

MESH GENERATION AND OPTIMISTIC COMPUTATION ON THE GRID*

Nikos Chrisochoides
College of William and Mary
nikos@cs.wm.edu

Craig Lee
The Aerospace Corporation
lee@aero.org

Bruce Lowekamp
College of William and Mary
lowekamp@cs.wm.edu

Abstract

This paper describes the concept of optimistic grid computing. This allows applications to synchronize more loosely and better tolerate the dynamic and heterogeneous bandwidths and latencies that are seen in grid environments. Based on the observed performance of a world-wide grid testbed, we estimate target operating regions for grid applications. Mesh generation is the primary test application where boundary mesh cavities can be optimistically expanded in parallel. To manage the level of optimistic execution and stay within the application's operating region, we are integrating grid performance monitoring and prediction into the supporting runtime system. The ultimate goal of this project is to generalize the experience and knowledge of optimistic grid computing gained through mesh generation into a tool that can be applied to other tightly coupled computations in other application domains.

Keywords: mesh generation, grid computing, optimistic computing, performance analysis

*This work is supported in part by the National Science Foundation under the grant EIA-0203974.
Appears in *Performance Analysis and Grid Computing*, Kluwer, October 2003.

1. Introduction

Computational grids offer a vast pool of computational and storage resources which can be utilized by large-scale engineering and scientific computing applications. The *efficient* utilization of these vast resources, however, can be difficult. Grids provide an environment where resources *and* performance can be dynamic and heterogeneous. It is clear that loosely coupled applications will be naturally “grid-friendly”. However, grids offer a way of aggregating more compute power than can be economically done any other way. Hence, there is strong motivation to find ways that enable more tightly coupled applications to effectively utilize grid resources.

To do this, a tightly coupled application must be able to “loosen-up” just enough to tolerate prevailing conditions. This “loosening-up” can be accomplished by relaxing strict synchronization and communication requirements, wherever possible, and by tolerating low bandwidths and high latencies, however possible. Ideally an application should also be able to tell how much “loosening-up” is enough by acquiring the appropriate performance information about the environment and itself. In general, an application’s computation-communication ratio should match what the environment can support as much as possible. Many established techniques exist that could be applied here. Such techniques include: partitioning to control the surface area to volume ratio, using clumps (SMP clusters) to increase a host’s “cycle density”, overlapping communication and computation, pipelining data communication, using shadow arrays or ghost zones, using aggregate communication, using data compression, and tuning the communication protocol.

These issues can also be addressed by *restructuring* or *reconceiving* an application, perhaps even using a different *execution model*. As an example, hierarchical n-body methods achieve much faster speeds than traditional all-to-all methods by allowing accuracy to be traded for performance within a certain bound. Different execution models could include coarse-grain dataflow models, such as workflow or stream programming models, and also *optimistic* or *speculative execution* models.

This paper reports on a new project to investigate all of these issues, and specifically *optimistic grid computation*. *Mesh generation* will be used as the test application. Mesh generation is an integral part of many important scientific computing applications and is a memory-intensive application with major impact on the overall performance of the end-to-end field simulations that could benefit greatly from a grid platform. Using traditional approaches for generating, partitioning, and placing very large meshes on a grid have two weaknesses: (i) I/O and data movement due to mesh re-partitioning, for adaptive applications, is prohibitively expensive, and (ii) there is a trade-off between the quality of the resulting elements and partitions and the performance of field solvers.

Hence, we have developed methods whereby mesh generation can be “loosened-up” by using *speculative* or *optimistic* execution. This presents three challenges:

- 1) *Maintain high-quality of the solution*, i.e., elements and partitions in the case of parallel meshing, in order to guarantee high-performance for the end-to-end parallel simulation,
- 2) *Control the optimistic execution* such that *optimal optimism* is achieved, i.e., prevent excessive optimism whereby too much work must be discarded and any performance benefits are lost, and
- 3) *Provide a runtime infrastructure and appropriate performance metrics* whereby the proper control decisions can be made.

In this project, we will address these challenges by performing the following three tasks using an end-to-end, mesh generation application from fracture mechanics:

- Develop a runtime infrastructure that enables an application to interact with its performance environment while facilitating quick prototyping, experimentation, and evaluation.
- Explore how optimistic execution and other techniques must be controlled to scale-up and enable an application to work in its grid “operating region”.
- Develop and evaluate new network metrics and performance models that cover a wide spectrum of communication characteristics using mesh generation methods to implement properly grid-aware applications.

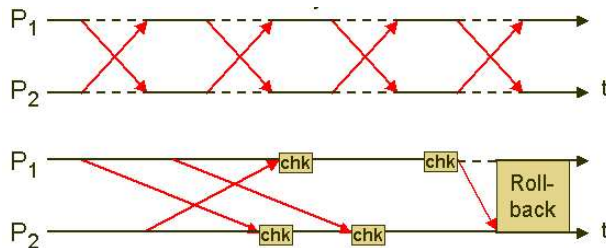


Figure 1. Cost trade-offs between strict control and optimistic control.

