On the Convergence of Computational and Data Grids

Dorian C. Arnold* Sathish S. Vadhiyar* Jack Dongarra*

Keywords: Computational Grid, Data Grid, Problem Solving Environments, Distributed Computing, Heterogeneous Network Computing.

Abstract

Great advances in high-performance computing have given rise to scientific applications that place large demands on software and hardware infrastructures for both computational and data services. With these trends the necessity has emerged for distributed systems developers that once distinguished between these elements to acknowledge that indeed computational and data services are tightly coupled and need to be addressed simultaneously. In this article, we compile and discuss several strategies and techniques that can be used to help resolve some of the issues, like co-scheduling and co-allocation of computational and data services, dynamic storage capabilities, and quality-of-service. We present our interactions with a distributed computing system, NetSolve, and a Grid service utility, IBP, as a case study of how some of these techniques can be effectively deployed and also offer experimental evidence from early prototypes that validate our motivation and direction.

1 Introduction

In recent years, computational scientists have been able to make tremendous advances in their research, studying increasingly larger and more complex models within their scientific domain. Simulations ranging from that of the electromagnetic field of a defibrillator in a virtual human to warfare scenarios with tens of thousands of components interacting with each other are now possible due in large part to significant technological advancements in parallel machine architectures and the constant, rapid increase of microprocessor performance. Scientists have been liberated with the emergence of boundary-less collaborations of geographically dispersed researchers, computational hardware, software and data. This vision of global scientific computing poses many challenges to computer scientists, whose role it is to provide tools that effectively and efficiently solve the problems of working with very large datasets – tens or hundreds of gigabytes – on this globally distributed computational fabric that has come to be known as the Grid [13].

Grid Computing systems (defined and discussed in section 3) have mainly focused on the issue of harnessing computational cycles and seamlessly bringing them to effective use by scientists and researchers. Until recently, little attention has been focused on techniques that exploit the relative cheapness of storage, as compared to network fabrics, to achieve high levels of computational throughput in the presence of large datasets. Moreover, the Grid community is progressing toward the realization that there can no longer be a decoupling of data and computational services. In effect, there is a single, compound question that needs to be answered time and time again, "Where is the data, and where will it be processed?" From this question stem all the other issues that need to be resolved like data size, ownership of resources, choices in resources, etc. The richness of the issues to be addressed will lead to complex solutions, some more effective than others.

It is our belief that an infrastructure designed to efficiently and effortlessly support complex interactions between computational and data resources can bear

*Computer Science Department, University of Tennessee, Knoxville, TN 37996-3450

remarkable impact and improvement on the way scientists, engineers, and even the business community, use computers. Our hope is that this article will help to motivate and encourage research avenues pointed in this direction. In this article, we explore various techniques for accommodating large, distributed datasets on the Grid. We begin by mentioning related efforts in section 2. We then continue our discussion by providing a more precise definition of the Grid in section 3, after which we discuss various approaches to addressing the cooperation of data management and computations on the Grid in section 4. This section discusses several strategies for the interoperation of data and computations on the Grid. We then present our case study, NetSolve and IBP, in section 5 along with experimental results that validate our research direction. We spend some time discussing algorithms and techniques and end with a discussion of a concrete environment where these strategies are being employed.

2 Related Work

We cannot possibly mention every system or project that implements some form of the various techniques described below. The techniques are quite numerous and are motivated from various computing areas including Grid Computing, Web/Internet Programming and Database Management. Instead, we make note of a few projects of particular interest that we have studied while investigating our ideas. Throughout the discussion, we also point to other on-going work in specific areas.

The Grid Physics Network or GriPhyN [1] is an expansive collaboration of many scientists from a handful of institutions. They aim to research, prototype and then develop a production environment targeted primarily at four successful physics projects. Their research agenda is to "enable groups of scientists distributed worldwide to harness Petascale processing, communication, and data resources to transform raw experimental data into scientific discoveries." By the year 2005, the project envisions a computational environment that can accommodate on the order of 5000 terabytes of data per year at a sustained aggregate access rate around 100 gigabytes per second.

The Data Grid [8] aims at identifying and satisfying the requirements and components of an integrating architecture that efficiently manages large data repositories, maintains high network transfer rates and schedules supercomputer-class computations on geographically distributed resources. A primary focus of the project is replica management and selection.

Much of our discussion is also motivated by various techniques that have been established in Web technologies for caching, cache cooperation-operation, content distribution networks, mobile agents, etc. We discuss the applicability of these services and strategies to the Grid (which from this point we will use to refer not to the traditional Computational Grids, but to a fabric that is meant to accommodate both computational and data services.)

