1 Overview

The goal of the research described herein is to gain a fundamental understanding of how programs can transparently leverage distributed resources to obtain high performance levels on behalf of their users. In particular, we study the execution environments, programming methodologies, and scheduling techniques that permit an application to adapt its behavior dynamically to changing performance conditions. A new distributed computing architecture – the Computational Grid [9] – describes a software infrastructure that, once implemented, promises vast potential performance. To study how that performance might be extracted, we wish to define and build software components that permit an application to adapt its execution dynamically, so that it can take maximal advantage of the resources that are at the disposal of its user.

The research plan described in this proposal focuses on the definition and experimental implementation of **EveryWare** *– a software toolkit for Grid programming, and* **G-commerce** *– computational economic systems for Grid resource control.* EveryWare will permit applications to implement flexible resource usage models that can tolerate dynamic performance fluctuation and failure, and G-commerce will ensure resource allocation is stable, fair, and high-performance. **The result of this work will be a novel programming methodology for the Computational Grid that is compatible with current Grid software infrastructures and research efforts**.

Complementing this research agenda is a commitment to education that emphasizes the application of theory to practical "real-world" problems as a way of answering scientific questions. Experimental computer science offers both promising research results as well as valuable learning experiences for students. Our principle objective is to foster new curricular directions in computer science that combine the mathematical and engineering aspects of the discipline with the tenets underlying good experimental design and data analysis. In addition to designing new courses that blend theory and practice with quantitative evaluation techniques, we plan to organize a series of practica in which students from different levels collaborate to design and execute computer scientific experiments. As EveryWare and G-commerce will enable the integration of otherwise disparate software infrastructures, they will serve as a useful vehicle for these educational experiences.

2 Introduction

Increasingly, the high-performance computing community is blending parallel and distributed computing technologies to meet its performance needs. A new architecture, known as *The Computational Grid* [9], has recently been proposed which describes a software infrastructure that supports high-performance applications using widely dispersed computing systems. The goal of a Computational Grid is to aggregate ensembles of shared, heterogeneous, and distributed computational, communication, and storage resources (potentially controlled by separate organizations) to provide computational "power" transparently to an application program. Applications should be able to draw compute cycles, network bandwidth, and storage capacity seamlessly from the Grid[1] in a way analogous to the way in which household appliances draw electrical power from a power utility.

Developing high-performance application programs for Computational Grids continues to prove difficult. While there are several successful software middleware development efforts [8, 14, 3, 35, 27] which provide basic low-level Grid functionality, significant research challenges must be met before the Computational Grid becomes a ubiquitous computing paradigm. In particular, the management of heterogeneity, federated ownership, resource performance fluctuation and failure must all be addressed effectively.

Our goal in the project is to provide a novel programming methodology and set of software tools for effective Grid programming. The methodology is based on application *customizable* services that allow the application programmer to manage heterogeneity and performance fluctuation, and a resource economy that applications can consult to determine resource worth. The software tools we propose are collectively called *EveryWare*. They allow an application to use other Grid middleware infrastructures such as Globus [8], Legion [14], NetSolve [3], and Condor [35] as well as "native" operating system functionality as basic com-

[1] We will capitalize the word "Grid" when referring to "Computational Grid" throughout this proposal.

putational building blocks. EveryWare has a three-part design based on a portable *linga franca*, distributed resource monitoring and management agents, and a distributed state exchange that allows these building blocks to be used to specify application-specific services. By structuring an application in terms of explicit customized services, the Grid programmer can ensure both high-performance and robust execution using a dynamically fluctuating Grid resource pool.

By itself, however, the ability to write adaptive and robust Grid programs does not ensure that they will achieve high-performance levels. Adaptive programs may constantly request resource reallocations as they respond to dynamic changes in resource load. We propose to study how an economic framework can be developed for the Grid that both ensures application performance and system-wide resource stability. We term the study of resource economies for the Grid *G-commerce*. While the idea of forming a computational economy as a way of controlling resources is not new [**?**, **?**, **?**, **?**] most of the computational economic systems that been developed are based on auctions. Auctions are attractive because they allow users or applications to bid for resources directly with no need to consult the global state of the overall system. Unfortunately, there is little theoretical or empirical evidence that an auction-based computational economy will lead either to market equilibrium or market stability. Our work will investigate alternative economic formulations from both theoretical and empirical perspectives. In particular, we will develop a *computational exchange* that acts as a pricing service for Grid resources based on dynamic estimates of supply and demand. Using the exchange, we will develop the economic rules necessary to build a computational commodities market that provably ensures market equilibrium and stability. Ultimately, the exchange will become the part of EveryWare responsible for management and allocation of federated Grid resources.

By combining the adaptive and flexible programming support provided by EveryWare with novel G-commerce resource scheduling techniques, we will define a new and powerful approach to programming Grid applications. We will measure the success of our approach in terms of the observable application performance demonstrated by non-trivial Grid applications in production computing settings. Since our research methodology will be experimental, we anticipate that the project will result in persistent software artifacts which we will make available though the University of Tennessee Scalable Intracampus Research Grid (SInRG) [**?**] NSF National Partnership for Advanced Computational Infrastructure (NPACI)[2].

In this proposal, we describe the structure of EveryWare, how it grew from Network Weather Service [41, 38], and AppLeS [1, 34] projects into a successful demonstration of Grid computing in the SC '98 conference, and how it will enable truly effective Grid programming. We also provide initial insights regarding computational economies for the Grid, and detail our plans for developing new G-commerce formulations. **The confluence of EveryWare and G-commerce, we believe, will result in a new and powerful programming methodology for the Grid**. In addition, we intend to make our research the basis for new educational opportunities and curricular directions for students. We plan to sponsor *Grid practica* for advanced undergraduates and graduate students which allow them to experiment with our software infrastructure in "live" Grid settings. We will also use the combination of systems, statistical, and economic research that this work will draw upon as the basis for a new cross-disciplinary curriculum in Grid computing.

The remainder of this proposal details our research plan, educational plan, and relevant prior NSF sponsored research. Section 3 describes the EveryWare and G-commerce research we intend to pursue, and how they will be combined to form a new Grid programming methodology and infrastructure. In Section 4 we outline how our unique approach will lead to novel educational opportunities, and Section 5 describes previous research efforts upon which this work will build directly.

3 Research Plan — EveryWare and G-commerce

Our research agenda is to develop a set of programming tools and resource allocation strategies that enable application programmers to develop high-performance programs for Computational Grids. Many sophisticated and powerful software infrastructures have been, and will continue to be, the subject of active research. Rather than defining a new and potentially competing infrastructure, we focus on on the development of

[2] The PI is currently both a SInRG co-PI, and an NPACI partner.

tools that allow an application to *combine* abstractions supported by different existing middleware as building blocks. We call this toolkit *EveryWare*. Using EveryWare, we will investigate effective programming methodologies for the Grid, where resource availability and performance can vary dramatically with time, and distributed resource allocation strategies that promote performance and system stability. Much of the current Grid programming support is based on legacy parallel programming abstractions from MPI [7] and PVM [10]. These systems assume that a a static and homogeneous resource pool will be available to an application throughout its execution. If a machine becomes slow due to contention or fails, or if connectivity across a network link is temporarily interrupted applications using the faulty resources must typically terminate. Checkpointing systems such as [35] and [?] partially address the problem by allowing an application to save global program state. They do not, however, permit the number of resources the application uses to change dynamically, nor do they provide migration facilities that the application can control.