In the next section, we will introduce the concept of optimistic computation and the behavior that an application must exhibit to benefit from it. This application behavior is tied to the concept of an “operating region” for grid applications based on “keeping the pipes full” and matching the computation/communication ratio to the environment. We then introduce parallel mesh generation methods which can be reconceived as optimistic computations. These methods can be supported by a specialized run-time system described in Section 4 that incorporates grid performance monitoring. We also describe the testbeds that will be used to evaluate these mesh generation methods and optimistic grid computation, in general. We conclude in Section 5.

2. Optimistic Computation for Grid Environments

2.1 General Optimistic Computation

Optimistic parallel computation is very similar to optimistic parallel simulation [19]. In that work, multiple end-hosts simulate in parallel and exchange time-stamped events. If a host receives an event with a time-stamp earlier than its current simulated time, i.e., “in its past”, then it must *roll-back* its simulation to the earlier time and start again. This requires that an end-host do *incremental state saving* to enable roll-backs, and possibly send *anti-messages* that undo the effect of messages sent to other end-hosts during the rolled-back period.

The critical trade-off that determines the advantage of optimistic simulation is the overhead of roll-backs (frequency and severity) versus the benefit of more loosely coupled, parallel simulations that have reduced synchronization and communication delays. This is a fundamental trade-off that applies to all optimistic computation. Optimistic execution can reduce synchronization and communication requirements but allows *inconsistent* results to be computed. What is considered inconsistent, of course, is application-dependent. Whatever may be considered inconsistent, an application must detect and correct inconsistencies such that (1) a correct final result is produced, or (2) whatever error may exist in the solution is limited to within some acceptable bound. Again, it is application-dependent whether strict correctness is required or whether bounded error is acceptable.

This trade-off is illustrated in Figure 1. A strictly synchronized application has some overhead due to this synchronization and communication control. An optimistic computation will have some overhead for consistency checking using *validation messages*. Since consistency validation can proceed independently, these messages can be sent asynchronously and also be pipelined. When an inconsistency is detected, however, roll-back must occur which could be a more heavyweight and synchronous operation. For optimistic computation to be advantageous, the following inequality must hold:

$$\text{Cost}_{\text{validation}} + \text{Cost}_{\text{roll-back}} \ll \text{Cost}_{\text{strict-control}} \quad (1)$$

The costs of validation and roll-back should be much less than the cost of strict control. If the costs are similar, or greater, then there will be no advantage, or even a disadvantage. Cost will typically be defined in terms of execution time but this cost could be broken down into communication time, synchronization delays, and also discarded work.

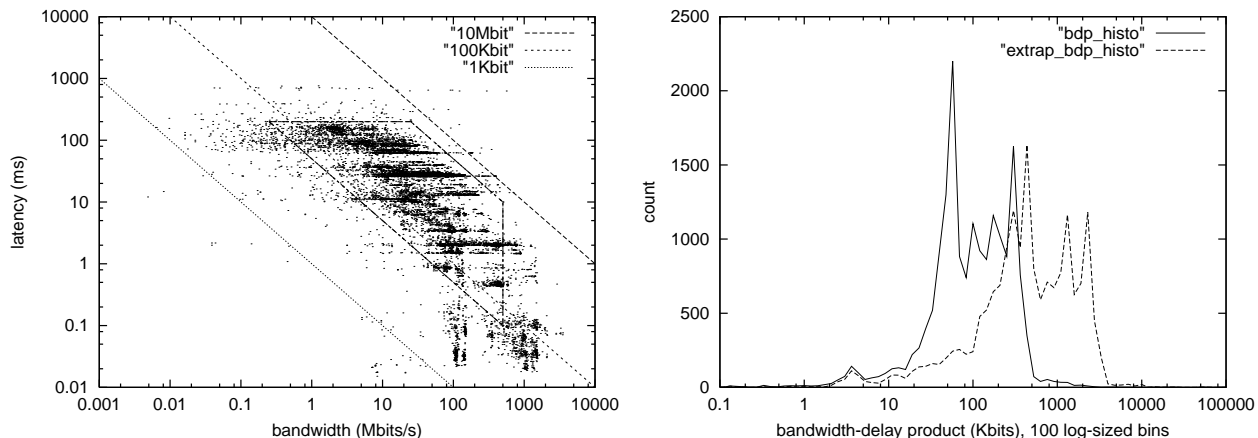


Figure 2. Left: Extrapolated Round-Trip Time latency and Bandwidth with operation region. Right: Bandwidth-Delay Product histograms, both measured and extrapolated.