3 Grid Overview

Before we initiate the discussion of techniques that consolidate computational and data services, we define more precisely these target infrastructures. The term Grid was coined from the electrical power grid where electrical power is uniformly interfaced and ubiquitously available from distributed sources. Electrical power is easily obtained by plugging into electrical sockets that outlet electricity usable for limitless services from powering toasters to supercomputers. Computer scientist envision an analogous infrastructure that will be able to uniformly and seamlessly channel computational services to clients who "plug in" to the Grid.

Figure 1 layers the services and protocols that typically comprise Grid-enabling infrastructures. The major functionalities of Grid systems occur in the middle layer and are often referred to as middleware components. Within this layer are the technologies that allow for resource discovery, resource scheduling and allocation, fault-tolerance, security mechanisms, and load-balancing. This layer also houses the resource management and information services. Above the middleware is a thin layer that typically implements an interface and protocols allowing ap-

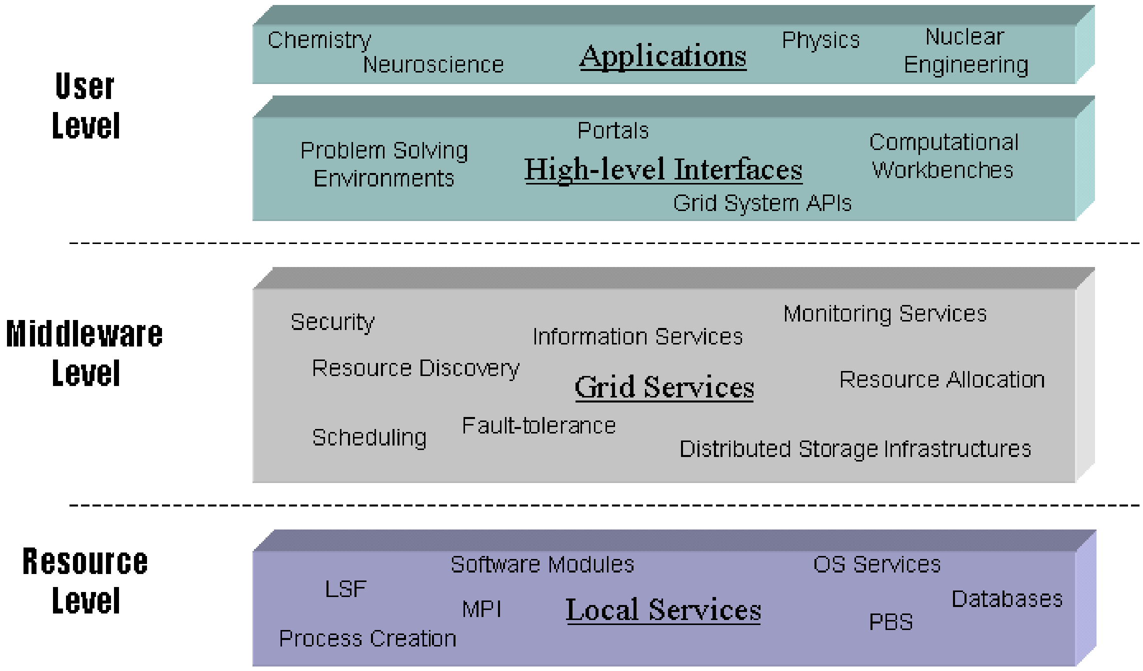

Figure 1: A layered representation of the major components of typical Grid Computing systems.

plications and users to access the middleware services. The layer below the middleware layer is the resource layer that typically provides local (usually system level) services that render computational resources like CPU cycles, network infrastructure, storage, software, etc. A popular model for Grid systems is the client-server model where daemon processes act as access points to either middleware or resource level services, and client tools or application programming interfaces are used to instantiate service requests to the system. Globus [15], Legion [16], Condor [17], and NetSolve [7] represent a few of the more visible projects in Grid computing. This handful of projects help to show the richness of Grid research as each project has a unique perspective on how resources should be represented, managed and accessed.

4 Data on the Grid

As stated above, the goal of this paper is to identify technologies that can be applied to Grid infrastructures like those described in this section. With this in mind, we now discuss various strategies that will assist in the realization of a Grid fabric that can accommodate high-performance computing applications with large data requirements.

