Infinite Tuples Final Report

Brad Israel
Jacob Hays
Team Project Final Report
05/20/07

Table of Contents

1. Project Introduction

As wireless devices become more common and more powerful, their ability to perform larger computations increases. We researched into creating a grid computing system over wireless ad-hoc networks. The goal is to create a framework that can split the workload of user created problems amongst other ad-hoc nodes using the same framework. The scope of the problems was narrowed down to just allowing the users to create and solve fractals, but still allowing us to study the techniques used for creating a more advanced system.

2. Project Design

The software contains several user interfaces. One interface gives the user a view of the current problems currently on the network and the options to start or stop them. From here, they can open up another interface which can load a premade java class file that contains the fractal problem. They can then fill in the parameters to the problem, and create the new fractal problem on the network. Another interface is used to view completed and in progress fractal problems. This view also displays statistics so speedup can be measured.

To represent a fractal, the fractal code must be implemented outside our application and complied as a java class file. The java class file can then be loaded and distributed among the helping nodes so that the problem can be split up. All fractal classes must implement the interface FractalImplementation. This interface has one method, getPixelColor(int x, int y) that must also be implemented. The constructor of the java class can contain any number of String arguments, which the user can use to change fractal parameters. The solver portion of the code uses java reflections to load and run this class file on all the client machines.

The communication for the ad-hoc distributed grid was all done using the Tuple board library. There is two Types of Tuples that are placed onto the tuple board. The ProblemTuple contains the problem name and the image parameters. It also contains a java class file that contains the code to create the fractal image. The ProblemTuple is created for each problem and every client will read the tuples to get a view of the current problems being worked on. Along with each problem tuple, is a set of ChunkTuples that are created with it. The set of ChunkTuples represents the current state of the problem, holding what parts of the problem have been solved so far. ChunkTuples specify which part of the problem to solve, and hold the result when it has been solved. The ChunkTuples data section is initially empty, but is updated as more processes complete chunks of the problem.

The Problem/Creator portion of the code consists of the following classes.

- ControlPanelLogic – Sets up Tupleboard and manages the current Problems on the Network. User input starts working on problems, or creates new ones.
- CreateProblem – Reads the problem class file, gets user input parameters, and posts a new ProblemTuple with its ChunkTuples.
- SolverLogic – Keeps track of the Chunks and the state changes of them for a problem. Creates a dispay GUI to show fractal and statistics.
- SolverThread – Thread created to solve a chunk of a problem. Runs the class file to solve for the chunk that it is given.
- ChunkSelector – A thread started up for each problem the program is working on. It chooses what chunks to work on and creates solver threads for each one.

- ChunkCollection – A storage object for the chunk tuples, whose methods are synchronized to work with threads.

2.1 UML

All UML class diagrams can be found in the same folder as the report in the jar file or linked on our website.

Overview class diagram (http://www.cs.rit.edu/~bdi8241/adhoc/Overview_UML.png) – This diagram shows the top-level design for the project, including which thread creates new tuples, how tuples are stored, and how chunk tuples are chosen and solved.

Control Panel class diagram (http://www.cs.rit.edu/~bdi8241/adhoc/Control_Panel_UML.png) – This diagram is the class diagram for the main program and the control panel gui and logic.

Create class diagram (http://www.cs.rit.edu/~bdi8241/adhoc/Create_UML.png) – This diagram shows how new problems are created and how the tuples are posted to the tuple board.

Solve class diagram (http://www.cs.rit.edu/~bdi8241/adhoc/Solve_UML.png) – This diagram shows how the solver gets chunk tuples from the tuple board, stores them, chooses them, and solves them.