A key issue for this work is “how much optimism is enough”? Optimistic application must have some type of independent parameter, or “control-knob”, that determines the amount of optimism. Clearly over-optimistic execution will suffer excessive roll-backs while under-optimistic execution will reduce parallel performance. Besides these application-specific considerations, an application should only generate *just enough* optimism to match the prevailing grid conditions. To understand this issue, we define the notion of an *operating region* for grid applications.

2.2 The Operating Region for Grid Applications

What properties must an application exhibit to effectively utilize grid resources under the prevailing grid performance among these resources? We can characterize this question on the macro-scale by examining actual performance data from a world-wide grid. As reported in [24], a snapshot of a global grid’s performance was used to derive the probable *operating regions* for grid applications based on the number of concurrent threads of computation, size of a “work unit”, and the prevailing bandwidths and latencies. This data snapshot captured 17629 unique bandwidth and round-trip latency measurements between 3158 unique host-pairs over 138 unique hosts on four continents. Based on the estimated improvement in processor speeds and network bandwidth in the next ten years, these data were extrapolated to show the expected bandwidths and latencies that will be available in 2010. These data are shown as a scatterplot in Figure 2(left) which shows the *density* of available performance. The three diagonal lines on this graph show the position of three different bandwidth-delay products: 1 Kbit, 100 Kbit, and 10 Mbit. This illustrates the number of bits that could be “in-flight” or “in the pipe” between any source and destination hosts. Figure 2(right) shows the extrapolated increase in the distribution of bandwidth-delay products.

These results show that since processor speed and bandwidth in a global grid will be increasing, while propagation delays remain constant, that the bandwidth-delay products will be increasing dramatically, i.e., communication “pipes” will be getting “fatter” but not commensurately “shorter”. Based on the expected performance density in Figure 2a, we can draw bounds on where applications should typically operate. Based on the bandwidth, latency and bandwidth-delay products that form this trapezoidal region, we can estimate the *operating region* for grid applications, based on the goals of matching the bandwidth-delay product (“keeping the pipe full”) and matching the computation/communication ratios.

These general operating regions are illustrated in Figure 3. Figure 3(left) illustrates the operating region for keeping the pipes full where an application must generate enough independent threads of computation (here 1 to 100) of a given work grain size (in bits) to fall within a given range of bandwidth-delay products (defined by the observed range of bandwidth-delay products in Figure 2). Figure 3(right) illustrates the op-

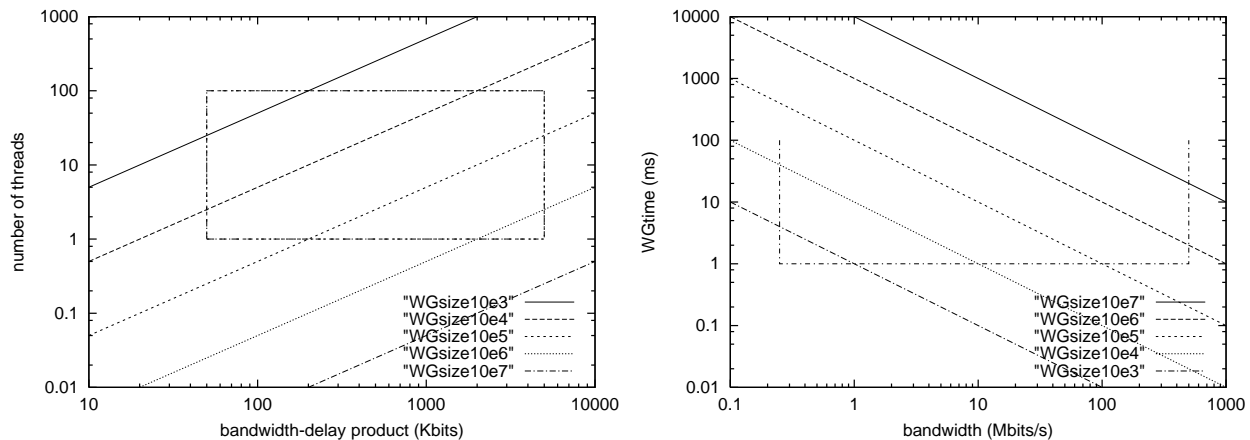


Figure 3. Left: Feasible operating region based on a reasonable number of threads of control and expected bandwidth-delay products. Right: Work Grain time and expected bandwidth.

erating region for matching the computation/communication ratio where the computation time represented by a work grain (in milliseconds) must match the grain’s communication time based on its size (in bits) and the available bandwidth (defined by the observed range of bandwidths in Figure 2). (This is only bounded from below since larger work grains will more easily fill the pipe.)

These operating regions are based on a simple, pipeline model of execution and, hence, are only rough estimates since they elide many application issues. Explicit synchronization and data dependency issues are ignored which can clearly affect an application’s ability to “hit” the operating region. Most applications will also have a distribution of work grain sizes and computation times, rather than a single, fixed value. Nonetheless, these operating regions capture the fundamental behavior that applications must exhibit and give us a target for evaluating application performance under optimistic execution.