4.1 Caching

Caching refers to the process of maintaining data as persistent at some convenient location with the hope that there will be subsequent use of this data while it is being cached. Current web technology accomplishes this caching via the use of a proxy cache server. This server sits between the component requesting the data, the client, and the data provider's server intercepting client requests. Objects found in the cache are returned to the user, while if an object is not found in the cache, the proxy retrieves the object from the originating server on behalf of the user, returning it to the user and possibly depositing it into the cache at the same time. The network literature distinguishes two main types of caching, demand-side and supply-side. We discuss how these and other caching schemes can be employed in the Grid.

4.1.1 Demand-side caching

Demand-side caching refers to the traditional caching mechanism where the cache is an agent of the client application or data consumer. Data is stored on (or near) the client host. Demand-side caches, or any cache for that matter, can only be helpful to the extent that they are able to accurately model the usage pattern of data. This is a difficult task. In fact, a simple cache content replacement strategy like "least recently used" is often a good choice. In the Grid, there is opportunity to reduce some of the guesswork by doing some analysis of requests for Grid services. In [2], a strategy is suggested of batching together requests from a client and doing parametric analysis to construct a data-dependence graph where the nodes are computational modules and the arcs are data dependencies. This graph can then be used to place data in storage locales or caches that will minimize overall network traffic amongst the client and potentially many server components used to service the request. Though this may seem like supply-side caching (see next section) as the client pushes data near the server that consumes it, the authors choose to view the entire client-server system as the consumer of data supplied by some third party data source. This strategy entails computational overhead for the analysis phases and yet may produce sub-optimal results, or become intractable as the number of components and parameters involved becomes large. Another method is for the human user of the application to manually pre-stage data at or near server resources and dictate to the Grid system that those are the resources of choice. This strategy can yield very good results, but forces the user to be bothered with decisions that he may prefer to leave to the system being employed.

An important aspect of the demand-side caching is the placement of the cache itself. In the World Wide Web, where network caches are most commonly employed, proxy cache servers are often deployed at the edges of a network (i.e. at company or institutional gateway/firewall hosts) to serve a large number of internal users. Web browsers also have the capability to cache to the local file system on a per-user basis. In Grid collaborations where it is atypical for more than one or two local researchers to be interested in a particular dataset at the same time, it is not immediately evident how effective community-based caching would be. Individual users, on the other hand, could experience tremendous benefits by the bandwidth savings that result in improved response time and increased availability of data objects.

4.1.2 Supply-side caching

In supply-side caching, the cache is an agent of the supplier, not the client. Supply-side caches are increasing in popularity, as Web sites receive increasing traffic, and the trend toward outsourcing content distribution continues to develop. A large market demand has evolved for content distribution networks that cache or replicate data across a wide area according to dictating demands. Primarily in content distribution networks, data access statistics are collected or modeled and used to distinguish usage patterns and determine logistics that will yield data consumers better accessibility (primarily quantified as data bandwidth.) The Data Grid project mentioned in section 2 offers a notion of user-asserted replication management. The user also asserts replication selection – presumably based on network performance measures. Another strategy might be an "on-demand" supply-side cache where the user asserts nothing more than a need for a specific data item. A storage service utility that manages distributed storage depots is able to automate the replication of the appropriate data to the most appropriate storage server. A system like this can be implemented on top of the replication management and selection services provided by the Data Grid project. The motivation behind supply-side caching, whether dynamic or static in implementation, is to have content providers put data as far toward the edges of the network as possible so that anyone desiring access will have high degrees of accessibility.

4.1.3 Cache-cooperation

In some situations, replication of data among the various individual caches is not desired due to constraints in the storage capacity of the caches, data sizes, network traffic etc. In these situations, one may consider migrating the data between caches belonging to different domains. Intelligence can be added to the caches, and caches can be made to operate in a cooperative manner, where each cache "knows" the location of the data that previously existed in it. Thus data accesses directed to the original cache will be redirected to the new cache where the data currently resides. This situation is similar to the Personal Communication Service (PCS) environment of wireless telephony where one can imagine the different domains as the individual cells, the data as the mobile phones and the intelligent caches as the control towers.

Inter-cache communication can also improve system scalability and availability, and also allows for load balancing. In [18], several protocols for inter-cache communication are discussed. ICP, one of the more mature of these protocols, allows caches to query each other to determine the best location from which to retrieve requested objects. Microsoft's CARP [22] uses a deterministic hashing scheme to identify where requested information is located. This method avoids the overhead and scalability issues associated with inter-cache communication. Finally, we mention the notion of cache-digests that are also used to reduce inter-cache communications by summarizing the objects contained in peer caches. An example of cache-digests is implemented in the Squid Project [10] where ICP was also deployed and investigated.