More sophisticated Grid programming middleware such as Globus [8], Legion [14], and NetSolve [3] provide better support for dynamic allocation and release, but *no extant system of which we are aware allows the application programmer to explicitly manage resource performance fluctuation, federated administration, and resource heterogeneity inherent in the Grid computing paradigm.* EveryWare [39, 40] is a set of software tools we are developing that allow a Grid application to be structured as a collection of robust and adaptive services triggered by independent computational clients. Its application-specific service architecture is similar in form to that provided by JINI [?] and HP-speak [?], but EveryWare is designed to support multi-language development and Grid performance levels.

We will use EveryWare as a tool for the study of robust, decentralized scheduling techniques based on resource economies. Economic resource allocation schemes are attractive because they require little or no centralized control, and they naturally capture cost-benefit relationships that are central to effective scheduling. We call this research of effective economic systems for the Grid *G-commerce*. Our plan is to study G-commerce analytically, in simulation, and through experimental prototype. As a vehicle for empirical investigation we will implement a *computational exchange* that dynamically monitors resource supply and demand, and is able to apply G-commerce pricing rules to determine a fair price. EveryWare applications will then purchase their resources based on the current price. Using the G-commerce economy and the computational exchange we believe we will be able to ensure that competing applications do not oscillate with respect to their resource usage (e.g. the overall system will remain stable) and that applications are charged the correct fair price.

By combining the robustness and flexibility of EveryWare with resource allocation strategies based on G-commerce, we will define a fundamentally new programming methodology for the Grid. Applications will be able to define customized services using the EveryWare primitives that dynamically choose the best resources in terms of G-commerce cost and benefit. At the same time, because applications are using a G-commerce resource model, we believe we can show that the overall system will remain stable. The key components of our research plan are

- to develop EveryWare as a Grid programming system capable of implementing *customizable* application services using existing Grid middleware as building blocks,
- to study (both theoretically and empirically) decentralized scheduling techniques based on *computational economies* that ensure fairness and overall system stability (G-commerce), and
- the development of a *computational exchange* that integrates the EveryWare infrastructure with G-commerce principles to form a new programming methodology for the Grid.

In the following subsections, we detail our approach. In addition to the initial results we present, our previous experience with the Network Weather Service and AppLeS systems encourages us to pursue this line of research, and to believe in our ability to achieve successful results.

3.1 The EveryWare Toolkit

The EveryWare toolkit will consist of three integrated components:

- a portable *lingua franca* that enables process, communication, and storage control across separate Grid infrastructures,
- a set of *dynamic forecasting* services and libraries that enable "on-the-fly" performance prediction, and
- a distributed, multi-infrastructure *state-exchange* service that allows application components to synchronize replicated data structures.

The *lingua franca* provides a base set of resource control abstractions that are portable across infrastructures. They are intended to be simple, easy to implement using different Grid technologies, and highly portable. Initially, we will focus on developing a simple *process abstraction*, *datagram message abstraction*, and *storage buffer abstraction* for EveryWare. The process abstraction creates and destroys a single execution thread on a target resource that is capable of communicating via both the EveryWare datagram message abstraction, and any local communication facilities that are present. EveryWare datagram messages are sent between processes via non-blocking send and blocking receive calls and processes can block waiting for messages from multiple sources. Processes can also create and destroy storage buffers on arbitrary resources (e.g. by creating "memory" processes that respond to read and write requests to their own memory).

Note that the EveryWare *lingua franca* differs from other message passing implementations such as PVM [10] or MPI [15] in several important ways. First, these other interfaces are designed to support arbitrary parallel programs in environments where resource failure is rare (i.e. on parallel machines). As such, they include useful primitives (such as global barrier synchronization) that make them attractive programming facilities, but sometimes difficult to implement in Grid environments. Often, when a resource fails, the entire PVM or MPI program must be restarted. EveryWare assumes that resource availability will be dynamically changing. As such, all primitives obey user-specified time outs, success and failure status is reported explicitly, and only those primitives that can fail individually (that is, without affecting more than the process calling them) are implemented. We do not intend the *lingua franca* to replace any of the existing message-passing or remote invocation systems that are available to Grid programmers. Rather, we will provide the minimal functionality required to allow these infrastructures to interoperate efficiently so that programs can span Grid infrastructures. We expect that the portability of the *lingua franca* will also benefit from this minimalist approach.

Dynamic performance forecasting is critical to EveryWare. It allows both EveryWare abstractions, and applications that use them to anticipate performance changes and minimize their negative impact. We will enhance the current Network Weather Service functionality by providing a set of *dynamic benchmarking* libraries that can be loaded with an arbitrary process. Using dynamic benchmarking, a program can time an arbitrary region of its code and pass the resulting timing data to the forecasting library. When a prediction of the execution time for the instrumented region is desired, the forecasting library will consult the previous timing data to make a statistical forecast. Like NWS resource performance forecasts, dynamic benchmarking forecasts include load and contention effects, but for arbitrary, user-specified program events.

The distributed state-exchange service provides an EveryWare application with the ability to replicate program state, and to keep replicants synchronized. EveryWare state-exchange servers (called *Gossip*s) allow application processes to register for state synchronization. The application component must register a contact address, a unique message type, and a function that allows a *Gossip* to compare the "freshness" of two different messages having the same type. All application components wishing to use the Gossip service must also export a state-update method for each message type they wish to synchronize.

Once registered, an application component periodically receives a request from a *Gossip* process to send a message indicating the freshness of its current state. The *Gossip* compares that message (using the previously registered comparator function) with the latest freshness message received from other application components. When the *Gossip* detects that a particular message is out-of-date (e.g. the state it refers to is "old"), it sends a fresh state update to the application component that originated the out-of-date message.