3. A Case Study: Optimistic Mesh Generation

Whether grid applications can in general be effectively scaled-up to work in these operating regions is an open issue. In essence, the *range of bandwidths and latencies* that an application must tolerate will be greater than in any other environment. Clearly, to be grid tolerant, applications will have to be designed such that they are, in effect, less tightly coupled. In the case of mesh generation, the Parallel Bowyer-Watson method offers an ideal test case. The *Optimistic Delaunay (OD)* approach effectively decouples adjacent mesh regions by allowing a processor to *asynchronously* proceed with remote cavity expansion. The drawback is that some optimistic cavity expansions will be incompatible and will have to be *rolled-back*. Of course, the goal will be to minimize the roll-backs and maximize the net performance gain for the application. While we will be studying optimistic grid computation in the context of mesh generation, the ultimate goal is to quantitatively understand these techniques and generalize them to other application domains.

3.1 Parallel Mesh Generation

Parallel mesh generation methods decompose the original meshing problem into smaller subproblems that can be solved (i.e., meshed) in parallel. The requirements for the solution of the subproblems on grid environments are: (1) *stability*, distributed meshes should retain the good quality of elements and partition properties of the sequentially generated and partitioned meshes, (2) *tolerance of long, variable, and unpredictable latencies*, (3) *scalability* (4) *fault-tolerant*, and (5) *code re-use*, in order to leverage the ever evolving and maturing basic scalar meshing techniques.

This project is focused on the design and implementation of a *stable, scalable, and latency tolerant* mesh generation methods for grid environments. Fault-tolerance and code re-use are equally important issues

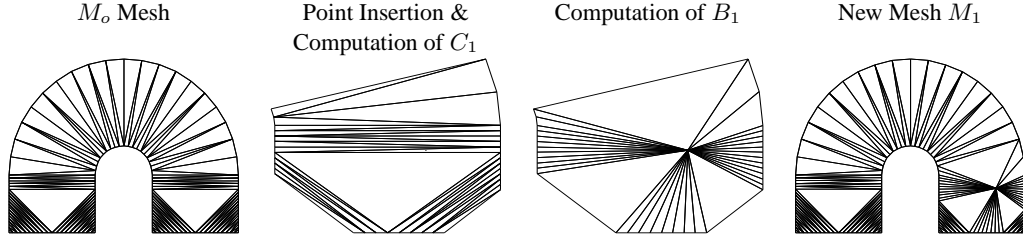


Figure 4. Point insertion and element creation steps of the BW algorithm; initial and final meshes at each iteration shown at the leftmost and the rightmost columns respectively.

and will be addressed in subsequent projects. Mesh *stability* is an important requirement for the accuracy and cost effectiveness of grid-aware Partial Differential Equation (PDE) simulations. *Latency tolerance* is important for the *scalability* of parallel applications, because communication and synchronization latency is the performance bottleneck on grid environments.

3.2 Mesh Generation Methods

We will focus on Delaunay triangulation methods [2, 17, 29] because they can guarantee mathematically the quality (and thus the stability) of the triangles [31, 8] and tetrahedra [8, 34]. By definition a triangulation T is called Delaunay if for each element $e \in T$ the open circumscribed sphere of e is empty i.e., none of the mesh points of the mesh are contained within the circumcircle (or circumsphere) of triangles (or tetrahedra) of the mesh. This criterion is called the *empty sphere* or *Delaunay* criterion.

The Delaunay criterion has been used successfully, for sequential mesh generation of complex geometries, since the late 80's. There are many different Delaunay triangulation methods [17], but the most popular Delaunay meshing techniques are the incremental methods. Incremental methods start with an initial mesh (usually a boundary conforming mesh, see mesh M_o , Figure 4) which is refined incrementally by inserting new points. Each new point is re-connected with existing points of the mesh in order to form a new mesh. The difference between the various Delaunay incremental algorithms is due to: (1) different spatial point distribution methods for creating the new points and (2) different local reconnection techniques for creating the triangles or tetrahedra.

The two most popular local re-connection methods are the flip edge/face methods [23] which are difficult and expensive to parallelize [9] and the Bowyer-Watson (BW) kernel [4, 37]. The BW kernel is an iterative procedure: at each iteration, an existing Delaunay mesh, M_i is refined and thus a new M_{i+1} mesh is generated by inserting a new point p_i into M_i after recovering the Delaunay property of the mesh M_i through the local transformation: $M_{i+1} = M_i - C_i + B_i$. Figure 4 depicts the 1st iteration of the BW kernel.

The implementation of the guaranteed quality BW kernel, for 3-dimensional geometries, is not an easy task [34], there are very few codes and none in the public domain to the best of our knowledge. The task of implementing a stable and latency tolerant parallel version of the BW (PBW) kernel is one of the key challenges of this project.