Within the context of the Grid, caching technologies can be leveraged in a wide range of applications from scientific applications with large datasets to collaborations where relevant data can be distributed even across continents. As the notion of the Grid becomes more prevalent and Grid systems become ubiquitous, community-based caching schemes will become more useful and practical.

4.2 Active Data Repositories

In [12], it is pointed out that an important characteristic of many scientific applications that make use of large data sets is that such applications only wish to retrieve a subset of the dataset. Additionally, [1] makes the claim that (in high-energy physics) over 90% of data access is to derived data – data that can be derived from the reproducible, raw experimental data. This has motivated research in active data repositories where users assert operations upon stored datasets to dynamically create or extract only data that is relevant to their particular analysis. [12] introduces the Active Data Repository (ADR) an infrastructure for building databases that enable the storage, retrieval and processing of multidimensional datasets. It does this by providing run-time support for an object-relational database management system for managing scientific multidimensional datasets and applications that make use of these datasets. Within the Grid, we can employ similar strategies to create storage or database infrastructures that allow users to fine-tune data queries at the storage depot. This is accomplished by executing functional modules on datasets thereby preventing the transmission of unnecessarily large datasets only to filter them elsewhere for the possibly small subsets of interest. Since the ultimate goal is increasing throughput, and not just reducing network traffic, performance measures of the network as well as those of the computational units at the client and storage facility should be considered to see whether it is more efficient to transmit a large, unfiltered dataset and refine it locally or to do the filtration remotely in order to reduce network latencies.

4.3 Mobile Agents

A mobile agent [9] is a piece of executable software that is dispatched from a client computer to a remote server for execution. This is somewhat analogous to the concept of the active data repository in the previous section, and the differences lie perhaps in philosophy and spirit. The idea behind active data repositories is mainly to execute a refinement or filtration process on data that is then transported to some other resource component for additional computational processing. In Grid Computing, a mobile agent's job is to dispatch a module that will actually perform a computational analysis remotely, quite similar to remote job submission. It inverses the procedure of sending data to a compute server; it sends computations to a data server. Security concerns are obvious when server or repository hosts accept code for execution on behalf of clients.

4.4 Co-scheduling

In the literature, the term co-scheduling usually defines the scheduling of a many-to-one or many-to-many relationship of requests for computational services to computational units. Here we use the term to mean the cooperative scheduling of the computational and data network demands of a request for a computational service. The Condor system [4] suggests a `matchmaker` service that matches resource requests with resource offerings. In Condor, a resource request includes size and location information of a requesting application's executable. Matchmaker then uses network resource limits and topology information in algorithms like "first-fit" to allocate CPU and bandwidth to requests. [4] also discusses provisions for priority-based allocation of resources to increase CPU utilization by prioritizing applications with small network demands. The NetSolve project is currently active in the research of co-scheduling data and computations and a discussion of this thrust is in section 5. This area of Grid scheduling encompasses a large body of current and future research endeavors to explore the measurement and prediction of resource performance for scheduling purposes as exemplified by the application level scheduler project, AppLeS [5].

4.5 Reservation Systems

In conjunction with the scheduling of network-based applications, researchers also investigate the effects that immediate and advance reservation policies can have on quality of service (QoS) determination. Immediate reservations are made at the establishment of a service request where available resources are allocated to meet user demands. Advance reservations allow users to "book" resources in advance yielding a higher expectation of getting the requested resources. There has been work done in reservation schemes for computational resources (e.g. GARA [14]), and the network community has extensively studied bandwidth reservation (e.g. RSVP and ST-2 [19]) but no integrating Grid architecture exists that combines both these approaches. When the reservation of both data and computational services are combined, the value added to the Grid is the ability to use a `Threshold QoS` that states a user's worst-case measures of acceptable resource performance or a stronger `Guaranteed QoS` where the system guarantees a performance value to the user and commits itself to provide that performance at all costs. Grid reservation systems must also appropriately increase the efficiency and predictably of performance of Grid applications while keeping starvation to a minimum, utilization to a maximum and render deadlock nonexistent.

5 Enabling NetSolve with Data Grid Services

We now transition from abstractions and theoretical methodologies to show how some of these techniques might be employed in an actual Grid System. We present the NetSolve system and its interactions with the Internet Backplane Protocol (IBP) to create an infrastructure that enables the efficient and robust servicing of distributed computational requests with large data requirements.

