Java-based Volunteer Grid Architecture

Robert R. Puckett
Computer Science Department
University of Hawaii at Hilo
Hilo, HI, USA
justpuckett@gmail.com

Abstract – *Grid Architecture coordinates the sharing of resources and allows for collaboration of often heterogeneous equipment that is geographically distributed. While research grids provide needed services, volunteer-based grids promise greater processing power provided that the overhead, availability, and heterogeneity of the resources are manageable. JAVA-based applets provide quick installation across heterogeneous platforms and are well placed to speed the transition to grid computing.*

Keywords: Grid Computing, distributed computing, JAVA applet, benchmark, resource management.

1 Introduction

Companies and institutions wishing to expand their processing power have typically faced among three typical options. That is, they can buy more equipment, subscribe to a service provider, or make better use of the equipment they have. Grid computing falls between the last two options. Research grids can provide institutions with the ability to combine their processing power with other institutions' resources into a collaborative, shared resource which outstrips the capabilities of any one institution. At the other end of the spectrum, volunteer grids form from users installing software to share their underutilized computer resources with a central authority which disseminates work units.

This paper provides an overview of grid computing as well as a case study at the University of Hawaii at Hilo computer science department. As part of the computer architecture class, the case study centered on using GreenTea Technologies' java-based grid computing software to build a distributed peer-to-peer grid. This tool is used to perform benchmarks and analysis on various organizations of computers scaling from single computers, to a collaboration of two small clusters.

Volunteer grids provide the opportunity for users to volunteer their personal computing power toward causes they deem worthy. Current such grid projects include Folding@home and Seti@home. Additionally, Seti@home averages 13Tflops of processing power making it more powerful than even the top three supercomputers combined. As such, Gordon Bell projects that grid computing and peer-to-peer computing coupled with Internet 2 will remain the most powerful supercomputer [7].

2 Grid Computing

Grid computing differs from many peer-to-peer distributed computing programs in that there is a central authority computer which manages the distribution and collection of work units. To process a job on a grid computing system, one would connect to the central authority and request one's job be processed.

When the central authority receives the job, the job must be subdivided into work units for processing on the work nodes, which are the work computers that are members of the grid. If the job is an application, then it must be designed for parallel computation. Alternatively the job could be a sequence of input to run against an existing grid-based application, such as a gene search against a genome database. Figure 1 demonstrates an example architecture for a grid.

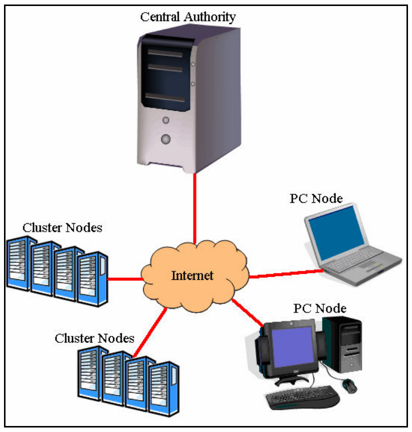

Figure 1 - Grid Architecture Example

2.1 Categories of Grids

Although grid computing serves a diverse market ranging from commerce to research, grid computing is typically divided into three broad categories: computational grids, data grids, and equipment grids.

2.1.1 Computational Grids

Computational grids focus on providing the raw computational power needed to solve computation intensive problems. Bioinformatics benefits from grid computing as gene sequencing and protein folding require immense computational power. Through computational grids more processing power can be harnessed than any single supercomputer.

2.1.2 Data Grids

Immense repositories of data, such as gene data banks, are overly burdensome to maintain multiple coherent copies. To provide higher levels of RAID redundancy and integrity may exceed or encumber the financial resources of an institution. Thus it is opportunistic for companies to cooperate and share repositories of data for research. While some overhead exists from not having local copies of the data, appropriate management of resources can ensure that this overhead is kept to a minimum. Data grids provide for control and access of large repositories of data which may be geographically distributed.

