8-2005

Automatic Service Discovery for Grid Computing: A Decentralized Approach to the Grid

Jeffrey Michael Larkin
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Jeffrey Michael Larkin entitled "Automatic Service Discovery for Grid Computing: A Decentralized Approach to the Grid." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jack Dongarra, Major Professor

We have read this thesis and recommend its acceptance:

Tom Dunigan, Jim Plank

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Jeffrey Michael Larkin entitled "Automatic Service Discovery for Grid Computing: A Decentralized Approach to the Grid." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

__Jack Dongarra__
Major Professor

We have read this thesis
and recommend its acceptance:

__ Tom Dunigan _____

___ Jim Plank _____

Accepted for the Council:

____Anne Mayhew____
Vice Chancellor and
Dean of Graduate Studies

(Original Signatures are on file with the official student records.)

Automatic Service Discovery for Grid Computing: A Decentralized Approach to the Grid

A Thesis Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Jeffrey Michael Larkin

August 2005

Abstract

The purpose of this document is to explore the feasibility of a decentralized system for grid computing. A proof-of-concept grid computing system was created based on the GridSolve computing environment and multicast domain name server with service discovery. This experimental grid computing environment showed that reasonable results can be achieved in a decentralized environment. Using these results suggestions are made for future research in this area.

Table of Contents

List of Figures

List of Abbreviations

DNS – Domain Name Service
DNS-SD – Domain Name Service w/ Service Discovery. For the purpose of this document, it will be assumed that DNS-SD also includes multicast DNS.
GridRPC – Grid Remote Procedure Call standard
mDNS – Multicast DNS
NAT – Network Address Translation, a technology used to address the increasing number of network interfaces by translating private network addresses to public Internet addresses
OGSA – Open Grid Services Architecture
SLP – Service Location Protocol

1. Introduction

In recent years an abundance of grid computing research has taken place, producing several widely-used grid computing environments. Although each of these environments takes a different approach to distributed computing, one common characteristic of most of these environments is the use of a centralized authority to serve as a directory and scheduler of available computing resources. The question posed by this research is whether such an authority is needed, or to restate the question: could a completely decentralized environment be developed that achieves results similar to a centralized system?

To avoid the need to write a completely new grid computing environment to test this question, which could take several years to complete, a proof-of-concept environment was developed using already existing technologies. GridSolve, from the Innovative Computing Laboratory at the University of Tennessee, was selected as the basis for experimentation and was modified to use an automatic service discovery library for selecting available resources. Using GridSolve as a starting place for this research not only expedited the development, but also gave a base of comparison for evaluating the results. A more detailed explanation of each component of the experiment is provided later in this document.

2. GridSolve

Grid computing has been a widely researched discipline for the past decade, resulting in several competing grid computing environments. GridSolve is one such environment developed at the University of Tennessee by the Innovative Computing Laboratory. The GridSolve [2] (formerly NetSolve) environment consists of three entities: an Agent, Servers, and Clients. The servers of a GridSolve environment provide problem-solving services based on the libraries available on the host machines. The servers report their availability and a list of available software to the agent, which maintains the complete list of available servers and software. Clients then poll the agent when a solver is needed and contact one or more servers from the returned list. It is assumed that because the agent has complete knowledge of the GridSolve grid, it can provide the best scheduling decisions to the client. In such a system, however, the agent creates a single point of failure for the entire grid, which becomes completely disabled during agent downtime. Likewise, having a centralized repository can hinder scalability, as the agent will be required to handle an increasing number of server and client connections. Future versions of GridSolve will support multiple agents in attempt to address these limitations.

An additional limitation appears when users migrate from one GridSolve environment to another. In order to use their program a user will need some knowledge about the new environment to find the agent, or rely on their usual grid, which could be on the other side of the world or even completely disconnected from their current network. This is not a significant hurdle to the user, but it is a small limitation of the system. Finally such a centralized system implies the existence of some administrator or administrators to maintain the infrastructure, which may or may not be a reasonable requirement, depending on the environment.

3. Automatic Service Discovery

The convenience of service discovery on local area networks has appeared in various forms for several years. Both Microsoft Windows and Apple Macintosh have had at least basic discovery capabilities for the past several operating system versions. With advancements made in network bandwidth and service discovery algorithms it is becoming more feasible to integrate service discovery into an increasing number of applications. Most modern service discovery applications use one of two competing standards: Service Location Protocol (SLP) and Domain Name Service with Service Discovery (DNS-SD).

Service Location Protocol (SLP)

The Service Location Protocol [15] was proposed as an Internet draft in response to the proprietary and mutually exclusive service location protocols that already existed, such as Microsoft SMB, Novell NetWare, and AppleTalk. The SLP environment consists of three entities: the user agent, the service agent, and the directory agent. As the name implies, the user agent is the program or user searching for a service provider. The service agent is the actual service provider, who will reply to service requests if the host is capable of handling the request. The final entity is the directory agent, which is an optional entity that serves as a cache for service information on the network in order to reduce the networking overhead of the multi-cast system. When a program is started to provide a given service, the program registers itself with the local service agent, providing enough information to identify the service type and how clients should contact the program. Clients discover these services by polling the network for a needed service and waiting for responses. Several vendors currently provide support for SLP, including Sun Microsystems, Novell, and various network peripheral vendors. SLP seems to be a fairly natural fit with GridSolve, due to similarities in the system entities; however, due to the requirement that a service agent be started by a system administrator on each machine, it was not selected for this research. As more vendors adopt SLP in their products, this requirement may become less of a limitation.

Multicast Domain Name Service with Service Discovery

The Domain Name Service is the de facto standard for name resolution on the Internet and within most small networks. The idea of using the domain name service to also provide information about available services was first proposed by Stuart Cheshire, an employee of Apple Computer, Inc. in February 2004 as an Internet-Draft. Cheshire recognized the need to replace the proprietary AppleTalk protocol with a protocol that could be used on IP networks. By storing service information in DNS records and organizing them in a well thought-out manner, service discovery could be achieved without implementing a new system or modifying the existing DNS system. Unlike SLP, DNS-SD [9] can be used without making any changes to the configuration of a network and without the installation of any additional software.