5.1 NetSolve Overview

The NetSolve project at the University of Tennessee provides remote access to computational resources, both hardware and software. The major components of the NetSolve system are:

- The NetSolve `agent`, an information service that maintains a database of NetSolve resources along with their capabilities (hardware performance and allocated software) and dynamic usage statistics in order to allocate server resources for client requests.

- The NetSolve `server`, a networked resource that serves up computational hardware and software resources, and
- The NetSolve `client` libraries, that allow users to instrument their application code with requests for remote computational services.

Figure 2 shows the infrastructure of the NetSolve system and its relation to the applications that use it. The shaded parts of the figure represent the NetSolve system. It can be seen that NetSolve acts as a glue layer that brings the application or user together with the hardware and/or software needed. The reader is encouraged to compare this diagram to figure 1 that shows a generic Grid architecture. Further documentation along with source code for the full product are available at http://icl.cs.utk.edu/netsolve.

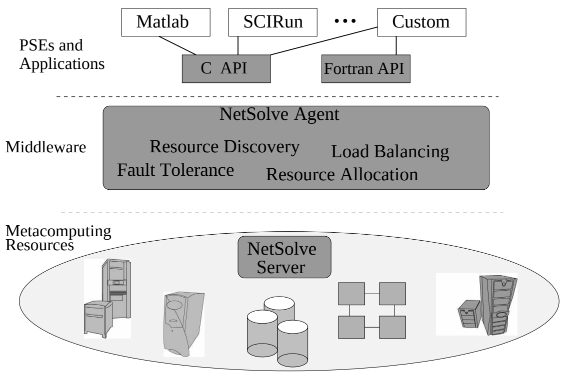

Figure 2: An architectural overview of the NetSolve system.

5.2 The Internet Backplane Protocol

IBP [21] is a Distributed Storage Infrastructure (DSI) for managing and using remote storage. This Grid service tool's motivation is to support logistical networking in large scale, distributed systems and applications, like NetSolve. IBP provides mechanisms for using distributed storage for logistical purposes.

By providing a uniform, application-independent interface to storage in the network, IBP makes it possible for applications of all kinds to use logistical networking to exploit data locality and more effectively manage buffer resources. It allows applications that need to manage distributed state to benefit from standardization, interoperability, and scalability.

IBP storage servers run a daemon process that interacts with storage client applications that have integrated IBP's API. Clients use this API to store and manage data for later retrieval (not necessarily from the same host). IBP uses a capability or handle to encapsulate remote data location and accessibility restrictions.

5.3 The NetSolve/IBP Integration

The motivation for this integration was to enhance the NetSolve Grid Computing System with some of the features described in section 4, mainly caching. Our hope is to eventually implement a system like that depicted in figure 3 where globally distributed caches cooperate to move data near consuming resources. We paid careful attention to make sure that our design, implementation and interface was not dependent upon the underlying DSI. We have intentions to experiment with other DSIs like GASS [6], but IBP is the system with which we first choose to work. Below, we describe the implementation, interface and results of our work where references to DSI specifically mean IBP facilities.

5.3.1 The interface

We extended the NetSolve API with a set of functions to create, destroy, open, close, read and write DSI storage. We modeled the API after the 'C' STDIO library. However, we did tune the read and write calls to take advantage of NetSolve system-specific characteristics. Though not an object-oriented system, NetSolve maintains all its components in object data-structures. Accordingly, NetSolve views data

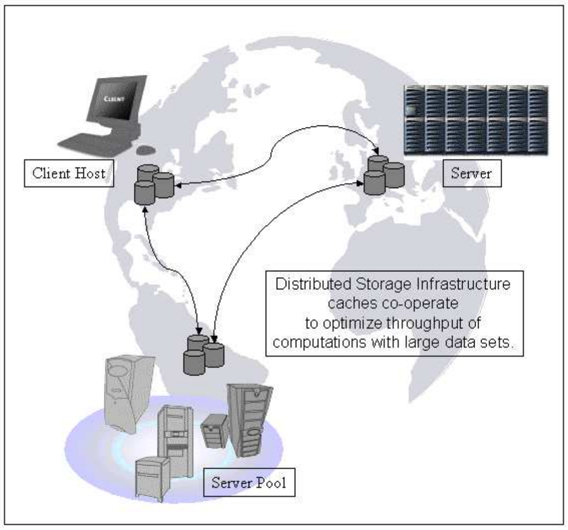

Figure 3: An architectural overview of the NetSolve system.