Figure 1 depicts the relationship between these components. The **lingua franca** enables application components written by the user for different Grid infrastructures to interoperate with each other, the EveryWare state exchange service, and with other multi-infrastructure services such as the Network Weather Service [41]. **Dynamic forecasting** libraries (small triangles in the figure) can be loaded with application

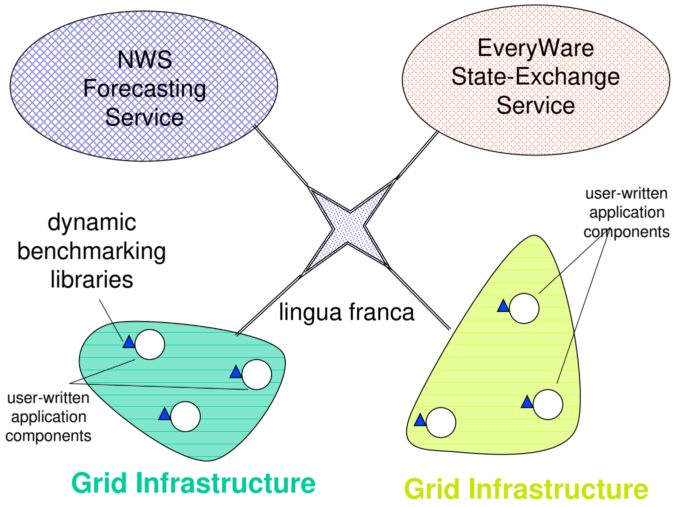


Figure 1: **EveryWare Components**

components directly. These libraries, in conjunction with the performance forecasts provided by the NWS, permit the program to anticipate performance changes and adapt execution accordingly. The distributed **state-exchange services** provide a mechanism for synchronizing and and replicating important program state to ensure robustness and scalability.

3.2 Customized Application Services – The EveryWare Programming Model

EveryWare programs are structured as sets of *computational application clients* or tasks that request run-time management services from *application-specific servers.* Application clients perform the actual "work" within the application using the features of a native Grid infrastructure. They may, themselves, be parallel or distributed programs and they are not constrained to use only the *lingua franca* for communication, process control, and storage management. For operations, such as scheduling, user interaction, etc. that require more global control, the computational application clients appeal to application-specific servers, also written by the application programmer. Like the clients, the application-specific servers are not constrained to use any single communication or process control mechanism – they may be written to use any native Grid infrastructure. However, using the *lingua franca* enables arbitrary client and server interaction and ensures portability across infrastructures. The EveryWare toolkit provides the basic implementation framework that the application programmer can customize for a particular application.

This application architecture offers several advantages. First, the overall program can be constructed incrementally. Since the EveryWare toolkit is robust with respect to host and network failure, new clients and servers can be added, old ones removed or improved, etc. while the application itself continues to execute. Another advantage is that it allows us to implement infrastructure-specific clients that can get the best possible performance by running in "native" mode. Application clients need only speak the protocol required by each server, obviating the need for a complete software veneer between the computational code and the native infrastructure.

Note that the EveryWare programming model is fundamentally different from that used by most procedure oriented Grid infrastructures such as NetSolve [3], NINF [27], and NEOS [25]. These infrastructures typically support applications structured as a single controlling client that makes remote-procedure calls to remote computational servers. Under the EveryWare programming model, computation is centered at the clients and program control is coordinated by a set of cooperating application-specific servers. Since the roles or client and server are reversed, we term this application architecture an *inverted client-server* model.

3.3 Initial EveryWare Results

To test the potential efficacy of EveryWare, we implemented prototype versions of the EveryWare software components and used them to deploy a globally distributed Grid application using the inverted client-server model at SC98 [32] – a nationally prominent conference focusing on large-scale, high-performance computing. The first EveryWare application attempts to improve the known bounds of classical Ramsey numbers by searching for counter examples [30], using different heuristics to cover the search space. We used the inverted client-server programming model in developing the application and implemented an application-specific scheduling service, a distributed persistent state service for application variables, and a distributed logging and I/O service for the application (as depicted in Figure ??).

The n^{th} *classical* or *symmetric Ramsey number* $R_n = R_{n,n}$ is the smallest number k such that any complete two-colored graph on k vertices must contain a complete one-colored subgraph on n of its vertices. It is possible to prove that R_3 is 6, and that R_4 is 18, but the exact values of R_n for $n > 4$ are unknown. However, some bounds are known, for example $43 <= R_5 <= 59$ and $102 <= R_6 <= 165$. If it is possible to find graph coloring on between 43 and 58 that has no monochromatic, full-interconnected subgraphs of size 5, then the bounds on R_5 will be reduced. Our Ramsey Number Search (RNS) application attempts to find such counter-example colorings.

This application was an especially attractive test of EveryWare because unlike other global search applications, such as RC5 cryptographic analysis [2] and the SETI-at-home [33] sky search, exhaustive search is infeasible. For example, to find a new lower bound for R_5 exhaustively, one must consider approximately 10^{270} different two-colored graphs on 43 vertices. Even if one could examine 10^{12} configurations every second, an exhaustive search would take over 10^{250} years. It is this level of combinatoric complexity makes exhaustive search techniques like those employed in cryptographic factoring [24, 2] ineffective. Rather, RNS uses a form of distributed "branch-and-bound" state-space searching in which different heuristics (tuned for various architectures) traverse the search space. Instead, the search heuristics must be carefully scheduled and must communicate non-trivial amounts of program state amongst themselves while they execute.

In Figure 2 we show the sustained performance of the EveryWare application we deployed at SC98, spanning the 12 hour period leading up to the High Performance Computing Challenge [18] (HPC Challenge) which was held at 11:00 AM (PST) on Wednesday, November 12, 1998. Using EveryWare, we were able to integrate application components written for Globus [8], Legion [14], Condor [35], NetSolve [3], Java [11], Windows NT [37], and Unix into a single, globally distributed execution instance. During the course of the experiment, we used approximately 550 different processors including rare or one-of-a-kind supercomputers such as the Tera MTA [36] the NT Supercluster [4] and the Berkeley NOW [5], powerful high-end machines such as the Cray T3E and IBM SP2, workstations and shared-memory multiprocessors, and desktop web browsers. The locations of the machines we were able to combine spanned four continents, and all machines were accessed in complete non-dedicated mode without special user privileges or enhanced user priority. That is, we formed a Computational Grid that consisted of the standard user access we could obtain from all of the systems to which we log in. The lefthand figure shows the sustained application performance (on a log scale) in terms of integer operations per second. During the twelve hours leading up to the HPC Challenge, the application was able sustain between 1.1 and 2.5 billion operations per second. This processing rate includes all communication delays and scheduling overheads – it is the performance *delivered* to the application by the Computational Grid. In the righthand graph, we detail the delivered performance by infrastructure. During SC98, many of the resources that we could access were heavily loaded or intermittently reserved for other dedicated demonstrations. As result, the performance our EveryWare application could obtain from any single infrastructure varied widely, particularly as the time of the HPC Challenge competition drew near. Despite wide and dynamic variation in the performance that any one infrastructure could deliver (shown in the righthand graph), the EveryWare application itself was able to sustain supercomputer performance levels (shown in the lefthand graph) using only non-dedicated, shared access to federated resources. As such, *the EveryWare application constitutes one of the first examples of an adaptive Grid program that is able to demonstrate fully the Computational Grid paradigm.* EveryWare was awarded "Best Acceleration" for this