2.1.3 Equipment Grids

Institutions with a primary piece of equipment and various associated support equipment may benefit from forming an equipment grid. For example, a telescope will have control, processing, and analysis equipment to support it. As such, an equipment grid can allow remote control and access to the unique resources of an institution.

2.2 Applications for Grid Technology

Applications that run on grid technology tend to fall into two classifications: highly parallel problems, and dependent parallel problems.

2.2.1 Highly Parallel Problems

Highly parallel problems can be decomposed into independent work units. Such problems are often dubbed "embarrassingly parallel" problems as they do not rely upon data within any other work unit which may be on a different computer. As such there is little or no communication between multiple nodes. Embarrassingly parallel problems include ray tracing, single frame rendering, and brute force cryptography.

2.2.2 Dependent Parallel Problems

In contrast to highly parallel problems, dependent parallel problems require much greater communication between the work nodes to derive solutions. Such problems rely on values derived in multiple work units. Some examples include heat diffusion, ecosystem modeling, and climate modeling.

2.3 Grid Computing Software Components

The software that drives grid computing must be adaptable to run the desired computers that will serve as work nodes in the grid architecture. Making the client software that runs on the nodes independent of the system architecture or operating system permits a highly heterogeneous community of computers to contribute.

2.3.1 Resource Management

Resource management involves dividing and disseminating the work units. The resource manager also may play a role in supporting collaboration between nodes for dependent parallel problems. Work division must balance communication overhead with computation time. That is, the greater number of work units, the greater the overhead from communication in transmitting and receiving the data to the nodes. On the work nodes variables that may affect the allocation of work units includes processor speed, disk space available, and quality of network connection.

2.3.2 Resource Discovery

Part of the management of grids includes software to discover new resources. Such software may include the capability of identifying computers running the client software within a network. Additionally, grid nodes may notify the central authority computer of their presence and readiness for processing jobs.

2.3.3 Resource Monitoring

Grid management software may be integrated with a grid monitoring application. Monitoring is important to ensure efficient and even distribution of work to resources. The software may provide a graphical interface or merely text log files.

2.3.4 Integrity and Security

Ensuring integrity and security is vital yet more challenging on a grid architecture given the typically distributed nature of the medium. Computers cooperating to build a solution may be separated by thousands of miles. Thus, the grid management software must ensure that the work units maintain their integrity from both accidental and malicious modifications.

Furthermore, given the great processing power capable in a grid, security is paramount to ensure that the grid software does not permit unauthorized users to access the machines. In addition to wasting resources, such access could be used to launch distributed denial of service attacks. Grid applications, like any other software, can be vulnerable to stack, buffer, and memory overflows which could permit malicious users to gain unauthorized access to node computers. Although the grid application may be secured, the tools used to build it such as the Globus toolkit may potentially hide additional security threats. Additionally missed upgrades, invalid certificates, and rebuilds of software may threaten security.

Even the Teragrid, an amalgamation of supercomputers forming a research grid in the United States, has proven susceptible to hacker intrusions. [1] Although providing openness and ease of joining or using a grid may seem like idyllic gestures, they can lead to compromised security and integrity of systems.

2.3.5 Transmission

Grid computing must provide a mechanism for transmitting and receiving work units between the central authority computer and the work nodes. Generally grid computing uses the existing TCP/IP protocols as a backbone for transmission. Additional application layer protocols may be needed to ensure that work units are not lost from, for example, a work node going offline.

2.4 Challenges to Implementing Grids

First of all there exists a public stigma attached to grids that they are primarily for grand scale projects requiring massive amounts of computation. However, grid architecture can scale up or down to meet the needs of a diverse population of clients.

Secondly, most commercial software is not designed to run on grid architecture. Similar to the transition from single processor to multiple processor machines, the software must be adjusted at the code level to accommodate spreading the distribution of work across multiple machines on a grid.