As described so far, DNS-SD is only useful for long-lived services, such as web servers, mail servers, and printers. However when DNS-SD is combined with Multicast DNS (mDNS) [13], it becomes possible to use DNS-SD for short-lived services and on networks without an established infrastructure. Multicast DNS was designed as a way to issue DNS queries using broadcast messages on IP networks with minimal changes to the DNS standard. The intent of this technology is to address the need for a defined networking infrastructure as an increasing number of network-aware devices become commonplace. Since broadcast systems have an obvious potential of negatively affecting network bandwidth, caching and other considerations were made in the standard. The combination of mDNS and DNS-SD was chosen for this research, rather than SLP, primarily due to the convenience that the required service daemons can be started by an unprivileged user or even integrated into the user programs. The library used for service discovery is Howl [16], an open source implementation of Apple's Rendezvous initiative developed by Porchdog Software. A direct comparison between DNS-SD and SLP is found in Table 1 at the end of this section.

Table 1 Comparison between SLP and DNS-SD

Service Location Protocol	Multicast DNS with Service Discovery
• Flat or Hybrid Topology • Must be started by administrator • Service-Oriented • Limited to local network (research being performed to correct this) • Developed to replace competing technologies	• Flat Topology • User-level support • Protocol-Oriented • Limited to local network (not limited when using standard DNS) • Derived from established networking protocol

4. Automatic Service Discovery for GridSolve

The purpose of this document is to show that a computational grid can exist and perform reasonably using some automatic mechanism for organization, service discovery, and scheduling. Developing an entirely new grid environment for such an experiment would be a waste of resources, since multiple environments already exist, and would likely require an unreasonable length of time to complete. For that reason a derivation of GridSolve was developed that replaced the GridSolve agent with DNS-SD for service discovery. Such an environment is not optimal, nor does it completely fit the DNS-SD paradigm, but it does provide a stepping-stone for experimentation purposes.

In order to remove the GridSolve agent it was necessary to modify both the GridSolve server and client libraries. The server will now need to advertise its availability and resources publicly, rather than registering with the agent. Strictly following the DNS-SD paradigm, this would simply mean advertising that the server understands and implements some protocol (**_gridrpc._tcp** was selected to describe the protocol for the purpose of this research). Advertising the servers in this way, however, would require that the client search first for service providers and then poll the service providers for the needed software. Such a system would obviously be inefficient and provide less than desirable results. For this reason some liberties were taken when modifying the GridSolve server so that each piece of software is advertised to the network, rather than just the server itself. The notation selected to describe each service is **_gridrpc_SERVICE._tcp**, which is not too large a deviation from the DNS-SD model. Since each solver requires different parameters to run, it is reasonable to view each solver as a different protocol, making the above convention fair.

Changes made to the GridSolve client fall directly in line with the above server changes. Rather than polling the GridSolve agent when a problem solver is needed, the client polls the network via mDNS for the needed service. As new servers are found the list of known servers is updated and used just as the list provided from an agent. By modifying only the portion of the GridSolve code that populates the server list for the client and not the portion that actually communicates with the server, it is possible to

make fair comparisons between GridSolve and the experimental system. Since GridSolve fully supports the GridRPC [7] standard, making minimal changes also implies that the experimental system would likewise support the GridRPC standard.

It is important to note that the use of DNS-SD introduces a limitation to the experimental system that would need to be addressed by a production-quality environment. One of the factors driving the development of GridSolve is the need to support clients and servers which may be members of different NATs. Since the DNS-SD library being used relies on multicast networking, the experimental environment must exist on a single, multicast network. This limitation only exists due to the way the proof-of-concept environment was constructed and would be taken into consideration in a production environment.

Experimental Scheduling

In the current GridSolve system all scheduling decisions are made by the agent from its global knowledge of the environment. Since the centralized scheduling authority has been removed from the experimental environment, some new scheduling scheme needed to be devised. The scheduling decisions made by the GridSolve agent are based on the results of a benchmark run on each server once, the current reported workload of the server, and the complexity of the needed solver. Rather than reporting this information to an agent and allowing the agent to make the needed calculations, each server will report the result of this calculation as its *scheduling number*. When a client searches for servers to solve a given problem, it will receive back in the servers' responses the information needed to contact the servers and the servers' scheduling numbers. As the client populates its server list, it performs an insertion sort of the servers, sorting on the scheduling number, thus creating the same ordered list that the agent would return. Giving the servers the responsibility to calculate their scheduling numbers reduces the complexity of client-side scheduling to just an insertion sort. If a client has specific scheduling needs, it can schedule based on the raw scheduling data, but at the cost of increased complexity.

This scheme lends to a question of scheduler integrity. If each server is responsible for reporting its own scheduling results, a dubious server could report results to hijack client scheduling. Several points should be made in defense of the scheduler. The first point is that grid computing is generally defined with a level of trust among participants, implying that each server will behave properly. Grids run in untrusted environments should always utilize some other mechanism for establishing trust among participants. The second important point is that this integrity issue is also a part of the current GridSolve system, where servers can manipulate the workload or benchmark results to hijack the agent scheduler. So it can be said that the scheduling mechanism of the experimental system is no less secure than the current GridSolve.

5. Centralized and Decentralized Results

To evaluate the performance of the decentralized GridSolve system, three tests were devised. The first test evaluated the potential overhead of the two systems with a simple, single-machine grid. The second scenario involved evaluating the performance of programs that rely more on quantity of processors than quality of processors, such as is the case with embarrassingly parallel algorithms. The final case evaluated the performance of a processor-heavy problem, which is concerned with the proper scheduling of a job to a suitable processor. The specifics of each test appear in the sections to follow.

Overhead Performance

To compare the scheduling overhead of decentralized system versus the traditional system, a single-node grid was configured for each. Therefore the automatic-discovery system consisted of a single server, while the GridSolve system consisted of one server and one agent on the same computer. Arranging the computers in this way allowed minimal network communication for service discovery. The purpose of this test was simply to determine if an obvious difference in overhead existed between the two systems.

Figure 1 shows the average time needed to solve a simple matrix multiply on the trivial grid. The matrix sizes ranged from 10x10 to 500x500 elements in increments of ten. Each point on the graph is the mean of ten experimental program executions. As can be seen in Figure 1, the performance of the two systems was virtually identical in this trivial case.