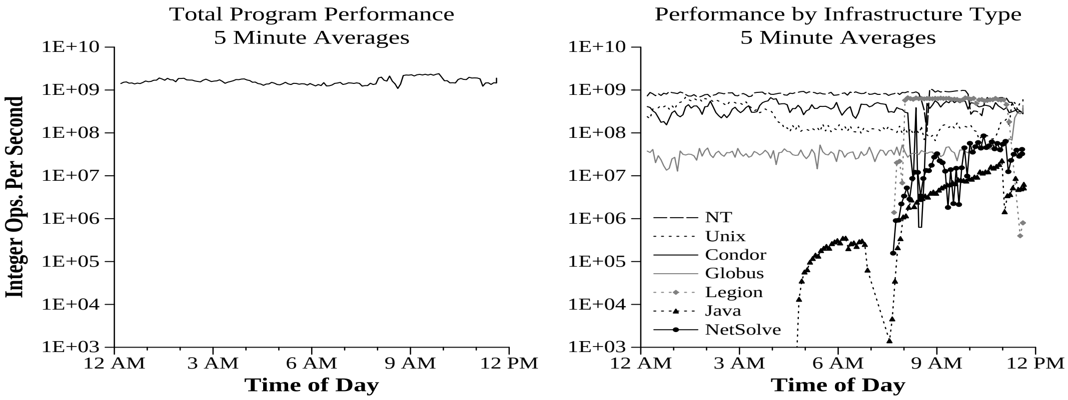


Figure 2: **EveryWare Application Performance (integer operations/second)**

accomplishment in the HPC Challenge. Details associated with the EveryWare experiment are provided in [39] and [40].

3.4 Resource Allocation – G-commerce

We have also been studying the problem of resource allocation in the context of large-scale Grid execution settings. One of the key weaknesses exposed by the EveryWare experiment at SC98 was that the EveryWare application schedulers (like AppLeS agents [1, 34]) made their resource usage decisions strictly in terms of the performance benefit each resource offers to the application. This greedy, self-interested strategy is effective if only a single application is active at any one time, and if resources are essentially free. The EveryWare Ramsey Number Search application, for example, would only abandon a resource if it would fail to execute an application component fast enough to prevent another component from covering the same part of the search space. The result was that many computations were performed multiple times since there was no true cost function in place. Further, if two EveryWare programs were to be executed simultaneously on the same set of resources, this greedy scheduling strategy would cause scheduling oscillations. The scheduler for one application would respond to the resource loads introduced by the other, and vice versa, causing the applications to continuously avoid each other.

Our solution to the problem of ensuring stability and preventing resource waste (due to excessive release and reallocation) is to define a computational economy for each application to use. Application schedulers will be charged (in terms of a fictitious currency) for their resource usage according to the current market price set for the available resources. Relying on an economic approach to resource allocation in distributed systems is not a new idea [**?**]. Indeed, several agent-based systems [**?**, **?**, **?**] have proposed the use of auctions as a method of controlling resource usage. Chun and Culler also use a modified auction system as part of the Millennium system [**?**] to control cluster allocation. While all of these approaches share the advantage of being decentralized, there is little evidence that they will lead either to a stable economy, or to one in which applications are charged a fair price for their usage.

Working with Dr. James Plank from the University of Tennessee, and Dr. John Brevik from Holycross College, we have been investigating dynamic pricing schemes based on the work of Smale [**?**] as an alternative. Smale's result says that a fair price which brings the market to equilibrium must exist, and can be efficiently found, while making very limited assumptions about individual producer and consumer behavior.

More formally, if commodity prices are represented as a *price vector* $\mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}$, where p_i stands for the price of the i^{th} commodity, we can define *excess demand* z_j for the j^{th} commodity as the demand minus the supply. As defined, z_j may be positive or negative; negative excess demand can be interpreted simply as excess supply. We assume that the markets for these commodities may be interrelated, so that each z_j is a function of all of the prices p_i, that is, of the vector $\mathbf{p}$. Smale's theorem says given a market consisting of n interrelated commodities with price vector $\mathbf{p}$ and associated excess demand vector $\mathbf{z}(\mathbf{p}) = \mathbf{z} =$ an *equilibrium point* with $\mathbf{p}^*$ such that $\mathbf{z}(\mathbf{p}^*) = 0$ exists [?]. Moreover, for any value of $\mathbf{p}$, we can form the $n \times n$ matrix of partial derivatives

$$D_{\mathbf{z}}(\mathbf{p}) = \left(\frac{\partial z_i}{\partial p_j}\right).$$

Then for any value of λ which has the same sign as the determinant of $D_{\mathbf{z}}(\mathbf{p})$, we can obtain economic equilibrium by always obeying the vector equation

$$D_{\mathbf{z}}(\mathbf{p})\frac{d\mathbf{p}}{dt} = -\lambda \mathbf{z}(\mathbf{p}). \tag{1}$$

This formulation's main drawback, however, is that it assumes that an instantaneous "image" of the aggregate supply and demand for all resources is available. We will discuss our proposed efforts to relax this restriction in the next subsection.

3.5 Proposed Work

While the experiment we conducted at SC98 helped to form our initial conception of EveryWare, it represents only the most initial phase of the work. Our overall goal is to explore the efficacy of combining EveryWare and G-commerce as a novel programming methodology for Computational Grids. To do so, we plan a research effort that will

- develop and expand the EveryWare tool set to work within a computational economic framework,
- study and formulate effective resource economies that EveryWare programs and the current batch of middleware infrastructures can exploit,
- verify, demonstrate, and quantify the utility of the results using working Grid programs.

Our general approach will combine theoretical and simulation studies with empirical observation of non-trivial application in "live" Grid settings.

3.5.1 Developing EveryWare

We plan to develop tools that allow an application to control ubiquitously computation, communication, and storage across separate Grid environments. In addition to expanding the base functionality, we will investigate scheduling abstractions and resource control primitives that will be able to interact according to the rules of a given computational economy. For the SC98 experiment, the prototype **lingua franca** focused on communication. We will expand this functionality to include both process control and distributed storage management. We will also add facilities for discovering the anticipated price of different resource combinations, and for monitoring application "spending" while it is in execution. To do so, we will add an application customizable event management facility to EveryWare based on the model proposed by the Grid Forum Performance Working Group [?].

The performance forecasting facilities will be enhanced to support the *dynamic benchmarking* of an executing EveryWare application. To predict application performance, current dynamic scheduling systems such as AppLeS [1] rely on statistical forecasts of resource performance that are then used to parameterize a compositional application model. As an alternative, we wish to use historical program instrumentation data, taken from the "live" running program, to make short-range predictions as well. That is, we will instrument the application to time arbitrary regions of code. If the timed regions are repeated, we will apply the EveryWare performance forecasting facilities to the "live" timing data and generate performance predictions for the application on-the-fly. We will explore the use of program instrumentation tools such as Paradyn [26], Pablo [6], and Autopilot [31] to generate the program performance measurement data.