3. User Manual

System Requirements:

- Sun Java JRE 1.5
- Put the Tuple Board Library in your class path (Found on Professor Kaminsky's website)

Installation:

- Extract the jar file using the command, 'jar -xvf IT_Team_Project.jar'
- Enter the “Distributed Fractal Generator” folder and compile the source using the command, 'javac AdHocFractals.java'

3.1 Creating a new fractal

- Create a class that implements FractalImplementation
- The fractal is calculated one pixel at a time, with each pixel representing an imaginary number. The program will split up the problem into pixels, all that has to be implemented is a method to return a java.awt.Color for each pixel.
- You must first create a constructor with the parameters that you want inputed into the fractal. Even if you do not have any parameters, a constructor needs to be created.
- The parameters of the constructor must only be Strings, same as if you were receiving constructors from the command line.
- Next, implement the **public java.awt.Color getPixelColor(double x, double y)** method. The parameters x and y represent the complex number which you can use to calculate the color at this pixel. While the image is shown on a 2D pixel grid, it is calculated on the complex plane, x and y are the complex number which you do the calculation with.
- Using this interface, you can create any fractal in which each point can be calculated independently from each other point.
- For more information about fractals of this type, see

http://en.wikipedia.org/wiki/Mandelbrot_set

- Figure 1 shows an example of a image which will return a random color for each pixel location.

```
public class myTestFractal implements fractalImplementation {
	private int maxIter;
	private double breakOut;

	public myTestFractal(String maxIter, String breakOut){
		this.maxIter = Integer.parseInt(maxIter);
		this.breakOut = Double.parseDouble(breakOut);
	}

	public Color getPixelColor(double x, double y) {
		Random chooser = new Random();
		Color pixel = new Color( chooser.nextInt(255), chooser.nextInt(255),
			chooser.nextInt(255));
		return pixel;
	}
}
```

Figure 1: Example Implementation

3.2 Using the Control Panel GUI

Start the program using the command, 'java AdHocFractals'.

When the program first starts up, it will bring you to the control panel, and will look similar to the one in Figure 2.

The list on the left will tell you the names of any fractals that are currently on the network from other people. To the left is buttons to control the operation of your client. To start working on one of these problems, select the problem from the list and hit the start button. Another GUI will pop up and the problem will be marked as running. If you want to stop working on a problem, you can select that problem and press the stop button.