However, consider a common operation in a business, that of querying a large database. This operation may be performed during payroll, sales, and support. On a single server system the server may need to scan a large portion of the database to return results for a query. A grid computing solution to this problem might divide the database into hundreds of work units and send it off to the work nodes in the field. The work nodes would run the query against its small subset of the database and return its result. A simple merge sort could then combine the results into the final result set returned to the user. Thus, the server would experience a reduced load as the work is distributed to the grid and the grid would function like a giant multi-processor machine. Therefore, the conversion of certain business operations to run on a grid could well justify the expense.

One other challenge facing grid computing is the lack of standards as the technology is still somewhat immature. Companies vying to create standards for grid computing include GCF, W3C, and OASIS.

3 Case Study: A Java-based Grid

Although development of complex software applications for grid architecture may be beyond many companies' capabilities, there exists a simpler method for incorporating benefits of grid computing. Java-based volunteer grids incorporate a platform independent application which manages the grid of computers. One such program is GreenTea provided by GreenTea Technologies.

3.1 GreenTea Description

GreenTea is a Java-based application that facilitates parallel computation, which can include peer-to-peer computing, grid computing, distributed computing, and network computing. Since it is programmed in Java, the application can run on most any platform including UNIX, Linux, Windows, Mac OSX, and even Java-enabled cell phones. [4]

Additionally, GreenTea is far simpler to install and configure than MPI or other low-level tools such as the IBM Globus Toolkit. However, GreenTea does cause 15-20% overhead. [4]

3.2 Project Overview

This project centers on researching the relationship between speedup of a process and the number of computers on a grid. Amdahl's law [7] has been applied to grid applications as a means of estimating the maximum speed up from applying a given code to parallel execution. Maximum speed up is given by the equation:

$$\frac{1}{F+(1-F)/N} \tag{1}$$

F represents the fraction of the code that is serial and cannot benefit from parallelization. N represents the number of processors, in terms of grid computing, work nodes that will work on the code. However, this law ignores the overhead from communication between nodes.

Therefore, the speedup is limited by the amount of overhead from communication between the nodes. From analyzing benchmark runs on various configurations it may be possible to derive the maximum number of nodes a given piece of code could derive benefit under parallel execution. If the percentage of serial vs. parallel code can be approximated, this may aid in the development of an optimization routine. Since both too little and too much division of the work can lead to sub-optimal time for completion, an optimization routine is desirable.

3.3 Test 1: Ray Tracing Proof of Concept

In this experiment I performed the ray tracing demo included with the GreenTea demo package. For all tests herein the number of lines and sphere/line is fixed at 10. The three trials, demonstrated in figure 2, show variations in the time not only from varying the number of nodes, but also from varying the number of subtasks. As shown, increasing the number of subtasks tends to decrease the time to complete the ray tracing operation. However, increasing the number of nodes tends to provide the greatest benefit.

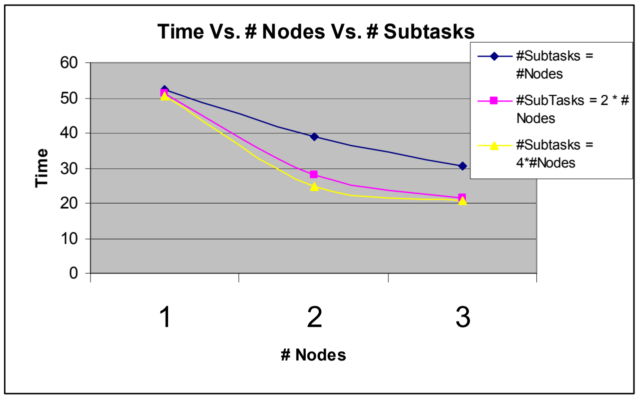

Figure 2 - Green Tea Trials

3.4 Test 2: Prime Numbers

In this experiment I developed a prime number validation routine that operates in a distributed manner across a GreenTea network. First, the program divides the test value in half since no factors can be larger than half the test value. Actually, the square root of the test value might have been more appropriate here. Anyway, next the lower half is divided by the number of word nodes in the GreenTea network. This value is provided by one of the GreenTea classes.