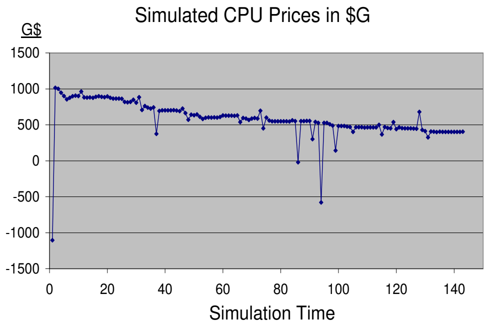
3.5.2 Investigating Grid Programming Models

We will use the EveryWare toolkit to investigate different Grid programming models. Currently, most Grid programs employ one of three broad classes of programming model: Single-Program-Multiple-Data (SPMD)[7, 10, 15], Remote Procedure Call (RPC) [3, 25, 27], or Object Oriented [14, 11, 29]. Since EveryWare can combine infrastructures that support these various models, it will be possible to write an EveryWare program that uses them individually or in coupled combinations. We will investigate the performance advantages and liabilities associated with each for different applications and Grid settings.

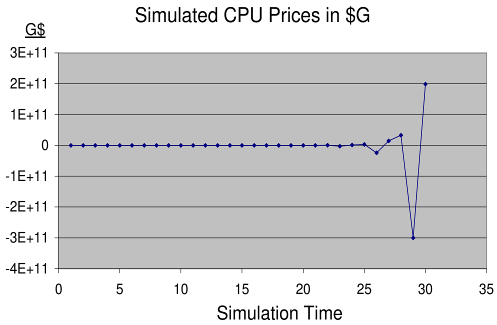
In addition, we will focus on the *inverted client-server* model in which computational *clients* request application-specific program management *services* from distributed application-specific servers. Application-specific distributed services can leverage the details of the application for performance while meeting the robustness requirements necessary to support the Computational Grid paradigm. However, building unique services for individual application functionalities complicates program development substantially. To ease this added burden, we will develop a *server framework* that will enable code reuse. Applications will register application-specific server modules (in a way that is analogous to the Java "servlet" mechanism [20]) with a specific service. The structure of each individual server frameworks as well as the necessary registration mechanisms will evolve as the model does during this project.

3.5.3 Developing Economic Formulations for the Grid

Our early work with G-commerce indicates that the problem of defining a resource economy for the Grid poses a substantial challenge. While Smale's result proves stability and equilibrium based on theoretical principles, economic formulations that conform to its underlying assumptions are not obvious. For example, consider the two graphs shown in Figure 3. Each graph shows a time series of $G (a fictitious currency)



(a)



(b)

Figure 3: Stable economic formulation (a) and and unstable economic formulation (b) in simulation.

for CPU usage by a set of simulated client Grid applications based on dynamic estimates of supply and demand. Each client is given a total amount of work to perform (split into jobs of random length) and an initial budget. CPU servers are configured to sell a fixed percentage of their capacity which varies randomly from one CPU to the next. Each client is willing to buy CPU only if the proportion of cost of executing its job to its total remaining budget is less than the proportion of the length of the job to the total amount of work it has to perform. That is, "small" amounts of work (relative to what each client needs to perform) should cost "small" amounts of money (relative to the wealth of each client) and "large" amounts should cost more. CPUs are willing to sell if the profit to be gained by executing a job is greater than the average profit (over time) the CPU has been able to garner. All distributions are uniform.

Figure 3a shows a stable configuration based on profit-oriented purchasing rules for both simulated clients and CPU servers. Prices decline over time since client budgets are fixed and average CPU profits (for non-selling CPUs) drops with time. Figure 3b, however, shows a slightly different economic formulation in which client budgets are periodically replenished. After a period of stability, the system begins to oscillate and eventually encounters numerical instability. Both economic formulations are plausible in actual Grid settings. We are studying the dynamics of the system to determine what characteristics are required to ensure stability. Notice that in either case, however, we do not mandate that the supply and demand curves have any particular formulation, nor in fact, that they remain constant. Rather, individual resource consumers (applications) and producers (CPUs, in this case) make localized decisions based on self-interest. The combination of this desirable property with the potential for provable stability makes our approach attractive over those that have been previously investigated.

As part of this project, we will develop analytical results exposing the dynamics of the underlying resource economy, a simulation environment (much like the one used to generate the results shown in Figure 3) for testing these results, and application scheduling algorithms and schedulers for "real" EveryWare applications. In particular, we will focus on the question of stability in the presence of imperfect information. It will not be possible to capture a completely accurate picture of Grid-wide resource supply and demand. Rather, we believe we will be able to bound the level of inconsistency as a function of resource performance fluctuation (i.e. network transit delay). We will attempt to identify economic formulations that remain stable with respect to these bounds, and to demonstrate their stability both analytically and empirically.

3.5.4 Building a G-commerce Infrastructure

As a tool enabling empirical investigation of G-commerce, we will develop a computational exchange for the Grid that *continuously gathers supply information* from Grid resources, and *demand information* from executing Grid applications, *calculates the price* for each resource, and *advertises the pricing information* via a number of well-defined APIs. We will modularize this "First Bank of the Grid" so that it can accept different economic formulations allowing us to verify the equilibrium and stability characteristics of our G-commerce results.

Our plan is to develop the First Bank of the Grid, using the Network Weather Service (NWS) [41, 38] as a substrate. Currently, the NWS dynamically determines (in near-real time) the performance that is available from a collection of Grid resources. In a G-commerce model, this information can be used to determine the aggregate *supply* that is available from one moment to the next. We will develop analogous *demand* sensors (based the EveryWare dynamic benchmarking facility) for applications so that the same mechanisms can be used to gauge aggregate *demand*. We will then add a distributed and robust *pricing service* that calculates resource prices and advertises them via a set of *Branch Offices*. Figure **??** depicts this architecture.

Our plan is to develop demand sensors, initially, for EveryWare and to deploy the First Bank of the Grid on the University of Tennessee Scalable Intracampus Research Grid (SInRG) [**?**] (recently funded by NSF). SInRG will provide a development and verification environment based on the usage patterns of real-world Grid users from the University of Tennessee Grid computing community. In addition, we will dovetail this work with the on-going development efforts within the Grid Application Development Software [**?**] project, and at the National Partnership for Advanced Computational Infrastructure.

3.5.5 Application Programming with EveryWare and G-commerce

To verify the efficacy of our approach, we will implement non-trivial Grid applications using EveryWare and G-commerce. We will develop customized scheduling services (using the EveryWare toolkit) for each application, instrument it with G-commerce demand sensors, and deploy the application in conjunction with the First Bank of the Grid in multiple "live" Grid settings. These application investigations will test the architectural characteristics of EveryWare, serve as the source of our empirical results, and act as programming templates for other similar Grid applications. While we expect to identify suitable test applications as the project evolves, we will focus on two codes initially: *Ramsey Number Search*, *PET Imaging*.