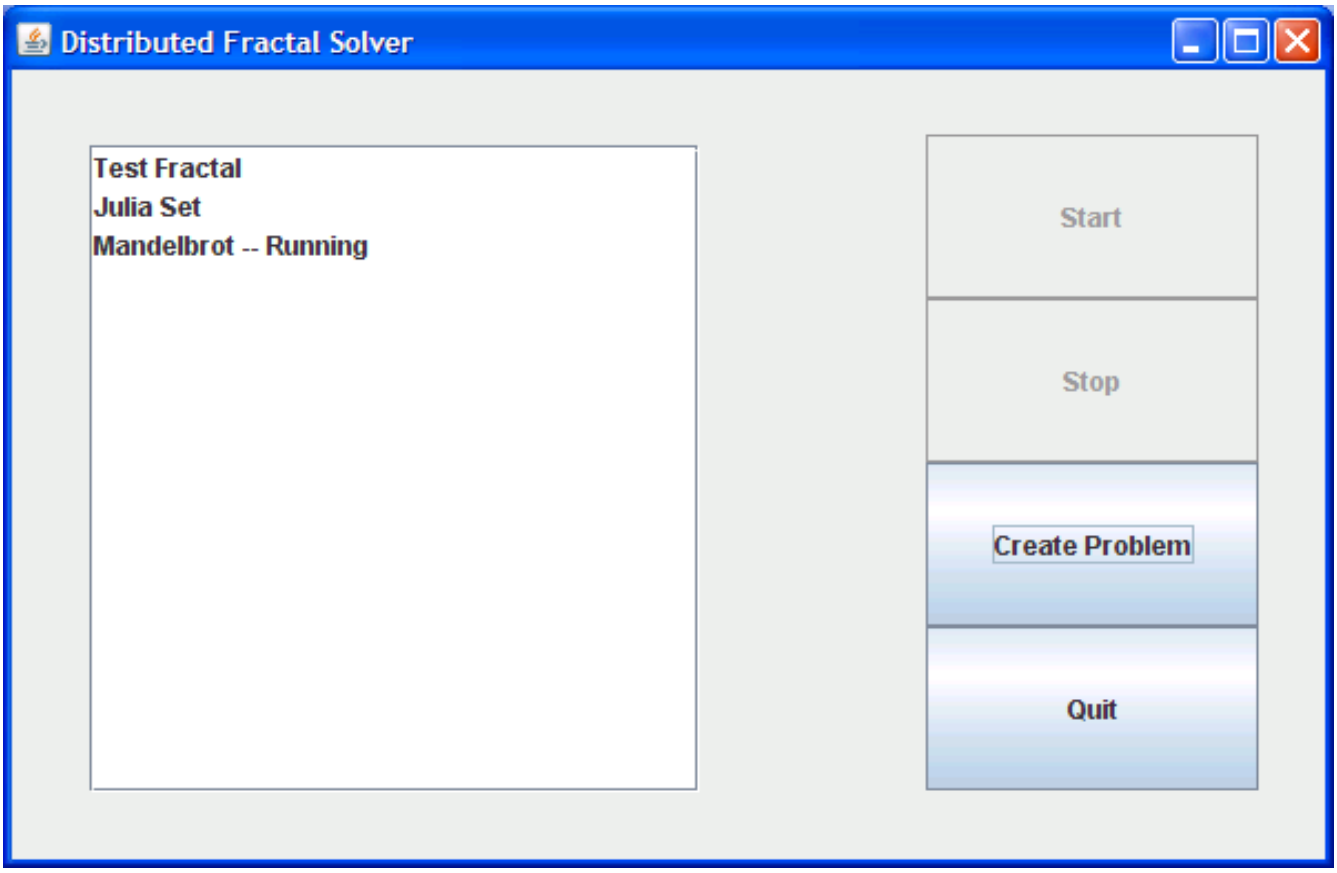

Figure 2: Control Panel GUI

3.3 Create Problem GUI

To create a problem, select the Create problem button on the Control panel. A file selection menu will pop up and ask you for a .class file. Select the .class file you have made previously that implements FractalImplementation. You can also select one of the preset fractals included with the program. myTestFractal, myMandelbrotFractal, and myJuliaFractal. It will open up a window like the one in Figure 3. While the image is shown on a 2D pixel grid, it is calculated on the complex plane, these parameters determine how the two are converted. Once al the parameters are filled in, hit the create problem button and the problem will show up on the Control panel problem list.

