Distributed Computing Using Java RMI (Remote Method Invocation)

Hongping Liang
hliang@curagen.com
Bioinformatics Department
CuraGen Incorporation
New Haven CT

S.-C. Chu
schu@runet.edu
Computer Science Department

Jurgen Gerlach
jgerlach@runet.edu
Department of Mathematics
Radford University
Radford, VA 24142

U.S.A

Abstract

This paper investigates another approach to realize distributed computing in a network environment using Java RMI (Remote Method Invocation). RMI is a collection of API (Application Programming Interface) developed by JavaSoft that allows objects to be used in a remote manner. Objects in this paper are computationally intensive tasks that can be distributed on different virtual machines to realize a reasonable speedup.
However, because of the security restrictions, an Applet running within a browser can only make a network connection to the host that it comes from. It can only talk directly to the server machine of its origin and cannot make a network connection to any other machines. Therefore, through the Applet, objects cannot be directly distributed to several machines at the same time, and parallel processing cannot be achieved. On the other hand, the Applet plays a very important role in Internet applications and users can access it from any machines of different platforms and from anywhere.
To get around this awkward situation, we develop a two tier scheme using RMI, which will allows the users to use browser as the interface and on the back-end, a parallel processing can be extensively utilized for mathematical research.

1. Introduction

The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (VM) to invoke methods of an object running in another Java VM. It is a collection of Application Programming Interface (API) developed by JavaSoft [3] that allows objects to be used in a remote manner. One of the most powerful RMI features is the Mobile Behavior, which allows Java applications to pass true objects (data and code) between virtual machines without having to distribute the supporting class files. In addition, the RMI can be used for one machine to talk to several machines. And the server of one RMI can be the client of another RMI at the same time or verse visa. Machines of different platforms can be

connected and can work together through several RMI's. Because of these features, computational intensive objects can be easily distributed among the machines and parallel processing can be implemented in extensively.
RMI can be used with an Applet as an interface to allow access through the web. Yet because of the security restrictions, an Applet running within a browser can only make a network connection to the host that it comes from. It can only talk directly to the server machine of its origin and cannot make a network connection to any other machines. Thus, we cannot talk directly to several machines through an Applet in order to achieve parallel processing. At the same time we need the Applet as the interface since it plays a very important role in Internet applications and users can access it from any machine of different platforms and from anywhere. In this paper, we develop a architecture using RMI, which will allow users to use a Web browser as the interface to initiate processing on different machines in a network in parallel.
This research is the continuation of our effort in reconstructing 3-D objects from digital holograms [1][2]. The approach discussed in this paper suggests a more generic yet powerful technique to allow parallel processing by distributing objects across the network to other machines in which the algorithm to accomplish the specific task does not exist. For the following discussion, we use a simplified example of distributing an image to some machines (of different platforms) along with the code for each machine to handle the image processing as directed. We envision the software architecture can support quite a number of diverse applications, from simple mathematics tutors to stock market investment watch. We will discuss the architecture of our proposed software system next.

2. Software System Architecture

2.1 Two-RMI Architecture

In order for an Applet to invoke methods on several machines directly, we design a two-RMI architecture (Figure 1). Two Java remote interfaces will be used in this structure. One of the RMI's is for the connection between the user interface to the machine where web server resides on. The web server and this RMI server is on the same machine. The client user interface will only need to talk the web server machine. Another RMI will be used to connect the web server machine to various machines used for computing. To connect these two RMI, we could make the server of one RMI as the client of the compute server at the same time. In this way, the Applet in the user interface can talk to the various computing machines indirectly through the web server machine. The first RMI will function as the distributor and the second RMI actually to process the computing task. The server for the first RMI will be the distributor and at the same time, it is the client of the compute server.
Through this design scheme, at the front-end, user is not aware of any difference whether he is running on a single processor or on several different machines. They can run the Applet anywhere and on any machine as long as the browser supports the RMI. At the back-end, an extensible and flexible distributed framework can be achieved easily. Based on the computational task, various computing machines can be invoked and an additional compute server can be added without changing of any code. The compute server can be changed very easily to handle various tasks, e.g., connect to database through Java Database Connectivity (JDBC), or access programs written in other languages through Java Native Method Interface (JNI).