as objects of primitive data types, as in `MATRICES`, `VECTORS`, `SCALARS`, `STRINGS` and `FILES` of `INTEGERS`, `FLOATS`, `DOUBLES`, `CHARS`, etc. The read and write calls are designed to deposit and extract these objects from DSI storage. Figure 5.3.1 shows a sample programming code that makes calls to a version of the interface (simplified for this document.) The 'C' struct `NS_DSI_FILE` encapsulates information about the remote storage server and remote file object being allocated, while `NS_DSI_OBJECT` contains information about specific objects present within these remote files. In essence, a call to `ns_dsi_open()` allocates a remote file returning a handle or `NS_DSI_FILE *`. A call to any of the `ns_dsi_write_*()` variants will transfer data to the remote storage specified by the given handle. These handles can then be used directly in NetSolve requests or by reciprocating calls to `ns_dsi_read_*()` variants. When used in NetSolve requests, the NetSolve system resolves the handles to appropriate datasets and uses them either to locate input on which to perform computational analysis or to store the results of computation.

```
int client_program(){
NS_DSI_FILE * rfile;
NS_DSI_OBJECT * robject;
int *data1, size_data1, status;

...
rfile = ns_dsi_open("machine.domain.edu", "write");
robject = ns_dsi_write_matrix(rfile, data1, size_data1);

status = netsolve("solve_matrix", robject, rhs);
...
}
```

Figure 4: Sample 'C' code program shows details of NetSolve's DSI interface.

5.3.2 The implementation

The primary technique used in this first iteration of development is an instance of supply-side caching where the application at the client layer of the system is the supplier of data to the server which acts as the data consumer. The system is manually configured to strategically place storage servers near pools of computational servers as in the experiments of the next section. We also export to the user-layer the low-level interface described above that the user must use to dictate where data should be stored and where computations should be performed. Section 5.4 explains that this is not the extent of our efforts and discusses our plans to provide an architecture that automates the scheduling and allocation policies on behalf of the user.

The major modifications made to the NetSolve system involve the expansion of NetSolve's object model to include representations for remote data objects and files. The API functions described above in section 5.3.1, apart from the obvious, also put and later extract information to and from of these structures. NetSolve maintains a File Allocation Table (FAT) that records the status of allocated remote files and objects much like that used by operating systems to keep track of STDIO files. The reference values of the `NS_DSI_OBJECT`s and `NS_DSI_FILE`s are used as the keys by which these objects are cataloged in the

FAT. When NetSolve requests are made, input and output references in the calling sequence are checked against the keys of the FAT to see if they represent a remote object. (If not found, they are assumed to be referring to local data, in-core or on disk). The NetSolve system protocols accommodate remote data by sending data handles to servers which the servers use to obtain data from their stored locations. This implementation allows the NetSolve system to leverage DSI storage without modifying the standard NetSolve functions for computational requests.

5.3.3 Preliminary results

For our experiments, we wanted to show how this relatively simple use of DSI facilities could yield benefits and help motivate the investigation of more complicated techniques that either make the system easier to use or give even better performance. To mimic the geographical expanse of the Grid, we placed a NetSolve client application at the University of California, San Diego and experimented with requests to a pool of computational and DSI servers at the University of Tennessee. We used a set of matrices from the Harwell-Boeing collection of the Matrix Market repository [20] to solve systems of equations using the MA28 [11] solver library.

Figure 5 shows the results we obtained when varying the number of accesses (cache hits) made to the data from the Harwell-Boeing set. For various data sizes, we found the average times of 10 runs using traditional calls to NetSolve transmitting data over the network. We then made another set of runs with the same dataset, this time storing the data in DSI storage and having the server retrieve the data from the storage server. During these runs we collected the time expended for compute cycles, NetSolve overhead, network transmissions and DSI overhead. We used this collected data to deduce what the turnaround time would be as we increased the number of times the client application requested the computation. The two graphs of figure 5 show the results for datasets of size 16.1KB and 2.68 MB, respectively. These represent both the smallest and largest data sizes with which we experimented. We also collected data for a range of sizes in between these points (21.4KB, 35.7KB, 55.4KB, 247KB, 302.4KB, 995KB, and 1.01MB) and testify that they bear similar results. The presented graphs show a worst case of 7 accesses (in the 16.KB case) and 2 accesses (in the 2.68MB) needed before the overhead added by the DSI is outweighed by the reduction of network activity caused by cache reuse. The graphs reach an asymptotic level that represent the points at which computational capacity, and not network bandwidth and latency, becomes the system bottleneck. For the 2.68MB sample, this occurs at a point when the enhanced system is operating at more than three times (3x) faster than the unenhanced system. These results lead us to a great anticipation for some of the future development described below.