In the PBW kernel, each processor concurrently refines the mesh by inserting new points and re-triangulating their cavities. A pair of concurrently expanding cavities are related to each other in two possible ways:

$$\text{Case I: } C_p \cap C_q = \emptyset, p \neq q \quad (2)$$

The cavities do not intersect and thus they can be re-triangulated concurrently. Without loss of generality consider an initial Delaunay mesh, M_o , and two new points p, q within the convex hull of M_o such that $C_p \cap C_q = \emptyset$. Then by the BW kernel and the fundamental Delaunay Lemma [17] we have that the new meshes $M_{p,q}$ and $M_{q,p}$ are Delaunay triangulations and they are the same: $M_{p,q} = M_p - C_q + B_q =$

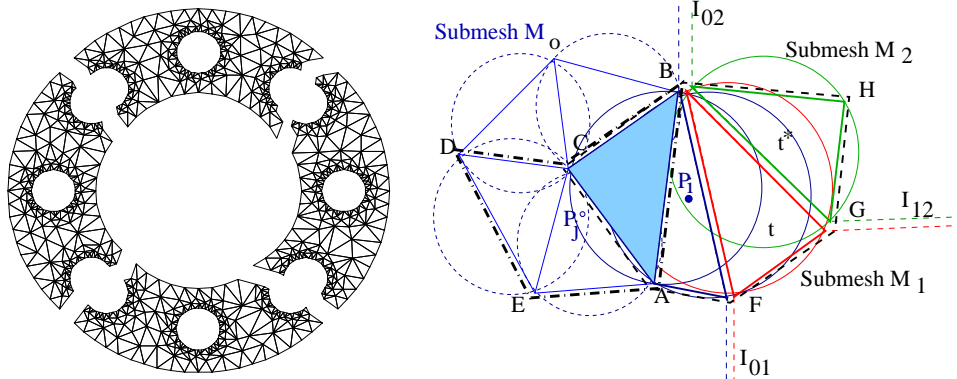


Figure 5. Left: Submeshes of a triangulation and their inter-submesh interfaces. Right: Cavity expansion over more than one submeshes. The tetrahedra $t \in M_1$ and $t^* \in M_2$ are non-Delaunay w.r.t point $P \in M_0$; thus the cavity C_P is an interfaces cavity. Dashed line show that inter-submesh interfaces.

$(M_o - C_p + B_p) - C_q + B_q = (M_o - C_q + B_q) - C_p + B_p = M_q - C_p + B_p = M_{q,p}$ —as long as the points in $M_o \cup \{p, q\}$ are in general position— and thus the points p and q can be inserted concurrently.

$$\text{Case II: } C_p \cap C_q \neq \emptyset, p \neq q \quad (3)$$

The cavities intersect and have to be re-triangulated in a way that non-conformity (see Figure 5(Right)) and/or instability issues like non-Delaunay mesh generation are prevented. In the case $C_p \subset S_i$ for some p , and i , the cavities are computed and triangulated automatically. Therefore the case $C_p \cap C_q \neq \emptyset, p \neq q$ it takes place only if a cavity C_p of a submesh say S_i intersects with another submesh S_j , in this case the cavity expansion interrupts and the cavity remains active (i.e., locks all of its elements) until non-local elements (using remote gather operations) are collected. We refer to these cavities as active interface cavities. While an active interface cavity is expanding remotely the control is released to another local cavity expansion, C_q , (i.e., thread) and thus the cavity C_q might intersect with the cavity C_p .

There are four different approaches to solve the problem of generating unstable (i.e., non-conforming or non-Delaunay) meshes in the context of concurrency: (i) allow remote cavity expansion but impose a partial order between intersecting cavities using a parallel Optimistic Delaunay (OD) meshing, (ii) mathematically guarantee the uncoupling of submeshes so that non-interface and non-intersecting cavities are computed first and compute interface cavities at the end; this leads to the parallel Delaunay Uncoupling (DU) meshing method, (iii) instead of expanding interface cavities use Constrained Delaunay Triangulation (CDT) techniques, and (iv) ignore interface cavities and compute only the local portions of the cavities, this leads to a non-Delaunay mesh; then using Domain Decomposition (DD) one can reduced the parallel meshing problem into many unrelated sequential problems. Clearly, these four methods vary in terms of stability and communication characteristics which make them an interesting test-case for grid environments.