Next, the work classes are stored and subsequently sent through the network to the work nodes. There the work nodes check ever value if the work unit includes values less than ten, or every odd value if the work unit is only numbers greater than ten. While this causes many numbers to be checked whose factors have already been checked, this method is very simple and it would be inappropriate to maintain a list of the prime numbers.

Lastly, a Boolean value representing whether the work unit's interval is relatively prime to the test value is returned to the main program via the network. The main program compares all returned results, returns the results to the user, and displays the total elapsed time for the process.

3.5 Test 3: Monte Carlo Integration

In this program I have coded a Monte Carlo integration estimation to the integral of one fourth of a circle with radius 1. This is accomplished using the Monte Carlo integration by inscribing that quadrant within the unit square. Note that this is equivalent to formula 2. For this I have implemented an existing random number generator. Supposing that the random number generator is sufficiently random, the Monte Carlo technique will provide an accurate estimation to the integral value at high numbers of intervals.

$$\int_0^1 \sqrt{1-x^2} \qquad (2)$$

For the random number generation I chose the fourth example from Park and Miller [9]. This example stipulates that MAXINT must be smaller than 2^{31}-1. Although quasi-random numbers might be more appropriate for a Monte Carlo simulation, this implementation uses pseudo-random numbers. The convergence of this solution, where N is the number of trials, is:

$$\frac{1}{\sqrt{n}} \qquad (3)$$

Overall the development of the random number generating code is relatively straightforward and self-explanatory. One point of note is that each of the work nodes needed to receive its own unique seed for the random number generation. Without that, the pseudo-random nature of the random number generator would produce identical results across most of the work nodes. Thus, the main program used its own copy of the random number generator to produce pseudo-random seeds which were then passed to and used in the work nodes.

Although the Monte Carlo method for integration approximation does converge upon the correct answer, it does so at an incredibly slow rate. If high rate of accuracy is needed, this method may be prohibitively expensive in CPU time. However, for what Monte Carlo lacks in speed to a solution, it makes up for in simplicity. Of course, the accuracy of the Monte Carlo method is dependent on the random number generator providing truly random numbers. Additionally, owing to the nature of the random numbers, the value of the estimation to the integral will not converge in a strictly decreasing manner. This method may be useful for functions that are difficult or cumbersome to integrate analytically, as was the example above. Additionally, higher dimension integrals can be performed in a similar manner by enclosing the desired portion within a known area/volume/etc.

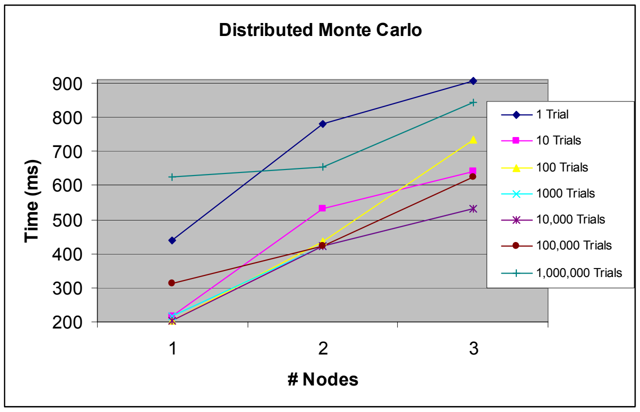

Figure 3 - Monte Carlo Trials (Lower Order Trials)

Figure 3 demonstrates that there is not always a cost savings in a distributed solution. As the number of nodes increased, the time to complete the same task actually increased! Additionally, the data shows that some subsequent, in other words larger, tests perform better than test cases where there is fewer trials. One reason is that overhead from division and distribution of work units plays a major factor in determining the run time of a program. Figure 4 demonstrates that in higher order trials there is indeed an appreciable decrease in completion time for using more nodes. The break even point for this algorithm appears to be around ten million trials.

