directory of a second job. The second job should invoke the JAVA application on the Factorization class with the number as parameter. Finally the outcome of the execution will be displayed in the standard out of the second job.

Exercise 5.2: In the previous exercise we transferred the intermediate class file back to the client for simplicity. We can improve the performance if we leave the class file in a temporary directory on the remote target system. The second job should then read the file from this temporary directory instead of transferring it from the local file system. Hint: Instead of using the class LocalGPEFile the class com.intel.gpe.client2.common.utils.RemoteGPEFile should be used to specify file locations on remote file systems. What is the risk of file spooling in temporary directories if multiple users are mapped onto the same login at the target system?

Exercise 5.3: Instead of spooling the class file at a temporary directory on the target system, the second job can even directly read the file from the working directory of the first job. This is actually the preferable solution for multi-step jobs, because it prevents the performance and security issues of the implementations in 5.1 and 5.2.

Hint: If a job is destroyed, its working directory will also be removed.