The main focus of this project is the OD method. The OD method proceeds with the remote cavity expansion, but in order to avoid generating a non-conforming mesh and/or a conforming but non-Delaunay mesh, it imposes a partial order on the triangulation of interface cavities. The remote cavity expansion of interface cavities is a source of very large communication overhead as well as variable and unpredictable communication latencies due to remote data gather operations. Figure 6 shows the time and message size distributions for generating meshes for a ‘‘Tee’’ geometry on a cluster computer with 16 processors. The per-node time distribution for meshing operations is given for generating a mesh of two million elements. The message size distribution is given for generating meshes of three different sizes, including two million elements. The PBW algorithm in its effort to tolerate the long and variable communication latencies of remote interface cavity expansions, uses an *optimistic* or *speculative execution model* i.e., it inserts new

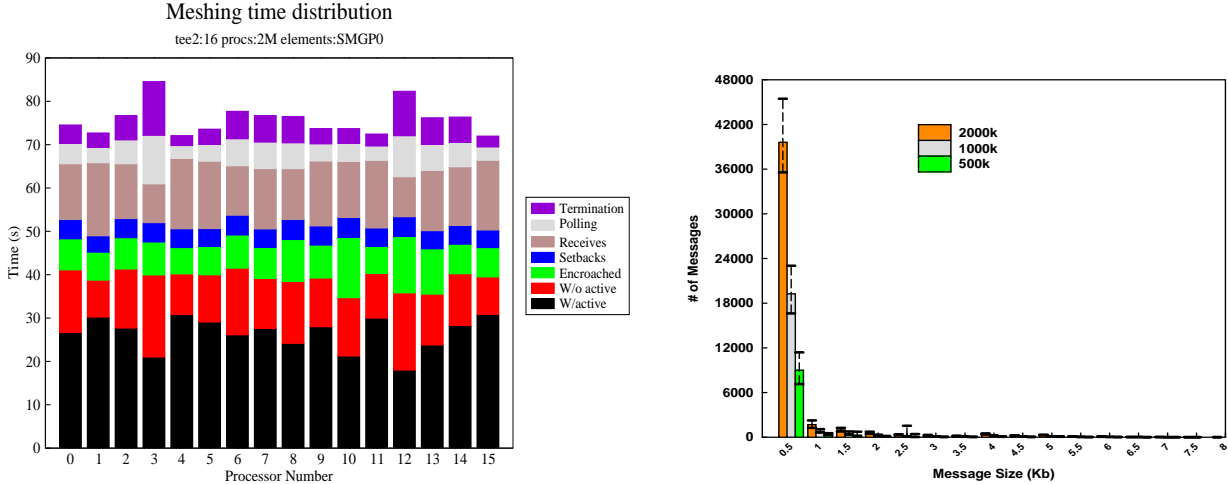


Figure 6. Left: Time breakdown, in percentages, incurred by the PBW algorithm using the OD method, for generating two million element meshes of a 3-dimensional Tee geometry on a 16 computers connected through Fast-Ethernet. Right: Distribution of message sizes for generating 3-dimensional Tee geometry meshes of three different sizes.

points on demand and expands their cavities — otherwise it would be waiting for the completion of remote part of interface cavity expansions. However, for stability reasons, some of the cavities terminate before they re-triangulate and free all of their elements. We call this form of a roll-back in the cavity computation a **setback** in the progress of the PBW kernel.

Such setbacks or roll-backs are a fundamental issue in all optimistic or speculative computational techniques. That is to say, optimistic execution can provide a significant gain in performance only if setbacks do not become excessive. When faced with poor communication (as in a grid), optimistic execution allows a computation to proceed in a less synchronous manner, thus enabling higher parallelism and utilization. If excessive optimism produces excessive setbacks, however, then any gains in performance may be lost.

Hence, a major goal of this project is to quantitatively study the use and control of *optimal optimism* in grid computations, i.e., optimistic execution that does not excessively waste cycles in communication and synchronization delays but also does not excessively waste cycles in setbacks that represent discarded work. We will do this by modifying the PBW kernel such that we can control the degree of optimism and setbacks — which depend on the number of outstanding interface cavities, and on newly inserted points in the presence of interface cavities. Determining *where* this optimal optimism occurs, however, is a major challenge. More importantly this control of optimism in PBW will be driven, in part, by the prevailing network conditions. That is to say, just enough optimism will be used to increase parallelism and utilization and overcome insufficient communication bandwidth and latency. For grid applications, such as mesh generation, the *operating region* concept defined in the previous section can be used to determine how optimistically the application can and should be. In essence, a grid application must provide just enough parallelism to keep the network pipes full *and no more*. Beyond this point, an application must be able to execute more asynchronously, or optimistically. Hence, we will use experimental network performance monitoring techniques to gain the information necessary to control the level of optimism in PBW and the demand for communication such that its overall performance remains in its operating region.