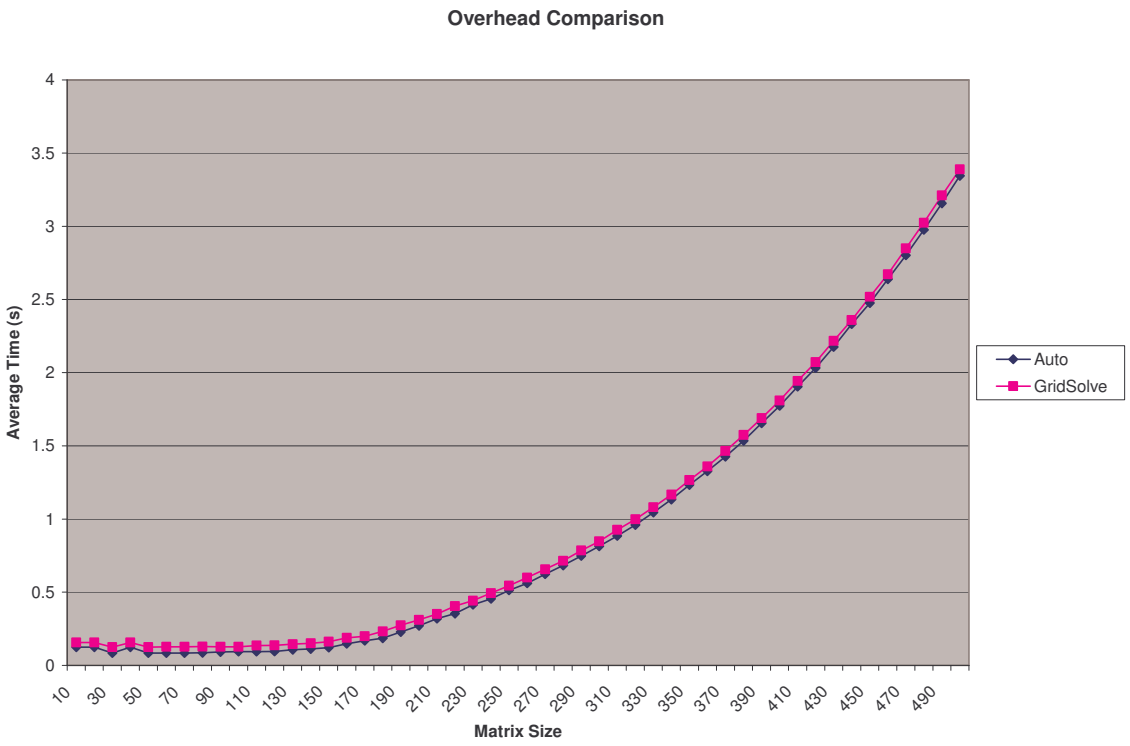

Figure 1 Overhead comparison for trivial grid instance.

Farming Comparison

A more interesting comparison between the centralized and decentralized systems is a program that requires several servers in order to complete. Many embarrassingly parallel applications are more dependent on using a large number of servers to execute on parts of the data than finding the best possible server to execute the code. For this test twenty machines were chosen, ten of which were 500 Mhz UltraSparc machines and ten were 400 Mhz UltraSparc machines. The problem that would be solved needed to be trivial enough that machine quality would be unimportant, so as to recreate the embarrassingly parallel case. The selected problem received a double-precision value **d** and returned **(d+1)**, which should be suitably trivial.

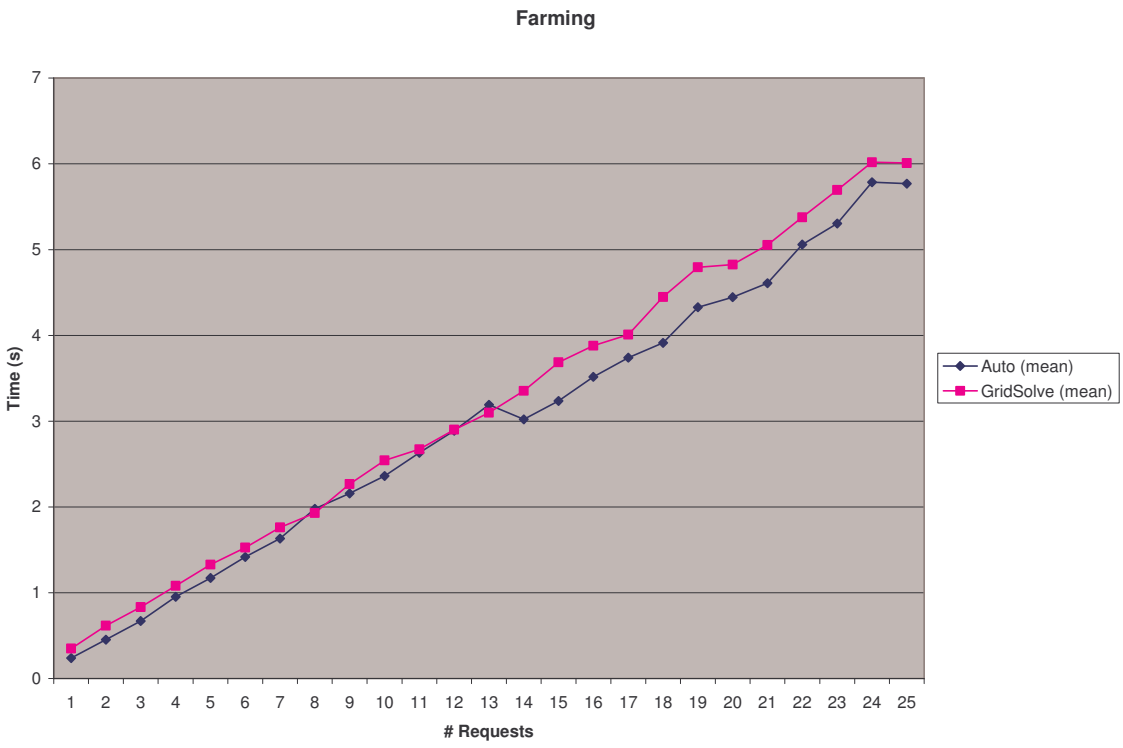

Figure 2 Comparison of task farming requests.

Figure 2 shows the result of running an increasing number of instances of this solver. Each point on the graph represents the mean of ten program executions, ranging in request size from one to twenty-five in increments of one. It should be noted that the test was run beyond the number of available servers to compare how each system performs in such an instance. The sudden change in performance by the automatic discovery system was due to a memory bug in the mDNS Responder daemon, which consistently necessitated a restart between 12 and 13 requests. This memory bug has been reported to the Howl developers. The standard deviations of each data point are provided in Figure 3.