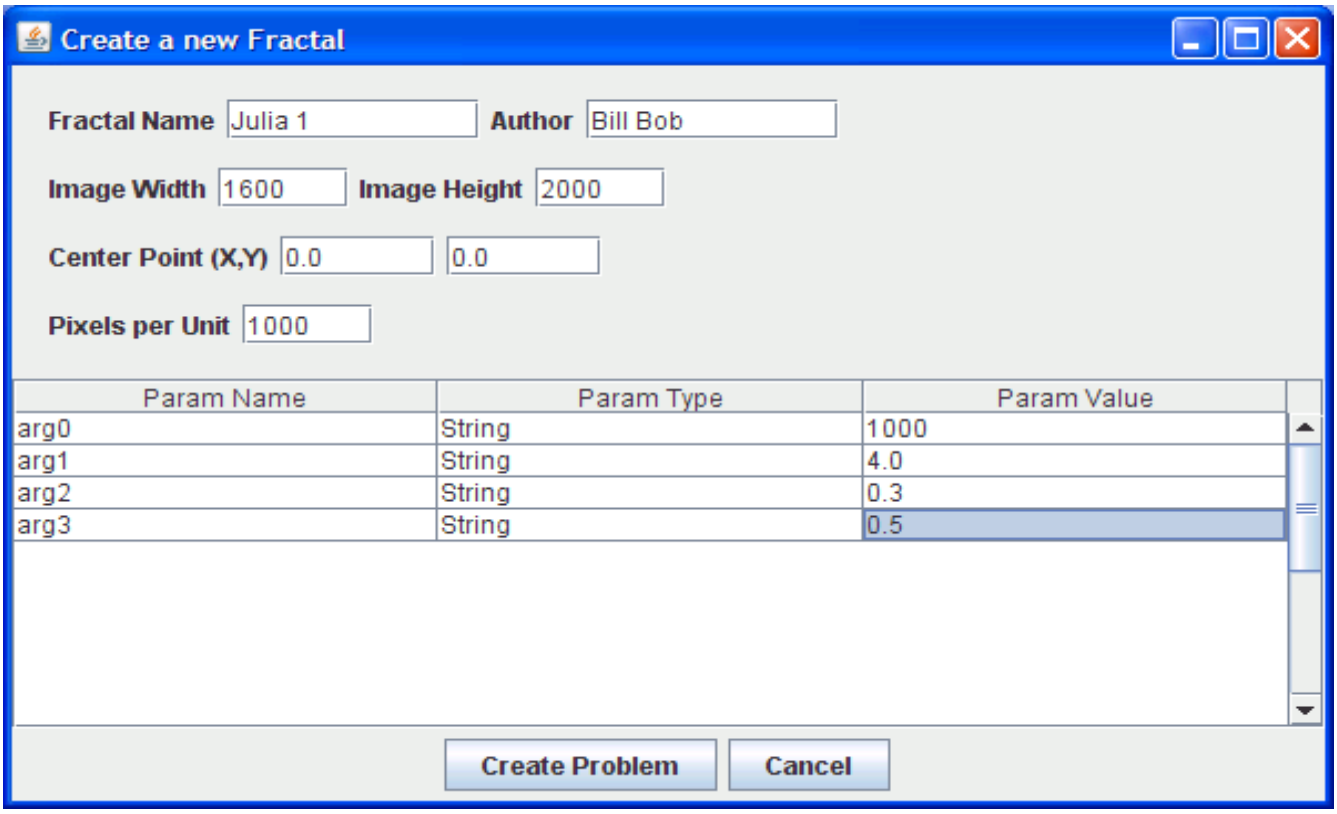

Figure 3: Create Problem GUI

Parameter Description:
Fractal Name: The name used to identify your fractal on the list.
Author: The name of the author or creator.
Image Width/Height: The size of the image you are creating.
Center Point (X/Y): The point that the image will be centered on. (Complex number)
Pixels Per Unit: This parameter is the resolution that the image is looking at. If the Pixels per unit is 1000, then each pixel represents a location that is 1/1000 away from the neighboring pixels.

Specific parameters to Preset Fractals
myMandelbrotFractal:
arg0: The number of iterations to compute before placing this point into the Mandelbrot set.
arg1: The breakout value. If the iterations reach above this breakout value, then it is concluded that this point is not in the Mandelbrot set.

myJuliaFractal:
arg0: The number of iterations to compute before placing this point into the Mandelbrot set.
arg1: The breakout value. If the iterations reach above this breakout value, then it is concluded that this point is not in the Mandelbrot set.
arg2/arg3: The complex number that the Julia set uses to calculate during the iterations.

Once the problem is created, you can start working on it by selecting it from the control panel.

3.4 Fractal Display GUI

When a problem is started, a display will come up showing the fractal and the current progress of the fractal. This process will then start working on this problem, grabbing and finishing chunks of it. As this process and other processes complete chunks of them, the image of the fractal and the statistics will be updated (see Figure 4).