5.4 Other Research Issues

In addition to the data staging that DSI facilities currently provide to NetSolve, we anticipate several areas for additional research:

- Request Sequencing – This technique mentioned briefly in section 4.1.1 uses graph-scheduling techniques to co-allocate resources for a sequence of requests for Grid services [2]. In NetSolve, data persistence is used so that parameters that re-occur in the sequence can be staged near the compute servers that will use them. Current implementations keep data persistent on a single server that must service all compute modules in the sequence. DSI infrastructures can be leveraged to logistically store data conveniently accessible to multiple servers so that throughput can be increased due to the use of additional computational units.

- Incremental Programming – This is a technique used in software design where an application is processed in several modular phases. One way this can be useful in a Grid setting is to implement computational steering: the online management and modification of the data parameters of a computation either to investigate performance enhancement or to directly control and manipulate scientific discovery during (instead

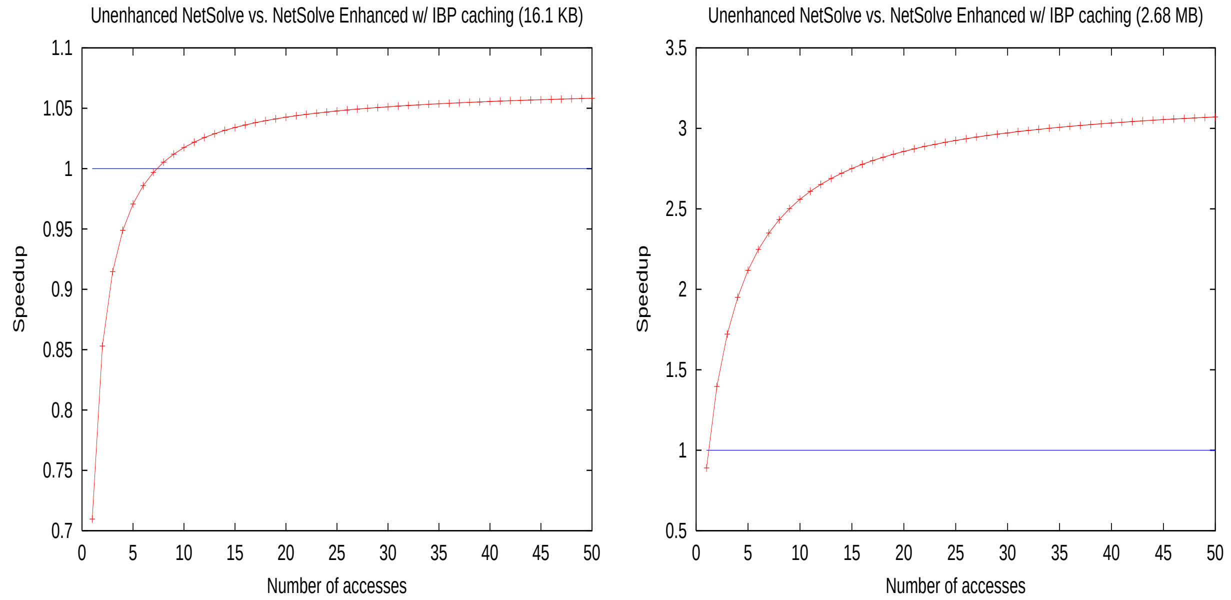

Figure 5: Encouraging results of improved computational efficiency when IBP caching is used to service NetSolve requests between components in California and Tennessee.

of after) the compute phase. Breaking the programs into finer-grained modules allows a user to perform an analysis of partial results mid-execution to determine if a phase needs to be repeated with new parameters or computation can proceed to a subsequent phase. Often in these scenarios the parameters that are modified are quite small, perhaps the value of initial water concentration in an oil reservoir simulation or the far-field pressure values in a structural acoustic finite element experiment. DSI infrastructures can be used to keep the invariable data housed near the compute servers as the user studies partial results to determine the direction of further computation.

- Data Repositories – DSI infrastructures can be used in NetSolve as data warehouses that can be used to store data from completed computations or even raw field data that can be expensive and difficult to come by. The National Weather Service, for instance, could house data concerning weather patterns and occurrences or satellite imagery allowing for access by University researchers that do not have the means to collect such data. An infrastructure like this greatly increases the number of scientists that can analyze what would otherwise be privileged collections of data – a fact that would potentially increase the rate at which the boundaries of science, atmospheric or otherwise, can be expanded. We already have a prototype implementation, within NetSolve, of the ability to store "named" DSI handles at the agent so that clients can access data by referring to some globally unique identifier. This allows clients to interact with and perform computations on third party datasets that they do not, and will not ever, possess.