Processor Limited Comparison

Although some applications may benefit from a large number of processors, regardless of processor performance, many are bounded by the processor performance and therefore require more intelligent scheduling. To examine performance in this

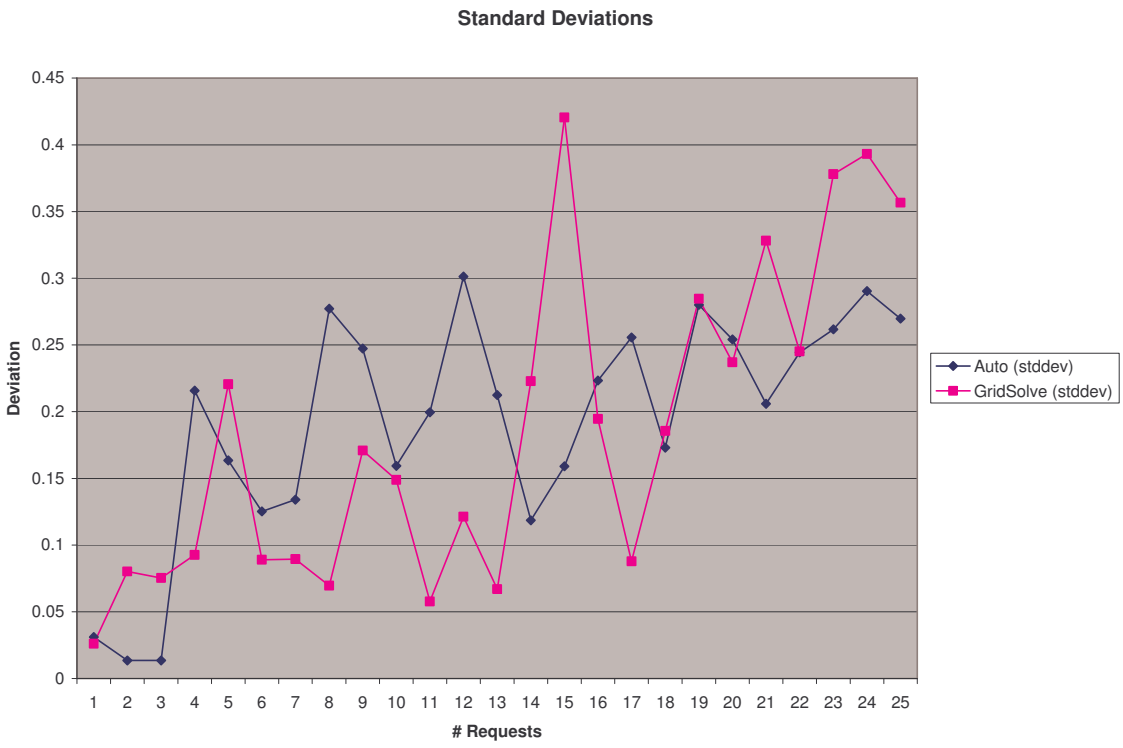

Figure 3 Standard Deviation of task farming request timings.

instance, a standard DGEMM matrix multiplication was selected. A reference implementation of the BLAS [14] was selected so that only the server hardware needed to be considered and not the library implementation, which would greatly complicate scheduling. The grid used for this test included the twenty machines listed previously and also ten machines consisting of two 933Mhz Pentium III processors each. These machines were included so that an obvious distinction could be made between the best and worst processors to solve the matrix multiplication. A dgemm benchmark comparing the performance of the best and worst of these machines is shown in Figure 6 in the Appendix, which shows the faster machine performing as much as three times better than the slower.

Since the automatic discovery system schedules based on responses made in a broadcast network, three different service discovery situations were considered. The first program execution was done on a machine that is geographically separated from the grid machines, which means that it could possibly be on a different network switch than each

of the machines. The second program execution placed the client among the slowest machines on the grid, which theoretically places it on the same network switch as the slowest machines. The third case placed the client among the fastest machines on the grid and theoretically on the same network switch. These distinctions were made to evaluate whether the automatic system would simply choose machines that replied to service discovery requests more quickly or whether it would properly schedule the requests. The matrix multiplies were requested for matrix sizes 10x10 through 500x500 in increments of ten and each point represents the mean of ten executions on all three client systems. The results of each of the individual cases are shown in Figures 7-9 in the Appendix. Figure 4 shows the mean results of both automatic service discovery and GridSolve, and Figure 5 shows the mean standard deviations.

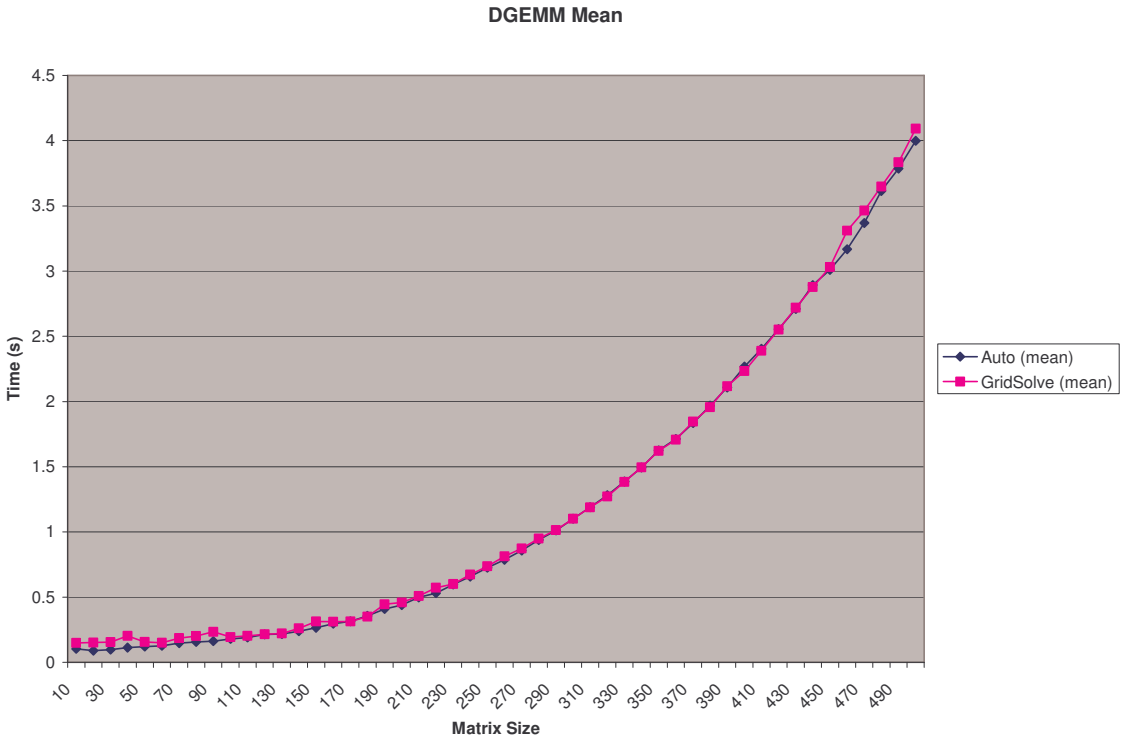

Figure 4 DGEMM mean time comparison.