We will continue our initial work with Ramsey Number Search (RNS) which attempts to improve the known bounds of classical Ramsey numbers. Our experience at SC98 showed that to search for R_6, we will

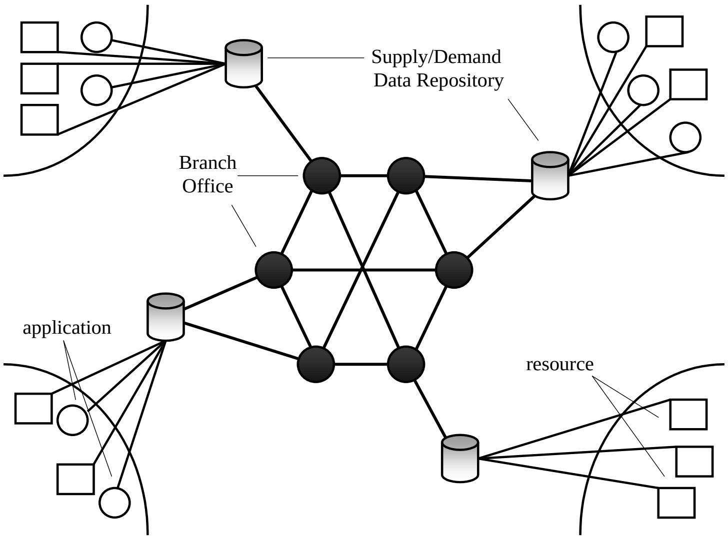


Figure 4: **The First Bank of the Grid Structure and Components**

need to parallelize some of the individual heuristics, each of which we will implement as a computational client within the application. As a result, we will develop ways in which EveryWare can be used to couple tightly synchronized parallel codes running on parallel computers with other Grid application components.

We will also implement a Positron Emission Tomography (PET) image reconstruction application using EveryWare. PET is used to study internal physiology non-invasively. By recording the positron emission patterns of an injected radioisotope as it is absorbed by soft tissue, a computer can construct an image of the infused tissue. Several iterative image reconstruction algorithms have been investigated [17, 23, 22]. For this project, we will use an implementation of the expectation-maximumization maximum-likelihood (EM-ML) algorithm coupled with a focus-of-attention pre-processing phase developed by Gregor and Huff [13]. This particular PET application is attractive because its pre-processing phase has been shown to dramatically improve is performance when implemented as a parallel program.

We have chosen these test codes for two reasons. First, each one combines both master/slave parallelism and data parallelism in its current implementation. Their complex structures will expose the degree to which EveryWare is able to support flexible management of concurrency. Secondly, the communication and synchronization requirements of their constituent algorithms are non-trivial. The test applications are not "embarrassingly parallel" and, we believe, will require careful scheduling to obtain good performance levels in Computational Grid settings.

3.6 Research Plan Summary

The research agenda that we are proposing to pursue will focus on the following research tasks.

- **We will define and implement the EveryWare toolkit**. The *lingua franca* will provide rudimentary process control, communication, and storage capabilities portably. We will develop a *dynamic benchmarking* forecasting system to incorporate program execution histories. We will also develop distributed state management facilities and registration APIs necessary to support the EveryWare *inverted client-server* programming model.

- **We will study and develop economic formulations that will form the basis of G-commerce based resource allocation strategies**. Our approach will be based in theory, simulation, and empirical implementation resulting in *application schedulers* for EveryWare applications.

- **We will develop the First Bank of the Grid as a research vehicle for G-commerce based resource scheduling.** Using the NWS as a substrate, we will construct a distributed, portable, and robust system for dynamically calculating and disseminating Grid price information.

- **We will implement non-trivial Grid applications using EveryWare and the First Bank of the Grid**. Our empirical research approach will rely on application implementation as a source of research results and validation.

This work complements and extends our previous work with AppLeS and the Network Weather Service. We will continue to collaborate with members of the AppLeS research team and with different Grid software infrastructure efforts. As with our previous projects, our approach will be experimental and applied, and we anticipate that it will result in useful software artifacts that we will make available through the National Partnership for Advanced Computational Infrastructure. Perhaps more importantly, however, we believe that a successful implementation of this research plan will yield a new level of understanding within the Computational Grid and distributed computing research communities.

4 Education Plan

Experimental design is a fundamental and sometimes overlooked aspect of computer science education. For systems with a dynamic performance response, it is often difficult to recreate experimental conditions and to determine the significance of observed results. Theoretical analysis techniques are often stymied by the mathematical complexity associated with the modeling of these systems yet the community continues to engineer and build them (e.g. the World-Wide-Web).

Our educational approach will be to foster curricular directions that emphasize an empirical approach to the analysis development of working systems. Theoretical techniques will be applied to real-world computer scientific problems and the results observed. A rigorous evaluation of the results will then be used to suggest better solutions. To this end, we will

- develop course offerings that blend theory, systems engineering, and scientific data analysis, and

- sponsor practica for visiting students during which they will design, carry out, and analyze computer scientific experiments in a controlled setting.

4.1 Course Development

We will develop a series of courses in distributed systems that ask students to experimentally evaluate alternative approaches to important systems problems. Topics such as distributed consistency, naming, scheduling, group membership, and stability all have strong theoretical underpinnings. The goal of the course will be to investigate, in a scientifically rigorous way, the value of different theoretically attractive approaches under different contrived and real-world conditions and to engineer useful solutions based on the results. The courses will combine an exposition of the theoretical solutions to the topic problem at hand, a description of potentially useful statistical analysis techniques, and a software laboratory that students can use to design and carry out experiments. EveryWare will serve as the toolkit for use in the laboratory. We envision each course as a two semester sequence – the first describing theoretical and engineering approaches, and the second focusing on experimental design, data analysis, and laboratory implementation of experiments to determine the utility and practicality of the concepts covered during the first semester. The goal will be to use real distributed systems problems to expose and motivate both theory and practice. We anticipate the initial course offerings to target graduate students interested in Computational Grids and distributed computing as research directions. However, after gaining experience with the course at the accelerated pace of graduate study, we plan to offer similar but less intensive empirically-based courses in systems design at the undergraduate level.

4.2 Computational Grid Practica

We will also sponsor short practica for visiting students that emphasize controlled experimentation using distributed computing and Grid computing infrastructure. During the EveryWare experiment at SC98, students from the University of California at San Diego and the University of Washington formed teams, each targeting a different Grid infrastructure. EveryWare allowed them to develop both collaborative and competitive solutions to the various Grid computing problems we needed to solve for the High-performance Computing Challenge competition. By carefully recording and analyzing the the performance data "on-the-fly" the student teams were able to identify useful techniques and improve the overall effort. Because students could see the results of their improvements immediately, and could work together in close contact with each other, they rapidly discovered the most effective techniques. Many of the students that participated in the SC98 EveryWare experiment ranked the project as one of the most beneficial (and exciting) among their educational experiences. At the same time, students gained valuable Grid programming skills using all of the available Grid infrastructures available at the time.