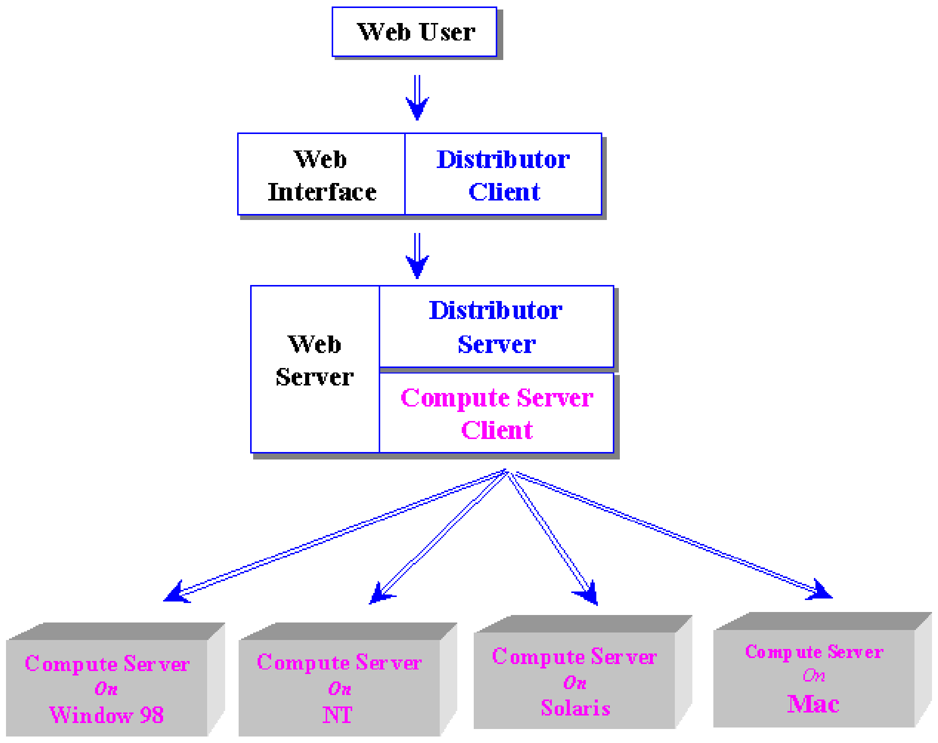


Figure 1. Two Tier RMI System Architecture

2.2 Design of Computing Tasks

In this framework, the computing task will be implemented as a behavior, which will be passed from a compute client (or client machine) to the compute server at run time. In this way, the compute server is not aware of what task it will process at first and can do any arbitrary task at run time without changing of the any code. The client machine will define the behavior. For different computing tasks, the only code need to be changed is the client's task class implementation and everything else remains same. The compute server will dynamically load the behavior from the client machine during run time. Due to this feature of the mobile behavior of RMI, the whole structure is extensible.

2.3 Design of Remote Interface

Java programming language provides an approach as a remote interface in order for two (different) machines to talk to each other. A remote interface is predefined and known to both machines, and it includes each of the methods that will be called remotely. This is a way to tell the RMI client that there are some functional methods on a network machine for the client to use, and to tell the client what are the parameters and return values for these methods. The client machine has no idea how these methods are implemented. As long as the client call these methods and provide with the parameters, these methods will function. The RMI server will has the actual implementation of these methods and is always running and waiting for the call of RMI client.
Two RMI's are used in our design and the first remote interface provides the RMI client with ways to connect to the distributor server, connect to the compute servers through the

distributor, send the task to the compute servers, process the task on the compute servers, and retrieve the file information on the RMI distributor server.
As contrast to the first remote interface, this remote interface only has three methods. The first one is used to check the connection and the second one is for sending the task. And finally, the last one will be used to start the actual computing and the return value is the task object that has been processed.

2.4 User Interface

For the front-end user, there should be ways for them to upload image file to the server, open the existing image file, set up the parameters for computing and choose the machines to run the process. There still should be a way for users to view the images that were processed on various machines.