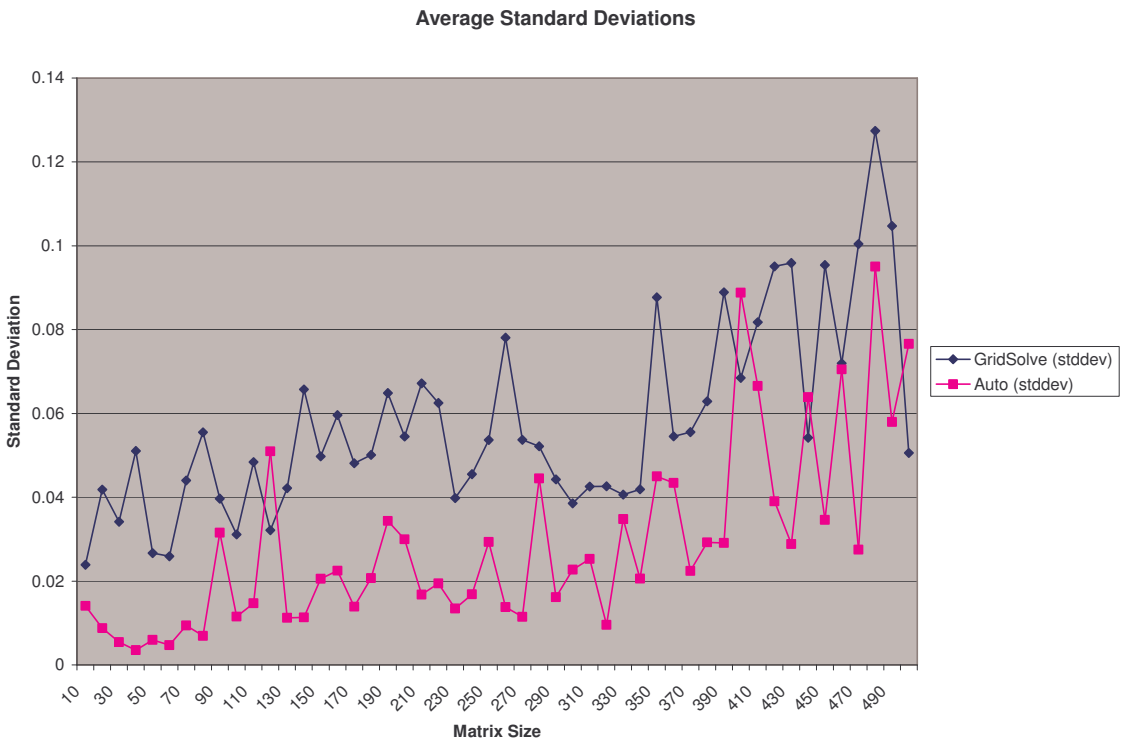

Figure 5 DGEMM mean standard deviations.

Results Summary

In each of the above experiments the two grid environments evaluated consistently returned competitive, and in some cases, virtually indistinguishable results. Although the aforementioned memory bug in Howl was obvious in the results of the farming experiment, there is no evidence that the results from the other experiments were also affected. The positive results from each experiment show that the service discovery changes made to GridSolve had little or no effect on the time to run each of the test programs. It is therefore reasonable to say that the decentralized system is a feasible alternative to the traditional centralized repository approach to grid computing using the GridSolve system.

6. Future Directions

The use of a centralized resource database, such as the GridSolve agent has given grid computing environments a simple means for resource discovery and scheduling, accelerating the research in this field. Such systems view grid resources as closely organized nodes on a connecting network, which is an accurate viewpoint, but perhaps a bit limited. By looking at grid computing resources from a different perspective, some interesting parallels exist between grid computing and basic networking. When a router joins a network it begins advertising itself as a viable route to some group of network addresses. Similarly, grid computing servers advertise that they are a viable choice for solving some set of problems. On IP networks each host must have a unique address; likewise it is crucial that solvers can be uniquely identified so that the proper parameters will be exchanged. Traditional networking handles the need for unique addressing by using some authority to assign addresses, either statically or dynamically. An emerging technology often used with the service discovery libraries explained earlier is Zero-Configuration Networking (zeroconf) [17], which uses address discovery and conflict-detection algorithms to handle assigning addresses on networks without a centralized infrastructure. Such a technology could also be used in place of the GridSolve agent for maintaining unique solvers on the grid.

Since multiple routes can exist to a given entity on the network, routers provide some metric for evaluating the quality of their routes. Likewise, grid computing servers should provide information to evaluate their viability for solving a problem. From this perspective, a grid server is essentially a router for a network of solvers. Thus it would seem that a broadcast protocol for grid computing resources is little more than a network routing protocol and could benefit from lessons learned from years of networking research. Just as an efficient routing protocol has little effect on the throughput of the network, a smart broadcast or multicast protocol for grid computing should be able to avoid negatively affecting network activity. Without careful planning, however, such a flat system could suffer from severe scalability issues.

Since a purely flat network of grid nodes has the potential of flooding the underlying network with unwanted activity and reducing scalability, some ideas may be borrowed from filesharing network algorithms to create a hybrid centralized-decentralized topology. It is important to realize that the scheduling needs of filesharing and grid computing differ greatly, but the two systems share common needs for scope and reliability. Some peer-to-peer (p2p) filesharing networks use the notion of nodes and supernodes, where all entities on the network are nodes and have basic knowledge about themselves and select neighbors and supernodes have a more global knowledge of the network. On such networks, supernodes are usually dynamically assigned by some protocol, often an election protocol. To put such supernodes into GridSolve terminology, a supernode is essentially a GridSolve agent, but is elected, rather than appointed to be the agent. Using supernodes can further reduce the networking overhead inherent in a broadcast/multi-cast system. Supernodes will keep an up-to-date registry of some number of, or perhaps all members of the network, while nodes will simply keep information about themselves and a short list of other nodes that have recently been discovered.