Description of statistics:
Time worked on(ms): The time that *only* this process has been working on the problem. This will not be an accurate representation of the time taken to complete the problem unless this process is the first process to start working on the problem. It will then be the time taken for the starting process to start the problem till it receives the final chunk.
Total Computation Time(ms): This is the summation of all the computation times from all the chunks it has received so far. When finished, this time will represent how much total work all the participating processes have put into the problem, so It could be higher then the time worked on.
Time per Chunk(ms): The average amount of time that it takes the processes to finish a single chunk of a problem.

The progress bar will reflect how much of this problem is completed. At any time, the Save Image can be pressed to save the current fractal Image to a PNG file.

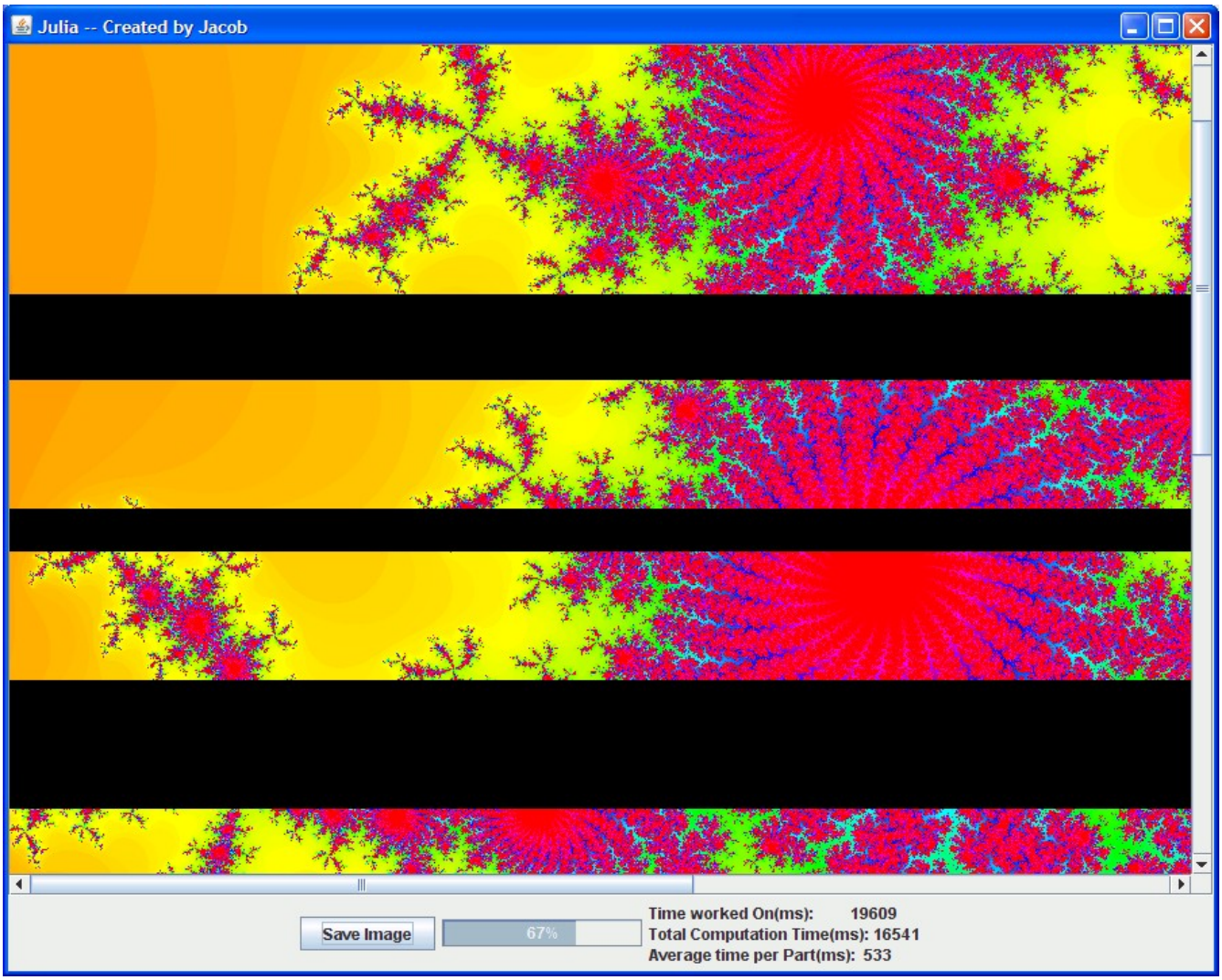

Figure 4: Fractal Display GUI

4. Test Results

We ran our program through several different tests to find the efficiency of the application.

Test: Mandelbrot set with following parameters:

Class File: myMandelbrotSet.class

Width: 2500
Height: 2500
X center: -0.75
Y center: 0.0
Resolution: 6000
Max Iterations: 1000
Break out Value: 4.0

4.1 Baseline Test

This test will see the amount of time to calculate using one processor without any extra features.

Use Sequential Mandelbrot set program

#	**Time**
Total Time	105836
Comp Time	105339

4.2 Baseline Test using Java Reflections

This test will see if there any slow-down when running the same test using Java reflections

Use Sequential Mandelbrot set program that runs the computation through Java Reflections.

#	**Time**
Total Time	43988
Comp Time	43744

4.3 Baseline Test using Reflections and Tuple Board

This test will see if there any slow-down when running the previous test but through the final program with TupleBoard, and test Speedup with multiple processors.

Run the Distributed Fractal Generator program.

Num Processes	Total Time (ms)	Computation Time(ms)	Avg Time per Chunk (ms)
1	90858	53461	106
2	78634	54919	109
3	71393	7277	144
4	70283	81552	163
5	64331	83753	167

4.4 Conclusions

There seems to be an anomaly in our tests. The Test without Java reflections is much slower then the test where it is used. When Java Reflections is used it should cause some amount of slowdown, so the results should be switched. From the looks of the test, there is quite a large increase in computational time when adding in the tuple board. If we were going to implement this for real, we would probably use some sort of stripped down version of the tuple board that has less overhead.