2.5 Selection of Machine Type and Operating System

The following common operating systems are chosen to work as the compute server: Window 98, NT, and Solaris (Unix).

3. Implementation

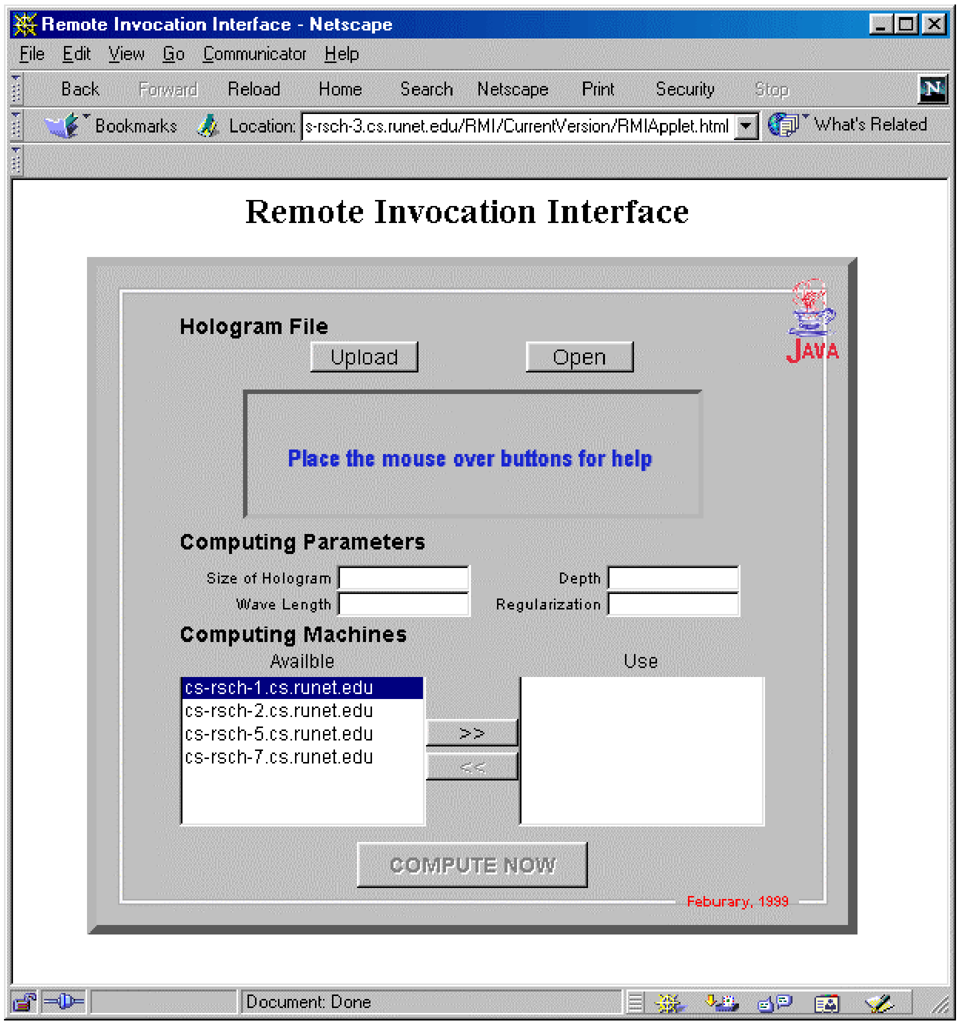
The whole design was implemented in the four packages, according to the functionality. The Client package provides the user interface using Applet and the RMI Client implementation. The Task package defines the task to be processed and task behavior will be defined in this package so that the compute server can dynamically load the behavior from this package. The Distributor package is responsible to serve the RMI user interface client, connect to the compute server, and send and forward back the task. The Compute Server package is actually for the computing and it will be running and waiting for the connection from the distributor server.