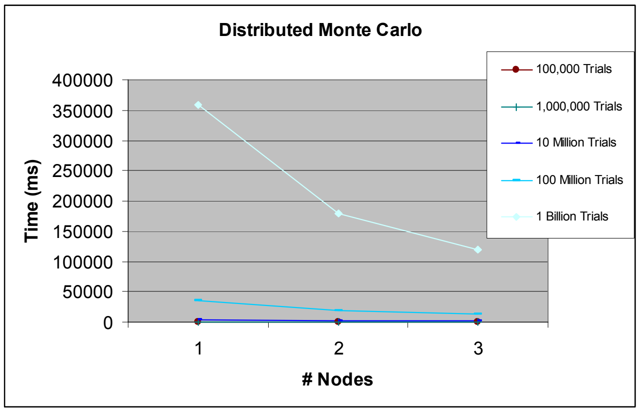

Figure 4 - Distributed Monte Carlo (Higher Order Trials)

3.6 Conclusions

Thus far programming experiments have centered about simple distributed programs based upon highly parallel problems. The programs have not incorporated timeouts for lost work units, such as from a computer node going offline. These can be incorporated in later upgrades to the code through. Additionally, the prime number and Monte Carlo programs I have developed have not used floating point numbers, which appears to be somewhat of a weakness in using JAVA for high-performance computing [8].

Although I had hoped to prove Amdahl's law as part of this experiment, it is ungainly to estimate the actual percent serial code of my experimental code inhabited in the GreenTea framework. However it may be possible to extrapolate the approximated percent of serial code from the speedup observed from increasing the number of nodes. One approach might be to perform an order two least-squares curve fit on observed speedup from the node count and time value pairs, such as from Figure 2. Then one could compare the resulting equation to the formula for maximum speedup given in Amdahl's law, Equation 1.

3.7 Future Work

GreenTea Technologies has been gracious enough to provide sponsor my research with a twelve-license issue of their product. As such, future research could not only include groups of standalone computers, but also the two test clusters in development by the other groups in the computer architecture class of the computer science department at the University of Hawaii at Hilo. Though using parallel execution programs as well as system monitoring software, such as Ganglia, future research can demonstrate the cost and benefits of java-based volunteer grid computing. It will be interesting to see the breakeven point between communications/java overhead and the division of work to an increasing number of nodes.

References

[1] Koch, Lewis E. 2004. A Quiet Time Bomb. The Raw Story. Retrieved February 23, 2006 from: http://www.rawstory.com/exclusives/koch/vulnerable_computer_grid.htm.

[2] Levinson, Meridith. 2005. Who's Afraid of Grid Computing? CIO.com. Retrieved February 25, 2006 from: http://comment.cio.com/soundoff/042505.html.

[3] Peel, Roger M. Grid Computing. Retrieved February 26, 2006 from: http://www.computing.surrey.ac.uk/personal/st/R.Peel/csm23/parallel-1.pdf.

[4] GreenTea Technologies, Inc. GreenTea User Manual. Retrieved January 25, 2006 from: http://www.geocities.com/gtusaus/current/docs/readme.html

[5] Foster, I. & Kesselman C. 1997. Globus: A metacomputing infrastructure toolkit.

[6] Gustafson, John L. Reevaluating Amdahl's Law. Retrieved February 20, 2006 from

http://www.scl.ameslab.gov/Publications/Gus/AmdahlsLaw/Amdahls.html.

[7] Bell, Gordon & Gray, Jim. 2002. What's Next in High-Performance Computing? Communications of the ACM, Volume 45 Issue 2. ACM Press. February.

[8] Kahan, W. & DARCY, J. 1998. How JAVA's Floating-Point Hurts Everyone Everywhere. ACM 1998 Workshop on Java for High–Performance Network Computing. Retrieved February 26, 2006 from: http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf.

[9] Park, S. and Miller, K. 1988. Random Number Generators: Good Ones are Hard to Find. Comm. ACM 31, 1192-1201, 1988.