5. What We Learned

Sharing computational resources in an ad hoc network is possible and can be very useful in the future. There is nothing the users have to set up for ad hoc distributed networks as they do that themselves. There is no extra infrastructure that is needed as well. But because of the lack of centralized services, and the frequent adding and removing of nodes, ad hoc distributed networks have a lot more overhead then conventional ones. In the case of our project, the overhead of the ad-hoc network slowed us down enough that even with 5 processes working on a problem, a single process without overhead was faster. While any network should add some overhead, it seems that using the Tupleboard greatly increased the amount much more then we expected. We also discovered that storing the fractal instructions in Java class files is not only a very flexible way to implement multiprocessing, but it is also quite efficient.

6. Future Work

Possible future features include:

- Give the user control in how to split up the problem.
- Implement various chunking methods, including methods that can split up chunks after the problem has started. (Ex. Logarithmic chunking)
- Allow user to specify how many problems to automatically work on as soon as they are posted.
- Allow the user to set the number of solving threads for each problem that they are working on.
- Expand the middleware for a broader use for things other then fractals. E.g. Image processing, or using Matlab scripts instead of Java class files to define problems.
- Improve the selection of problems to work on GUI.
- Increase efficiency of communication. Possibly remove Tupleboard.

7. Research Paper Analysis

7.1 A Collaborative Problem-Solving Framework for Mobile Devices

Mobile Internet connected devices have grown in huge amounts recently. As the amount and processing power of the mobile devices grows, so does the ability to have these devices be used for large scale computation. A new field is developing collaborative applications for mobile devices. These include peer to peer and grid computing. Grid computing is a field of research Collaborative applications in mobile networks run into many problems not found in static networked collaborative applications.