3.1 Task Package

The Task package defines the task that will be distributed on various computers for parallel processing. In consideration of the flexibility and extensibility of the program, the task is designed as an abstract interface and the actual class for the implementation. In this way, the program can be easily modified to handle the various tasks. For a different task, the implementation of the task is the only class need to be changed, and everything else including distributor server, compute server can still remain the same. This kind of the technique is called behavior-based application. In this case, the behavior is the task, which is unknown at the design time and can be defined later. Once the task is defined, the behavior defined by the task will be passed to the compute server. And the compute server will take this behavior and process as the behavior instructs. Therefore, the compute server is able to work on any arbitrary task at run time and the client machine will tell the server what to do.

3.2 Client Package

The Client package includes many classes for the user interface, event handling, and communication between the client machine and the distributor. The major user interface is the RMIApplet class, which will use the HelpWindow, OpenImageFile, StatusWindow, StatusPanelThread, and ImageDisplayWindow classes. And it display a user interface as follows (Figure 2):



s-rsch-3.cs.runet.edu/RMI/CurrentVersion/RMIApplet.html

Place the mouse over buttons for help

cs-rsch-1.cs.runet.edu
cs-rsch-2.cs.runet.edu
cs-rsch-5.cs.runet.edu
cs-rsch-7.cs.runet.edu

Figure 2. Client User Interface

Through this interface, the user can upload a hologram image file from the client machine to the distributor server. Because the applet is not allowed to access the client local file system due to the security reason. The file uploading will bring up a separate window and is handled outside the applet. On the server side, the uploaded file is processed by the UploadFile class, which is a Servlet class and will accept the uploaded image file and write the directory for the image file on the server. Once the image file is uploaded on the server, user can choose the available image file on the server for further process. This process is handle by bringing up a new window (Figure 3) in order for user to browse and select an image file for distributed computing.

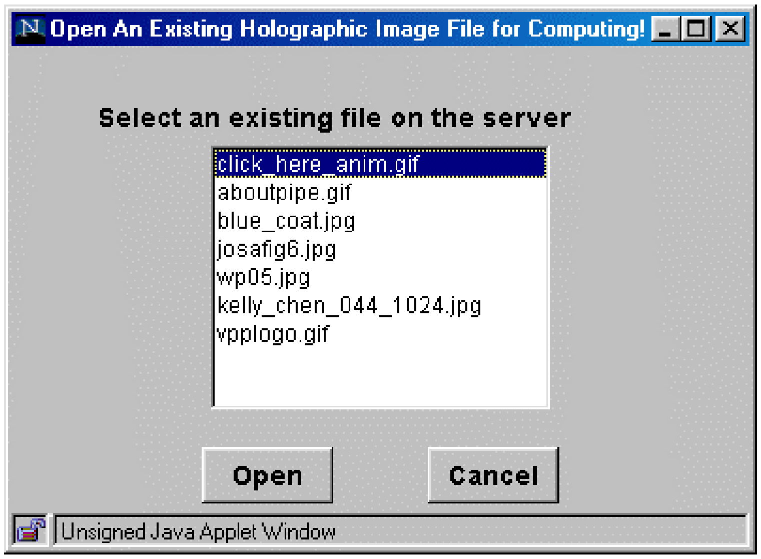

Open An Existing Holographic Image File for Computing!
Select an existing file on the server
click_here_anim.gif
aboutpipe.gif
blue_coat.jpg
josafig6.jpg
wp05.jpg
kelly_chen_044_1024.jpg
vpplogo.gif
Open
Cancel
Unsigned Java Applet Window

Figure 3. Open Holographic File Window

After the user selects an image file, a thumbnail of the image will be displayed to inform the user. To view a full size image, user can click on the thumbnail and a separate window will pop up and show the full size image (Figure 4). At this point, the image is ready for distribution and computation.