- Scientific Collaborations – DSIs can also be leveraged in large, multi-institutional, geographically distributed scientific collaborations where different scientists are responsible for the development and execution of inter-dependent modules. Again, the DSIs can serve as globally accessible warehouses allowing all scientists to store and retrieve their own scientific results as well as that of their colleagues.

6 CONCLUSION

In this article, we have explored a variety of techniques and models to show how they might be employed in a Grid infrastructure designed to accommodate the execution of computations on large, cumbersome datasets in a widely distributed environment. As Grid developers continue to investigate the easy and efficient deployment of large applications on their systems, they will render some of these techniques more useful than others. We have also presented the initial results or our experiments with one such system, NetSolve, and shown how deploying even the least complicated of these strategies can yield significant application performance improvements. There are also a variety of important managerial, social and policy-related issues with respect to the management and sharing of data that the discussion in this paper does not address. These issues, which include the construction and management of metadata (or descriptional properties of data), cataloging and indexing, data protection and security, and the protection of intellectual property rights, on their own comprise a challenge to researchers to find satisfactory resolutions. We anticipate that the experiences obtained through continued research with computational and data grid infrastructures will eventually yield a practical and useful artifact.

Acknowledgments

The authors are also grateful to Susan Blackford and Victor Eijkhout of the Computer Science Department at the University of Tennessee for their constructive comments and critiques on earlier drafts of this paper.

References

[1] The Grid Physics Network: Project Description, 2000. www.griphyn.org.

[2] D. C. Arnold, D. Bachmann, and J. Dongarra. Request Sequencing: Optimizing Communication for the Grid. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, *Euro-Par 2000 – Parallel Processing*, pages 1213–1222. Springer-Verlag, August 2000.

[3] G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine Internet Technology Series*, May 2000.

[4] J. Basney and M. Livny. Improving goodput by co-scheduling cpu and network capacity. *International Journal of Supercomputer Applications and High-Performance Computing (IJSA)*, 1999.

[5] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. In *Proc. of Supercomputing'96, Pittsburgh, PA*, November 1996.

[6] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Sixth Workshop on I/O in Parallel and Distributed Systems, Atlanta, GA*, May 1999.

[7] H Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 1997.

[8] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets.

[9] D. Chess, C. Harrison, and A. Kershenbaum. Mobile Agents: Are They a Good Idea. Technical report, IBM Research Division, T.J. Watson

Research Center, Yorktown Heights, New York, March 1995.

[10] K. Claffy and D. Wessels. ICP and the Squid Web Cache, 1997.

[11] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matices.* Clarendon Press, Oxford, 1986.

[12] R. Ferreira, T. Kurc, M. Beynon, C. Chang, A. Sussman, and J. Saltz. Object-relational queries into multidimensional databases with the Active Data Repository. *International Journal of Supercomputer Applications and High Performance Computing (IJSA)*, 1996.

[13] I. Foster and C. Kesselman, editors. *The Grid, Blueprint for a New computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1998.

[14] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. 1999.

[15] I. Foster and K Kesselman. Globus: A Metacomputing Infrastructure Toolkit. In *Proc. Workshop on Environments and Tools.* SIAM, to appear.

[16] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Jr. Reynolds. A Synopsis of the Legion Project. Technical Report CS-94-20, Department of Computer Science, University of Virginia, 1994.

[17] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proc. of the 8th International Conference of Distributed Computing Systems, San Jose, CA*, pages 104–111, June 1988.

[18] I. Melve. Inter-cache Communication Protocols, 1999. IETF WREC Working Group Draft.

[19] D. Mitzel, D. Estrin, S. Shenker, and L. Zhang. An Architectural Comparison of ST-II and RSVP. In *Proceedings of Infocom*, 1994.

[20] Mathematical & Computational Sciences Division of the Information Technology Laboratory of the National Institute of Standards and Technology. Matrix Market Website. http://math.nist.gov/MatrixMarket/index.html.

[21] J. Plank, M. Beck, W. Elwasif, , T. Moore, M. Swany, and R. Wolski. IBP – The Internet Backplane Protocol: Storage in the Network. In *NetStore '99: Network Storage Symposium, Seatle, WA*, October 1999.

[22] K. Ross and V. Valloppillil. Cache Array Routing Protocol v1.1, 1998. IETF WREC Working Group Draft.