The quality of service is very important to the performance of the application. Network instability is inherent in the system, as Wireless communication has more interference, and the devices are mobile and can enter/leave the network at any moment. The issues that comes up is how the work load is redistributed when devices enter or leave the grid. If large numbers of devices enter or leave the network in a short time, their could not be enough resources to continue working on current tasks.

This paper presents a framework for a wireless grid. The architecture of the grid can be explained as four parts. Subordinate, Initiator, Brokering Service, and Keep alive server. The Subordinate and Initiator are two roles that each of the wireless devices connected to the network can be. The Initiator is any device initiates a task or computation for the grid to work on. A subordinate is any device that participates in solving a task on the grid. All devices advertise them selves as part of the network to the Keep-alive server. This server keeps tracks of available devices and passes this information on the the Brokering service. The Brokering service is the task manager of the network. An Initiator will tell the Brokering service the problem, and it will distribute the problem among the available subordinates. If devices enter or leave, the Brokering service will redistribute tasks among the remaining nodes.

To test out the architecture, a simulation of a mobile wireless grid was created. There was 3 main control variables, grid population, device mobility, and task initiation frequency. A experimental plan was presented, but the results were going to be included in future articles.

The article outlines many of the issues that will have to be resolved when creating collaborative applications. All of these have to be considered when we design our project. The solution that they propose involves centralized servers such as the Keep-alive and Brokering servers. Our system will not be centralized, focusing more on the flexibility of a pure ad-hoc design. Both our systems will perform in a similar manner though. Any user of the network can initiate a task, and the others will help compute the solution. A difference in our approach is that we plan to create an actual working application instead of a framework that can only be tested in simulations.

7.2 Hot Service Deployment in an Ad Hoc Grid Environment

In the paper, they present their solution for dynamically deploying grid service factories onto computing nodes. The nodes are running an implementation of the Open Grid Services Infrastructure (OGSI). Grid service factories are services which create other instances to run computation, and allowing dynamic deployment of them will all organizations to form ad-hoc grids with their unused and scattered resources.

The basic model of a Grid is a collection of high performance computing clusters that can work together and be accessed by all participates. A Service Oriented Grid (SOG) which combines all available clusters into a single resource can create a more versatile grid. An example of this would be having all the idle workstations be placed into a dynamic ad hoc grid to handle extra computation when the main grid is overloaded. This paradigm needs a more flexible infrastructure to handle topology changes, but this allows extension of the network to be much easier.

When making an ad hoc grid, several requirements must be met. Unlike a static, unchanging centralized grid, the ad hoc grid must be able to undergo frequent nodes entering or leaving the grid. The number of possible nodes can be much higher then in a traditional grid, so the ad hoc middleware must be efficient. The following considerations must also be met.

- Service and resource discovery – A reliable efficient way to detect new and leaving nodes without too much network overhead.
- Service Deployment – Services must be automatically deployed to machines in the network, manual deployment is too slow for an ad hoc network.
- Discovery and Platform capabilities – The implementation of the service should be hidden and system independent.
- Security – Many new security scenarios arise in a large ad hoc grid. Malicious code must be prevented from being inserted into the network, and uncooperative nodes need to be dealt with.

Open Grid Services Architecture is considered the foundation for service oriented grid computing

Hot service deployment is a meta-factory used for deployment of services. It was implemented using the Java implementation of GT3 with Tomcat, as it is the most common used OGSI platform. If something requests a new service from the service factory, the service factory will create a new unique service and return it. The unique handle is used to interact with the service and its users. The service can change locations on the network, and still the factory can communicate with it.

The Hot Deployment Service (HDS) deploys service factory's using a Grid Service Archive (GAR) file that contains the classes, schema, and other parameters. The application has to make sure that every node it wishes to use for computation has the corresponding service factory. If there is no other services on a node, it can deploy a new service factory to a node. Service factories can also be undeployed and redeployed onto different nodes€.