We plan to create a similar development environment and to invite students from different Universities to work collectively on selected Grid computing applications. Both competitive and collaborative techniques will be explored. Students who wish to participate will be furnished with a test harness and a problem description three to four weeks before each practicum begins. Participating students will then be invited to attend a three to five day workshop at the University of Tennessee during which they will be asked to deploy their solutions on a controlled Grid testbed. We will analyze and compare, quantitatively, the efficacy of each solution during the first part of each workshop outlining the strengths of each approach. The students will then be asked to integrate the most effective techniques collaboratively based on this analysis.

Both upper division undergraduates and graduate students will be invited to participate as will researchers from the field of Grid computing. The University of Tennessee Computer Science Department will provide both the laboratory space and the computing infrastructure necessary to support each workshop. In addition, we will collaborate with the Joint Institute for Computational Science (JICS) [21] at the University of Tennessee to design, administer, and publicize each practicum. We believe that between five and ten students and at least one visiting researcher can be accommodated on an annual or bi-annual basis.

5 Prior NSF Sponsored Research

EveryWare and G-commerce are intended to leverage emerging Computational Grid infrastructure technology [8, 14, 35, 3, 11, 37] and to build upon prior results gleaned from the Network Weather Service (NWS) [41, 38] project. In addition, it will benefit from, and complement the on-going research efforts of the Grid Application Development Software (GrADS) [?] project. Finally, we plan to use the Scalable Intracampus Research Grid (SInRG) [?] currently being deployed at the University of Tennessee as testing and verification platform. The NWS uses up-to-date performance histories gathered from Grid resources and statistical forecasting techniques to make short-term performance predictions for those resources. Schedulers, be they human or automatic scheduling programs, can consult these forecasts when making resource usage decisions. The AppLeS methodology [1, 34] uses a customized scheduling agent to make these decisions automatically, on behalf of the application's user, and has proved effective as an application scheduling strategy. Our previous work indicates that using NWS dynamic performance forecasts as the basis of dynamic scheduling systems can yield performance benefits in Computational Grid settings [9].

5.1 The Network Weather Service (ACI-9701333)

For any application scheduler to be able to generate a schedule that is tailored to the ambient performance conditions, it must be able to *predict* what the deliverable resource performance *will be* when its application executes. If it relies solely on readings of the current load conditions, the schedule will be crafted considering the performance that was available when the readings were taken, and not as it will be when the schedule is eventually implemented. Rather, a dynamic scheduler requires a forecast of future performance levels to make its decisions.

To provide this functionality, we have developed the Network Weather Service (NWS) [38]. The NWS

- operates a distributed set of performance sensors, from which it periodically collects performance measurements,
- applies a set of statistical forecasting techniques to individual performance histories, and
- generates forecast reports for the resources being monitored, which it disseminates via a number of different APIs.

The goal of the system is to apply forecasting methodologies to "fresh" performance monitoring data gathered from Grid resources to make forecasts of available performance in near-real time. The current implementation of the NWS is fully distributed, robust, and compatible with a variety of Grid infrastructures [41] including Globus [8], Legion [14], NetSolve [3], Condor [35], and NINF [27] as well as MPICH [15], and native Unix sockets. In addition to performing its function as the primary source of dynamic information for AppLeS, we maintain a nationally distributed installation of the NWS as part NSF's National Partnership for Advanced Computational Infrastructure (NPACI) and the National Computational Science Alliance (NCSA) which can be accessed via the World-wide Web at the address given in [28]. The system is also being used at over 50 sites in the U.S. as well as at the Tokyo Institute of Technology and the Japanese Electronic Technology Laboratory [27], the French Universite Lyon I, the Vrije Universiteit in the Netherlands, and the Italian Universita' del Piemonte Orientale to support various research activities.

Performance sensors provide periodic measurements of different resource performance characteristics to the system via a well-defined protocol. Currently, we provide end-to-end TCP/IP bandwidth and latency, and available CPU capacity as "native" NWS sensors, but the system's sensor control facilities are easily extensible via the sensor interface protocol. Sensor development is currently underway for SNMP, UDP/IP, ICMP ping, Java RMI, Condor host availability, real memory, disk storage speed, and disk capacity.

Once gathered, sensor data is maintained by a set of NWS persistent storage servers and tracked via an internal name service. All internal NWS subsystems export well-defined access protocols so that emerging Grid technologies can be easily incorporated.

The forecasting facilities implemented by the NWS are both adaptive and extensible. Rather than relying on a single forecasting model, the NWS exports a well-defined forecasting interface allowing it to support a variety of models. When forecast data is required, the NWS invokes *all* of the configured models on a portion of the previous performance history and determines which model would have generated the most accurate forecasts. The "winning" model (in terms of forecast accuracy) is then chosen and used to generate an actual forecast for the required time frame. This method of using *postcast* accuracy to determine the best forecasting method "on-the-fly" has proved remarkably useful. It allows the system to incorporate rigorous forecasting methods from economics [12], digital signal processing [16], and control theory [19] and apply them to "live" Grid performance data. For each resource and each time frame, the system automatically identifies the best technique [38].

Figure 5 depicts a typical forecasting scenario [41]. The lefthand graph shows a time series of bandwidth measurements taken over a four day period between the University of California, Santa Barbara (UCSB), and Kansas State University (KSU). Each measurement represents the observed TCP/IP bandwidth (measured in megabits/second) from UCSB to KSU when transferring 64K bytes in total. In the righthand graph, we show the NWS forecasts for the corresponding time frame. Even though the measurement data has a noticeable diurnal cycle, the NWS is able to effectively identify the trend in the data and track it. The NWS is able to obtain similar results for available CPU capacity as well [42].

In this proposal, we will draw heavily from our NWS research experiences. The NWS enables schedulers and Grid users to anticipate future resources performance levels. We will use it to construct EveryWare program schedulers that make decisions based both on available resource performance, and the current resource price. The NWS will also serve as the backbone technology for gathering and maintaining the resource supply and demand information necessary to form a computational economy.

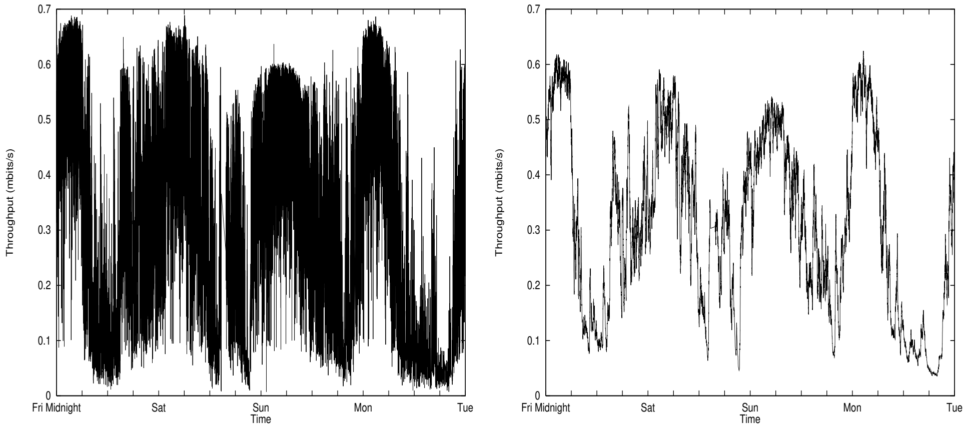


Figure 5: The left graph shows four days of bandwidth measurements between UC Santa Barbara and Kansas State University. The right graph shows the corresponding NWS forecast values.