The addition of some method of automatic organization and discovery into the GridSolve environment could greatly enhance system reliability. By borrowing the idea of hybrid topologies from peer-to-peer filesharing networks, GridSolve would be able to benefit from the decentralized paradigm while still practicing responsible networking. A comparison of network topologies in peer-to-peer networks is available in [6]. Such an environment would not only provide increased reliability but also increased flexibility and ease of use in highly voluntary environments or environments where no administrator is available to maintain the grid infrastructure. The exact details of migrating from GridSolve's current centralized environment to a less centralized or completely decentralized environment would make interesting future research.

7. Related Works

Other grid computing researchers are beginning to realize the potential of taking a decentralized approach to grid organization. Recent versions of the Globus [10] Monitoring and Discovery System (MDS) have switched from a centralized to a decentralized resource directory. Other groups have also combined the Globus Toolkit 3 implementation of OGSA [11] with JXTA [12] from Sun Microsystems to study an implementation of OGSA using peer-to-peer algorithms for service discovery in [1] and [8]. [3] and [5] both conclude that decentralized resource discovery shows promise from the perspective of both administration and performance. It has even been suggested that peer-to-peer filesharing and grid computing will one day converge into a common architecture [4]. To date, however, no similar research has been performed on the GridSolve/NetSolve framework.

8. Conclusion

In recent years, progress has been made in the area of grid computing, resulting in several mature computing environments. In general these environments have been designed with a centralized resource database, which allows easy scheduling and can provide a simple means for grid membership controls. The research presented in this document shows, however, that similar results can be achieved with a decentralized environment without the potential problems associated with relying on a single authority. With proper thought, a fully decentralized or hybrid topology can replace the centralized database with a reliable and equally efficient form of resource discovery and scheduling.

Bibliography

[1] Amin, K., von Laszewski, G., Mikler A., *Toward an Architecture for Ad Hoc Grids*, Proceedings of the IEEE 12th International Conference on Advanced Computing and Communications (ADCOM 2004). December 2004, Ahmedabad, India.

[2] Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Fabianek, C., Hiroyasu, T., Meek, E., Miller, M., Sagi, K., Seymour, K., Shi, Z., Vadhiyar, S., *User's Guide to NetSolve V2.0,* University of Tennessee, 2003, Innovative Computing Laboratory Technical Report, October, Knoxville, TN.

[3] Iamnitchi, A., Foster, I., Nurmi, D., *A Peer-to-Peer Approach to Resource Discovery in Grid Environments*, Proceedings of the 11th Symposium on High Performance Distributed Computing, Edinburgh, UK, August 2002.

[4] Iamnitchi, A., Foster, I., *On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing*, 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), Pages 118-128, 2003.

[5] Iamnitchi, A., Foster, I., *On Fully Decentralized Resource Discovery in Grid Environments*, Proceedings of the Second International Workshop on Grid Computing, Pages 51-52, 2001.

[6] Karwaczyński, P. and Kwiatkowski J., *Analysis of Overlay Network Impact on Dependability*, Proceedings of the 38th Hawaii International Conference on System Sciences -2005.

[7] Seymour, K., Nakada, H., Matsuika,S., Dongarra, J., Lee, C., Casanova, H., *Overview of GridRPC: A Remote Procedure Call API for Grid Computing*, Proceedings of the Third International Workshop on Grid Computing, 2002.

[8] Smith, M., Friese, T., Freisleben, B., *Towards a Service-Oriented Ad Hoc Grid*, Third International Workshop on Parallel and Distributed Computing, 2004.

[9] website: http://www.dns-sd.org/

[10] website: http://www.globus.org/

[11] website: http://www.globus.org/ogsa/

[12] website: http://www.jxta.org/

[13] website: http://www.multicastdns.org/

[14] website: http://www.netlib.org/blas/

[15] website: http://www.openslp.org/

[16] website: http://www.porchdogsoft.com/products/howl/

[17] website: http://www.zeroconf.org/

Appendix

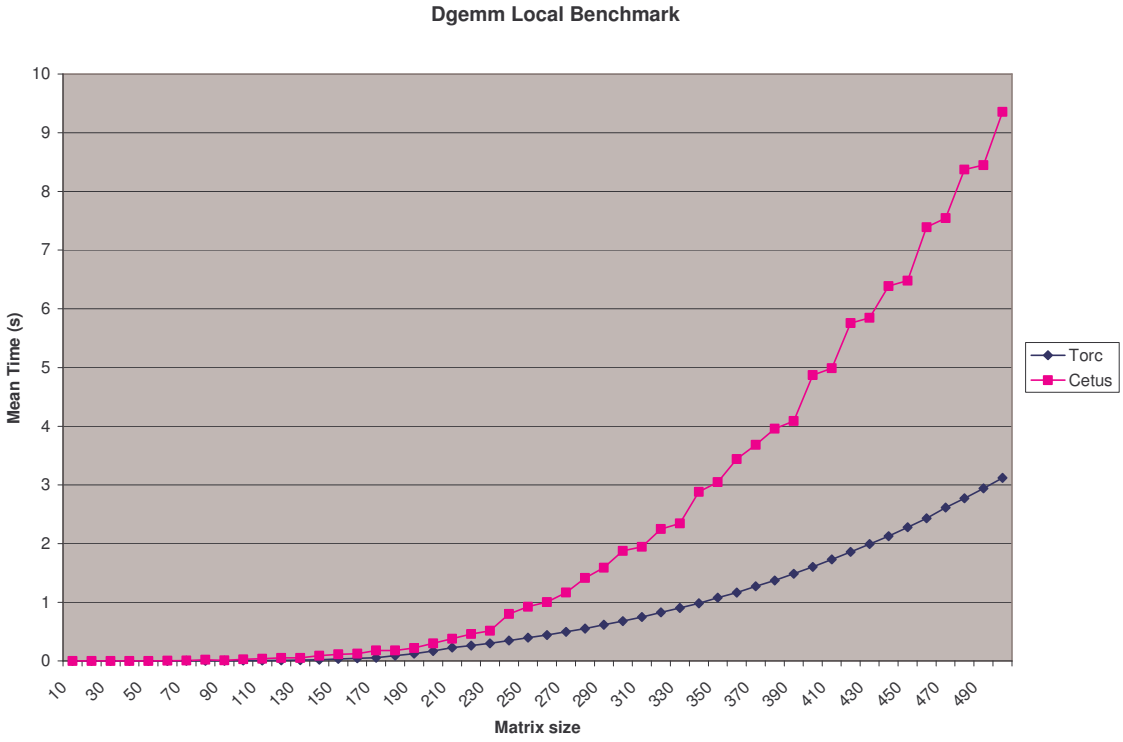

Figure 6 DGEMM performance on Torc and Cetus workstations.