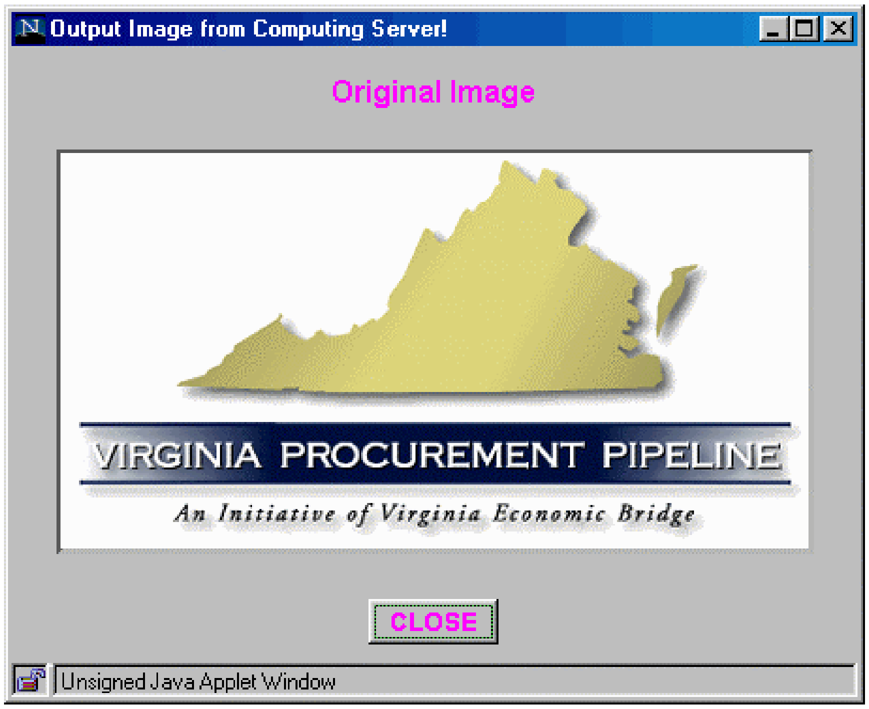
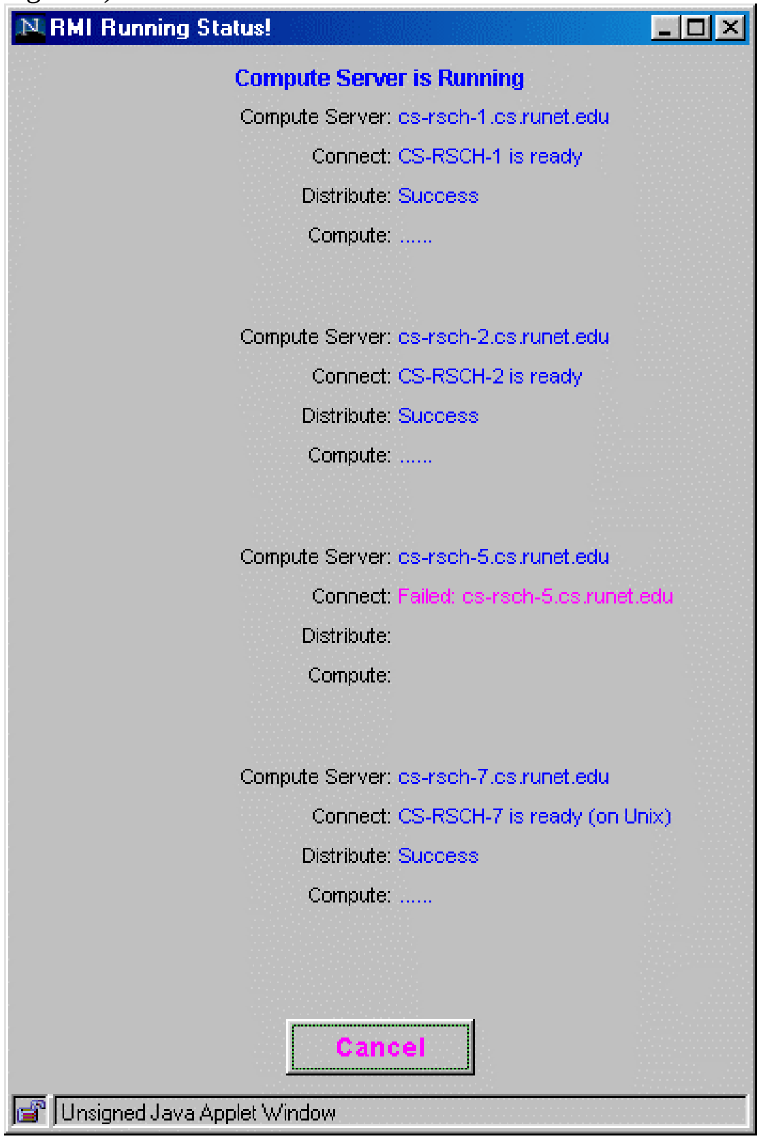