5.2 Grid Application Development Software (EIA-9975020)

The Grid Application Development Software (GrADS) project attempts to define a comprehensive program development and execution environment for the Grid. Compilation, advanced library development, dynamic runtime systems, and on-line performance analysis research results are combined and applied to the problems of Grid computing. The GrADS "soup-to-nuts" approach attempts a more comprehensive scope than previous Grid research projects.

Of particular relevance to our work is the GrADS notion of "performance contract." [**?**] Applications and resource providers share an abstraction that specifies the work to be performed on behalf of the application's user, and allows a resource provider to claim the specified work. Our research plan for EveryWare and G-commerce complements the GrADS effort by providing the economic and implementation framework within which these performance contracts may be negotiated.

5.3 SInRG: The UTK Scalable Intracampus Research Grid (EIA-9972889)

The Computer Science Department at the University of Tennessee is cultivating a core Grid research competency. Led by professors Dongarra, Plank, Wolski and Beck in collaboration with other researchers in various disciplines across the campus (e.g. computational biology, medicine, chemical engineering) this effort resulted in funding from NSF to build a **S**calable **In**tracampus **R**esearch **G**rid (**SInRG**)[**?**]. The purpose of SInRG is to provide a distributed, heterogeneous collection of computation, storage and networking elements spread across the campus that supports both Grid systems research and which encourages a Grid application community. SInRG will provide the critical resources and "live" resource load necessary to verify EveryWare and G-commerce results.

References

[1] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application level scheduling on distributed heterogeneous networks. In *Proceedings of Supercomputing 1996*, 1996.

[2] The bovine rc5-64 project – `http://distributed.net/rc5/`.

[3] 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.

[4] The concurrent systems architecture group – `http://www-csag.ucsd.edu/`.

[5] D. Culler, A. Arpaci-Dusseau, R. Arpaci-Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong. Parallel computing on the berkeley now. In *to appear in JSPP'97 (9th Joint Symposium on Parallel Processing)*, 1997. available from `http://now.CS.Berkeley.EDU/Papers2`.

[6] L. DeRose, Y. Zhang, and D. Reed. Svpablo: A multi-language performance analysis system. In *Proceedings of 10th International Conference on Computer Performance Evaluation*, September 1998.

[7] M. P. I. Forum. Mpi: A message-passing interface standard. Technical Report CS-94-230, University of Tennessee, Knoxville, 1994.

[8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1998.

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[11] J. Gosling and H. McGilton. The java language environment white paper, 1996.

[12] C. Granger and P. Newbold. *Forecasting Economic Time Series*. Academic Press, 1986.

[13] J. Gregor and D. Huff. A computational study of the focus-of-attention em-ml algorithm for pet reconstruction. *Parallel Computing*, 24:1481–1497, 1998.

[14] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds. Legion: The next logical step toward a nationwide virtual computer. Technical Report CS-94-21, University of Virginia, 1994.

[15] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[16] R. Haddad and T. Parsons. *Digital Signal Processing: Theory, Applications, and Hardware*. Computer Science Press, 1991.

[17] G. Herman, A. Lent, and S. Rowland. Art: Mathematics and application. *Journal of Theoretical Biology*, 42:1–32, 1973.

[18] The high-performance computing challenge at sc98 – `http://www.supercomp.org/sc98/hpcc/`.

[19] V. Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, volume 18, August 1988.

[20] The java servlet api from the sun microsystems javaserver product group – `http://jserv.javasoft.com:80/products/java-server/documentation/webserver1.0.2/servlets/api.html`.

[21] Joint institute for computational science at the university of tennessee – `http://www-jics.cs.utk.edu`.

[22] L. Kaufman. Maximum likelihood, least squares, and penalized least squares for pet. *IEEE Transactions on Medical Imaging*, 12:200–214, 1993.

[23] K. Lange and R. Carson. Em reconstruction algorithms for emission and transmission tomography. *Journal of Computer Assisted Tomography*, 8:306–316, 1984.

[24] A. Lenstra and M. Manasse. Factoring by electronic mail. In *Advances in Cryptology – EUROCRYPT '89*, pages 355–371, 1990.

[25] J. M. M. Ferris, M. Mesnier. Neos and condor: Solving optimization problems over the internet. Technical Report ANL/MCS-P708-0398, Argonne National Laboratory, March 1998. available from `http://www-fp.mcs.anl.gov/otc/Guide/TechReports/index.html`.

[26] B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tools. *IEEE Computer*, 28(11):37–46, Nov. 1995.

[27] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi. Utilizing the metaserver architecture in the ninf global computing system. In *High-Performance Computing and Networking '98, LNCS 1401*, pages 607–616, 1998.

[28] The network weather service home page – `http://nws.npaci.edu`.

[29] "OMG". The complete formal/98-07-01: The corba/iiop 2.2 specification, 1998.

[30] S. Radziszowski. Small ramsey numbers. In *Dynamic Survey DS1 – Electronic Journal of Combinatorics*, volume 1, page 28, 1994.

[31] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed. Autopilot: Adaptive control of distributed applications. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, Aug 1998.

[32] The sc conference series – `http://www.supercomp.org`.

[33] Seti at home web page `http://www.seti-inst.edu/science/setiathome.html`.

[34] N. Spring and R. Wolski. Application level scheduling: Gene sequence library comparison. In *Proceedings of ACM International Conference on Supercomputing 1998*, July 1998.

[35] T. Tannenbaum and M. Litzkow. The condor distributed processing system. *Dr. Dobbs Journal*, February 1995.

[36] The tera mta – `http://www.tera.com`.

[37] Microsoft windows nt operating system – `http://www.microsoft.com/ntserver/nts/techdetails/overview/wpglobal.asp/`.

[38] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1998. also available from http://www.cs.utk.edu/ rich/publications/nws-tr.ps.gz.

[39] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Running everyware on the computational grid. In to appear *SC99 Conference on High-performance Computing*, April 1999. available from `http://www.cs.utk.edu/~rich/papers/ev-sc99.ps.gz`.

[40] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Writing programs that run everyware on the computational grid. Technical Report UT-CS-99-420, The University of Tennessee, April 1999. available from `http://www.cs.utk.edu/~rich/papers/ev-results.ps.gz`.

[41] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems (to appear)*, 1999. available from `http://www.cs.utk.edu/~rich/publications/nws-arch.ps.gz`.

[42] R. Wolski, N. Spring, and J. Hayes. Predicting the cpu availability of time-shared unix systems on the computational grid. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*, 1999. available from `http://www.cs.utk.edu/~rich/publications/nws-cpu.ps.gz`.