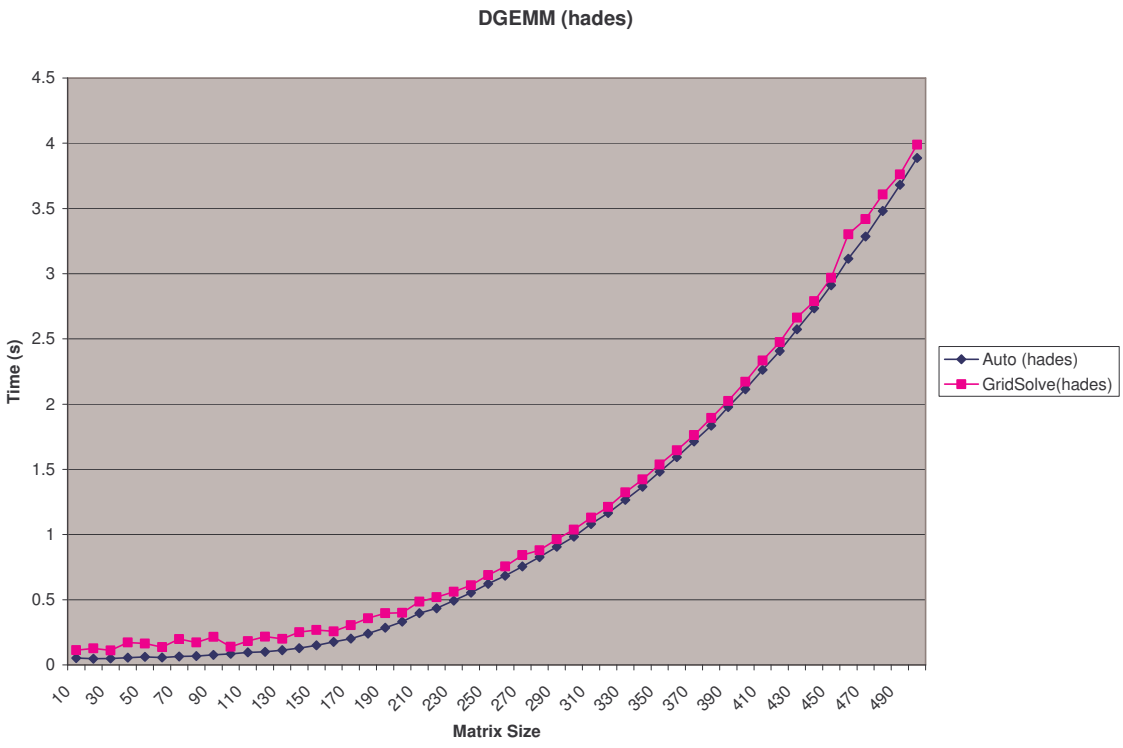

Figure 7 DGEMM time-to-solution from geographically separate client.

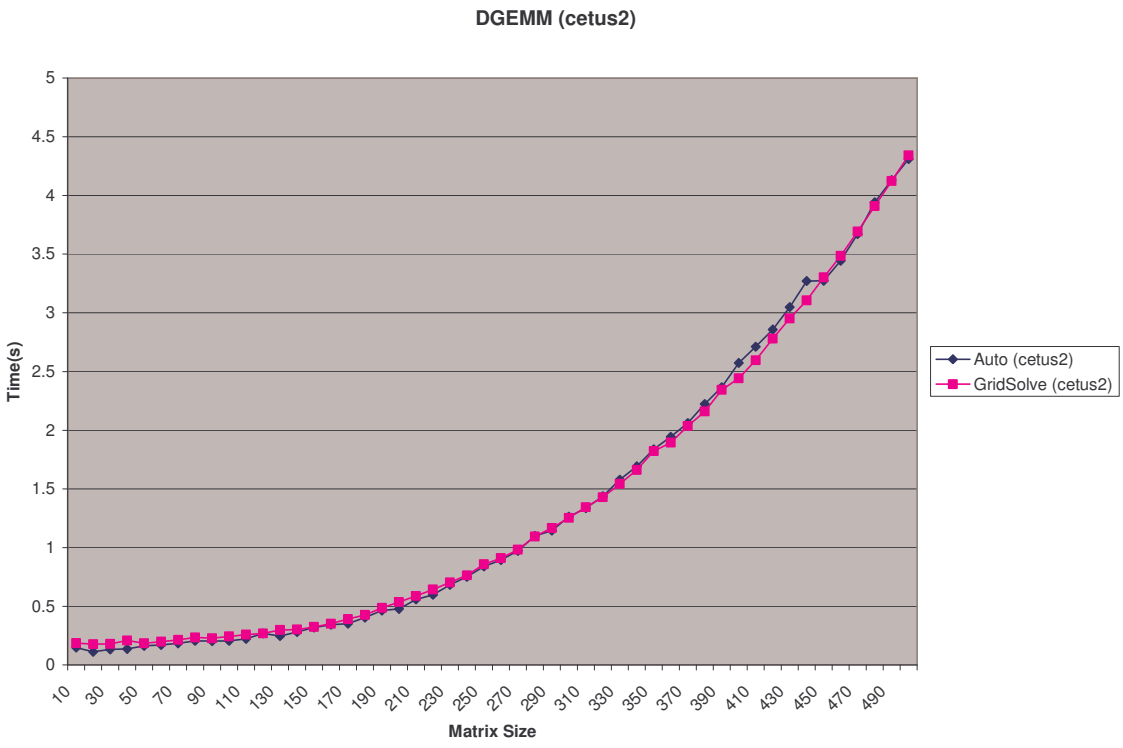

Figure 8 DGEMM time-to-solution for client located among slow servers.