4. Supporting Optimistic Computation on the Grid

Enabling an optimistic application such as mesh generation to execute effectively in a grid environment will require some *control mechanism* that can (1) monitor the current grid performance, and (2) interact with the application to help it reach the operating region. To this end, we now describe a *runtime system*

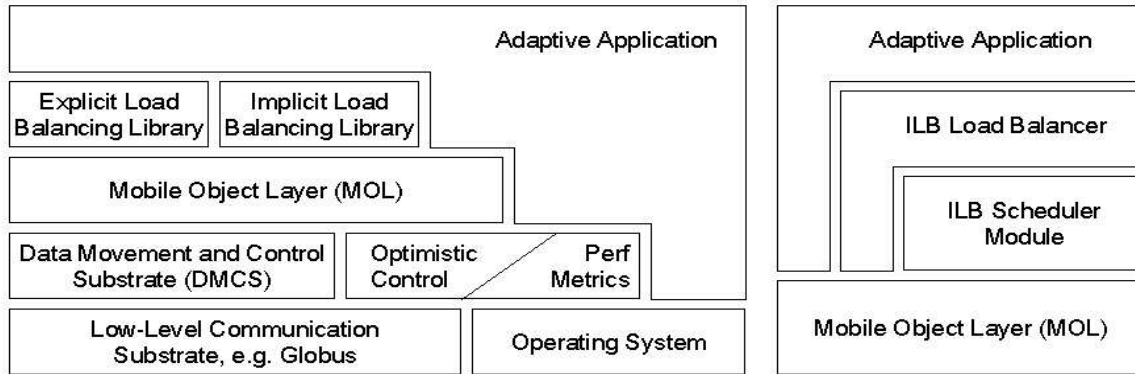


Figure 7. Runtime System Architecture

that will contain a control module for optimistic execution that utilizes current performance monitoring information. We also describe the testbeds that will be used to evaluate the potential benefits of optimistic grid computation.

4.1 The Runtime System

The architecture of the runtime system we are using is shown in Figure 7. This leverages existing work on the Portable Runtime Environment for Adaptive Applications (PREMA) for the Crack Propagation Project [6] and Remos [11, 26, 25]. We have ported the Data Movement and Control Substrate (DMCS) [13] on top of Grid communication substrates like those provided as part of Globus [35, 15]. The Application Program Interface of the Mobile Object Layer (MOL) is also being extended in order to permit meaningful applications queries on the characteristics and condition of the underlying Grid environment. Such queries will be performed by higher runtime system support libraries like Implicit Load Balancing (ILB) and Explicit Load Balancing libraries.

This will allow us to explore two approaches regarding the utilization of grid performance information: (1) an explicit approach providing an API to the application to directly use performance information, and (2) an implicit approach where a control module uses the information to control aspects of the application, such as the automatic scheduling of applications tasks. Specifically an *Optimism Control module* will monitor the current application behavior (e.g., number and size of messages, task compute times) and current grid performance (e.g., achievable bandwidths and latencies among end-hosts in use). Based on the operating region model described earlier, the control module will then be able to determine the most appropriate level of optimism and inform the application accordingly.

This runtime architecture approach will have several additional important benefits:

- *Ease-of-Use*: For applications written using the programming model provided by the MOL, the move from clusters and MPPs to grid environments should be a straight-forward evolutionary step.
- *Flexibility*: The runtime should provide a load balancing interface which will be used with a wide variety of scheduling algorithms. Figure 7 depicts the Scheduler module in the overall architecture. By replacing the Scheduler, applications we will be able to customize the overall load balancing policy without needing to modify any application code. This allows application developers to quickly and easily experiment in order to find the most effective load balancing method for a given grid environment.
- *Low Overhead*: The overhead incurred by the runtime system will be kept as low as possible. While an application may implement any desired scheduling algorithm, the scheduler should not have to conform to an interface that precludes high performance.

4.2 Performance Monitoring

The research described in this section focuses on what information is required, how resource information is now gathered, and the new developments that must be made to fully meet the needs of optimistic grid computation.

4.2.1 Network Measurements. In a general sense, we are interested in the bandwidth and latency of the portions of the network connecting the resources forming the grid. Specifically, we are interested in the end-to-end latency of each possible connection as well as capacity and availability of the paths between endpoints.

The simplest measurement technique is application level, where the performance of the network is measured by the performance an application receives. Typical application-level approaches rely on actively generating a TCP flow and monitoring its performance [36, 38, 27]. The result is time-averaged available bandwidth over the interval of the communication.

Packet train approaches attempt to measure more fundamental characteristics of the network traffic [7, 22, 21, 20]. Rather than send an actual data stream, a series of packets, ranging from two to tens of packets, are sent between endpoints as quickly as possible. Based on their separation at arrival time, properties of the path at the time that train was sent can be determined. Although this method is most easily used to determine capacity, some information about the link availability may also be obtained.

Either approach can be used in either an active or a passive form. Systems have been developed to passively monitor entire application streams [33, 10]. They have also been developed to monitor for individual packet pairs exchanged by actual applications, rather than generating artificial packet pairs [21].

4.2.2 Predicting Application Performance. The key issue is predicting an application's performance. While past performance is of significant interest to application performance tuners or network engineers, for grid-based distributed applications short-term prediction is the most important feature. Applications with lifetimes of minutes or hours need predictions over their execution, or even just the next few timesteps.