To HDS uses a variety of functions to interface with nodes. It can use different versions of Java class loaders for flexibility. Each service has its own class loader, to ensure security among the nodes. If the factory is removed, no new services can be created.

The Hot Deployment Service was tested using a service called TimeBeacon, which just returned the current time of the systems. The HDS was seen in the paper as just one of the important elements in creating an ad hoc grid. It must be able to remotely deploy grid services without disrupting others, and

ensure security. Future work would include adding versioning and improve the dynamic node discovery.

This article focuses on a small part of the many problems to be solved when taking Grid computing into ad hoc networks. The work has to by dynamically distributed to the changing computational nodes. We did this by using the TupleBoard, which broadcasts the problem to the rest up the network in Tuples everyone can see. Inside the tuple, the problem is contained in a Java class file, to allow for flexibility in the problems that can be solved.

7.3 Wireless Grids – Distributed Resource Sharing by Mobile, Nomadic, and Fixed Devices

The paper talks of a new type of resource sharing network, wireless grid computing. Normal Grid computing lets devices connect through the Internet to share resources. This can be expanded to Mobile wireless devices, but has many differences compared to fixed location devices. Wireless grids are easy to expand to large numbers of devices and may more interest casual users, instead of the large research applications that grids are used for now.

Devices on wireless grids are nomadic, entering and leaving the range of each other often. The mobile nature of wireless allows grids to be used in settings where they have never been used before. It also adds another level of difficulty in creating them. Wireless grids can even take advantage of other wireless technologies, including sensor networks. Grid computing started with large super-computers only corporations could buy. Grids then shifted to cheaper clusters of networked machines. Perhaps the next step is wireless grids?

Wireless grids run into many challenges. Like all wireless communication, it has to deal with sharing the same transmission medium with every other wireless device that exists. Wireless devices have to be much more careful about power management and power saving. The challenges can be outlined in five requirements

- Resource Description – Each device must describe what resources it can share.
- Resource Discovery – Various standards can be used to allow devices to publish their resources.
- Coordination Systems – How to allow one device to use the others resources, and how to pool the resources. How to coordinate with nomadic devices.
- Trust Establishment – How to identify trustworthy devices to share and resist man in the middle attacks.
- Clearing Mechanisms – How devices exchange resources, work, and power saving with each other.

The paper also looked at a particular application for wireless grids. DARC* is an audio recording and mixing software, which allows many ad hoc wireless devices create individual recordings, and mix them together to create a superior recording or a surround sound recording. A user starts a recording session and takes the role of the mixer. Any users may then join the session, sending their live recordings to the mixer. The mixer will then broadcast back the mixed signal.

This article outlines many of the problems that must be considered when creating adhoc wireless grids. Our project is not dealing much with wireless, but some of the challenges must still be addressed.

8. References

Lee W. McKnight, James Howison, Scott Bradner, "Guest Editors' Introduction: Wireless Grids--Distributed Resource Sharing by Mobile, Nomadic, and Fixed Devices," IEEE Internet Computing, vol. 08, no. 4, pp. 24-31, Jul/Aug, 2004. http://doi.ieeecomputersociety.org/10.1109/MIC.2004.14

Kurkovsky, S., Bhagyavati, and Ray, A. 2004. A collaborative problem-solving framework for mobile devices. In Proceedings of the 42nd Annual Southeast Regional Conference (Huntsville, Alabama, April 02 - 03, 2004). ACM-SE 42. ACM Press, New York, NY, 5-10. DOI= http://doi.acm.org/10.1145/986537.986540

Friese, T., Smith, M., and Freisleben, B. 2004. Hot service deployment in an ad hoc grid environment. In Proceedings of the 2nd international Conference on Service Oriented Computing (New York, NY, USA, November 15 - 19, 2004). ICSOC '04. ACM Press, New York, NY, 75-83. DOI= http://doi.acm.org/10.1145/1035167.1035179