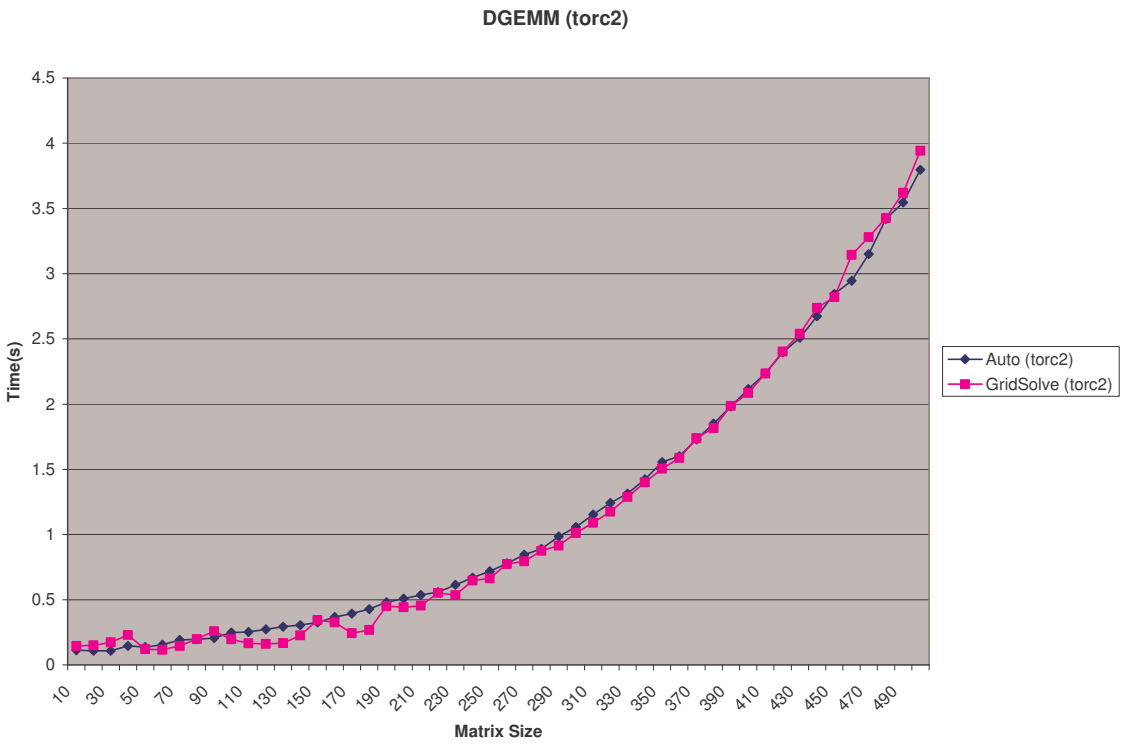

Figure 9 DGEMM time-to-solution for client located among fast servers.

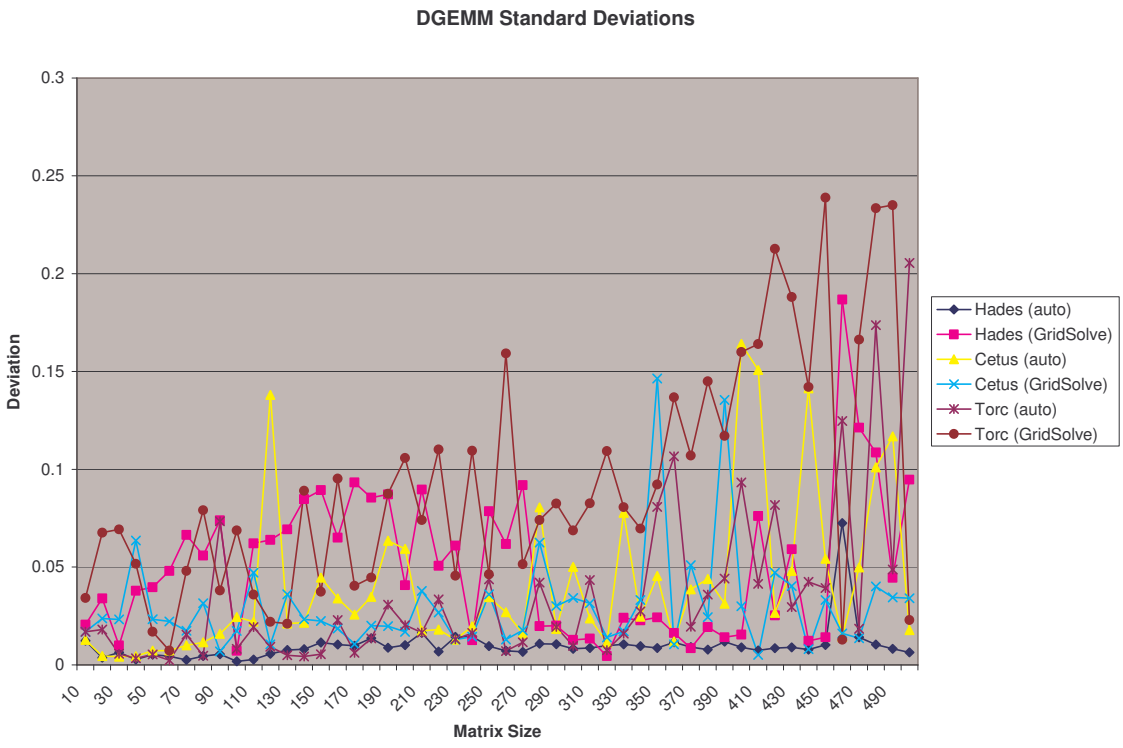

Figure 10 Standard deviations for each DGEMM test case.

Vita

Jeff Larkin received a Bachelor of Science degree in Computer Science from Furman University in 2002. While at Furman he performed research in 3D visualization of satellite space weather data and cluster computing. His research in cluster computing won him a first place award from the ACM Mid-Southeastern Conference in 2001 and second place award from the ACM International Conference in 2002. Jeff began attending the University of Tennessee in Fall 2002 and began working as a research assistant in the Innovative Computing Laboratory in January 2003. While working as a research assistant he worked on several research projects in grid computing.