Output Image from Computing Server!
Original Image
VIRGINIA PROCUREMENT PIPELINE
An Initiative of Virginia Economic Bridge
CLOSE
Unsigned Java Applet Window

Figure 4. Image Display Window

Through the main user interface, user can set up the parameters for computation such as the size of the object, the wave length used to scan the image, the depth at which the reconstruction will be made, and the regularization parameters. In addition, there is a panel on the main interface for user to select the machines to process the image (Figure 2).
The "Compute Now" button will be enabled once the image is ready, parameters are set, and machines for parallel processing are chosen. This button will initialize a series of processes

and do the actual parallel computing. First, it connects to the computing machines chosen by the users. And then, it will send the image data to these machines through the distributor. Finally it will ask the computing machines to process the image. To inform the user what happens in these processes, a status window will pop up to show the status of each step for each machine (Figure 5).


RMI Running Status!

Compute Server is Running

Compute Server: cs-rsch-1.cs.runet.edu
Connect: CS-RSCH-1 is ready
Distribute: Success
Compute: ......

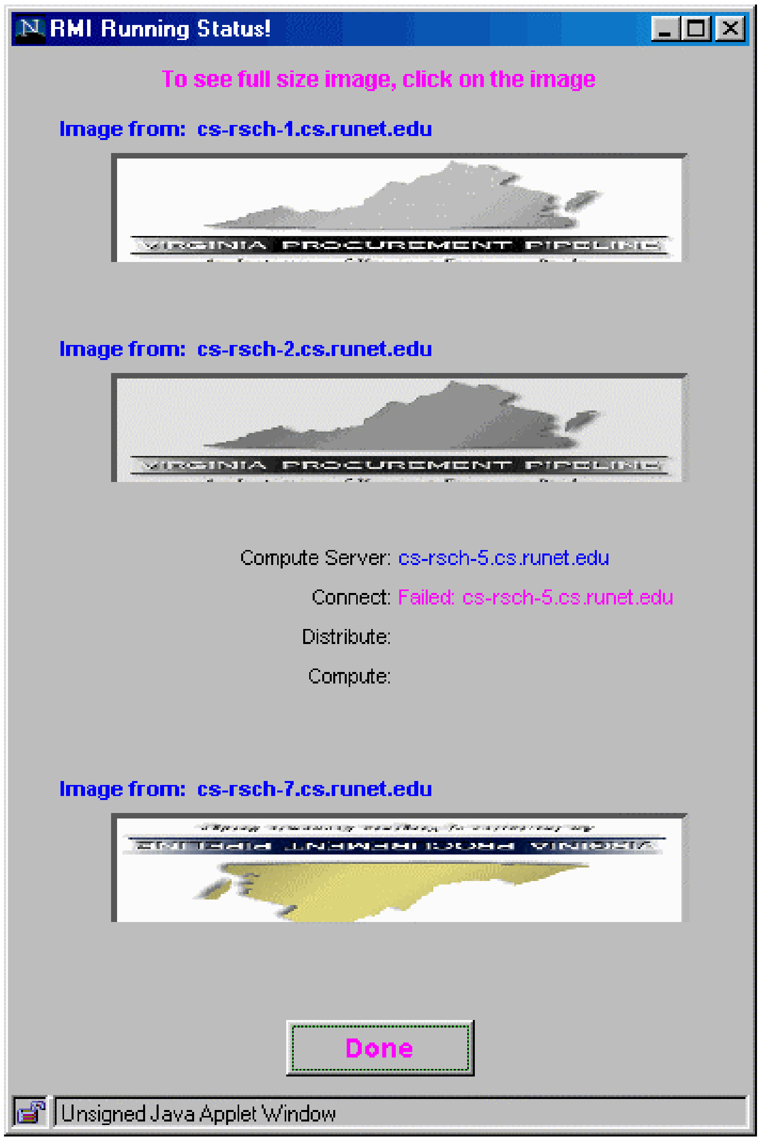
Compute Server: cs-rsch-2.cs.runet.edu
Connect: CS-RSCH-2 is ready
Distribute: Success
Compute: ......

Compute Server: cs-rsch-5.cs.runet.edu
Connect: Failed: cs-rsch-5.cs.runet.edu
Distribute:
Compute:

Compute Server: cs-rsch-7.cs.runet.edu
Connect: CS-RSCH-7 is ready (on Unix)
Distribute: Success
Compute: ......

Figure 5. Status Window
(to show the status of connection to compute machines, distribution of images, and the process for each machine)

To ensure the parallel process, for each machine chosen by user, a separate thread will be created. In this way, all processors will process the image simultaneously. At the end of the process, the status window will display the thumbnails of each individual processed images (See Figure 6).


RMI Running Status!

To see full size image, click on the image

Image from: cs-rsch-1.cs.runet.edu

Image from: cs-rsch-2.cs.runet.edu

Compute Server: cs-rsch-5.cs.runet.edu

Connect: Failed: cs-rsch-5.cs.runet.edu

Distribute:

Compute:

Image from: cs-rsch-7.cs.runet.edu

Figure 6. Images after Processing

3.3 DistributorServer Package

The DistributorServer package works as a middle tier between the client Applet and the compute server. Its major responsibility is to connect the Applet and the compute server and pass the information back and forward between them.

It consists of two classes, Distributor and DistributorServer. The Distributor class is a remote abstract interface, which is used by the client Applet to communicate with distributor. The DistributorServer class is the actual implementation of the Distributor interface. It works as the server as the Applet client, and at the same time, it will function as the client of the compute RMI server. The major functionality includes letting users to get the list of the uploaded image files on the server, to send the request of the Applet client to the compute server and to pass the result back from the compute server to the Applet client.

3.4 ComputeServer Package

The ComputeServer package will actually do the computational work. It will take the image sent by the distributor, process it, and send back to the distributor. There is a RMI to connect the distributor and the compute server. The remote interface is the Compute class and the class ComputeServer is the implementation of the remote interface. The image process on the ComputeServer is implemented as behavior-based model as follow.
To the compute server, it will dynamically load the implementation of the task class and get the behavior of the task. And then, it simply calls the process method passed by the task object.

3.5 UploadImageFile Package

This package allows user to uploaded image file from client Applet to the server. It implements as a Servlet class to get the image passed from the Applet and write it to a directory on the server for uploaded image file.

4. Conclusions

This work is the beginning stage of the full functional parallel process model and it sets up the framework to pursue distributed computing using Java RMI. There are much more works and improvements need to be done in this area in the future.

4.1 Full Function Holographic Image Process

The actual implementation of holographic image process should be completed including the Fast Fourier Transformation and Inverse Fast Fourier Transformation. Because of the complexity of these processes, they can be implemented in C/C++ and the Jave JNI interface can be used to connect to these routines.

4.2 Other Applications

Based on our proposed framework, the whole model can be easily modified to apply other tasks where work can be split and carried out on different machines. For example, applications like distributed databases to support automatic generation and grading of mathematics examination questions, evaluation of any question, e.g., in Calculus, can be distributed to any machine independent if the machine has any Calculus compute engine.

5. References

[1] Carl Tu, Sung-Chi Chu, J. Gerlach & T.-C. Poon, *A Parallel Approach to Digital Holographic Reconstruction with Twin-Image Removal*, Proceedings of the IEEE SouthEastcCon '97, April, 1997, pp 258-259.
[2] Sung-Chi Chu, *A Web-based Digital Holographic Image Reconstruction System (DiHIRS),* presented at ATCM97, Penang, Malaysia, June, 1997.
[3] http://java.sun.com/docs/books/tutorial/rmi/ [September 6, 1999.]