Performance prediction for grid applications is typically done using time-series models [3, 39, 12, 25, 5]. The output of a time series model is a series of predictions for the expected value of the metric for the future. Equally important, however, is the variance and error in the signal. In particular, for fine-grained applications, the short-term variance in the latency or available bandwidth can be more important than the expected value.

What is the relationship between the metric and the application's performance? In a simple case, the metric may measure exactly what the application does. For example, if an application makes a bulk TCP data transfer across the network, then a simple bulk TCP measurement will be completely accurate. However, using bulk TCP measurements to predict the loss rate of streaming video is quite challenging, and rarely reflect performance of short-term applications [14].

What metrics are needed to predict an application's performance? The shorter the message sent by an application, the more fine-grained the measurement information must be to accurately predict the performance that particular message will receive. Knowledge of the short-term variance in message latency will be important to properly tuning a "loosened-up" application for its execution environment. Furthermore, the performance that an application receives from the network is heavily influenced by how that application uses the network. An application that sends a continuous stream of data may receive different service from TCP than an application that sends only occasional bursts of data, as the bursty application may never force the TCP window open enough to fully take advantage of the bandwidth available on the network.

As the issue of finer-grained application and performance monitoring is approached, the question of traffic predictability appears. The self-similar nature of network traffic has caused some to label network traffic inherently unpredictable. The actual usefulness of predicting network traffic is somewhat more complex, as different network environments have extremely different types of cross traffic. For example, a router pro-

cessing very bursty WAN traffic may have its queue length vary from empty to overflowing over very short timescales. An Ethernet LAN may have a utilization under 5% 99.97% of the time, and be totally saturated the other 0.03% of the time as a machine performs a high-speed transfer. Neither of these environments has classically “predictable” performance, but it is possible to make meaningful statistical predictions about each environment over the timescale of our applications. It may be completely impossible to predict the short-term behavior of the bursty WAN, but over a longer time interval it may always be possible to transfer significant data across the WAN. Similarly, it may not be possible to determine when the bursts on the LAN will arrive, but an application may make use of the knowledge that only one out of every 3000 messages will experience congestion. A number of researchers have studied the statistical effect of congestion on TCP flows [32, 16–1], but the effect on individual flows remains an interesting research area.

Predicting application performance cannot be done properly unless predictions are available for the network behavior over the time-scale to which the application is sensitive [18]. Those predictions will only be available if metrics are selected that are capable of recording performance on the time-scale needed by the application. Once those metrics are available, the prediction service must be able to take the measurements and report both the expected value, but also the expected variability in the measurements.

For the parallel mesh generation problem, we will first rely on the models of the DO, CDT, DD, and DU implementations. The amount of data available per processor and the probability of successful speculative execution being rolled back dictate the latency that can be tolerated during data fetches (such as illustrated in Figure 6). With this knowledge, the variability of the network, and the variability of the communication pattern and bandwidth, a distribution of the latencies for the remote requests can be calculated. If this expectation exceeds the tolerable latency, then a different algorithm will have to be selected.

4.2.3 Proposed Network Measurement Approaches. Currently available systems and software do not provide the temporal resolution or variability information required for running fine-grained applications in grid environments. There are, however, measurement techniques that have been developed by the networking community to monitor network behavior that may be appropriate for use supporting these new applications.

We are pursuing three approaches for improving the granularity of network measurements and predictions. The first approach is to add additional instrumentation to currently used network monitoring tools, such as *iperf*. *Iperf* is already capable of reporting its performance at regular intervals, although at a coarse scale. However, much information is being lost in the TCP stack by only reporting user-level performance. By adding additional OS-level instrumentation, such as recording packet loss and delays introduced in packet pairs, we will collect more information without additional overhead because we will use same probes already in use by performance measurement systems. Collecting this information and making it available at user level through tools such as *NWS* and *Remos* will allow applications to benefit from the knowledge.

The second approach will be to integrate established packet dispersion-based measurement approaches (cite *nettimer*, *cprobe*, *pathchar*, *pathload*) into the toolchains of existing grid measurement systems such as *NWS* and *Remos*. Furthermore, we are exploring integrating passive approaches into these systems. By combining their existing active approaches with passive measurement options, we can maintain a constant stream of measurements for each path whether the application is currently using the path or not, without creating contention for that path and diminishing the performance of our application.

The third component of this approach is the process of making variability and confidence of prediction first-class information at the application level. In our previous experience developing the *Remos* toolkit, network variability was discussed, but never fully implemented. Similarly, *NWS* does not address this issue fully. We will propose modifications to existing measurement APIs to offer more detailed representations of metric variability and prediction confidence than is currently available. We will also enhance the ability of applications to request performance information over specific intervals, for example, specifying that it is interested in the variability of network performance at 250ms intervals for the next 20 minutes.

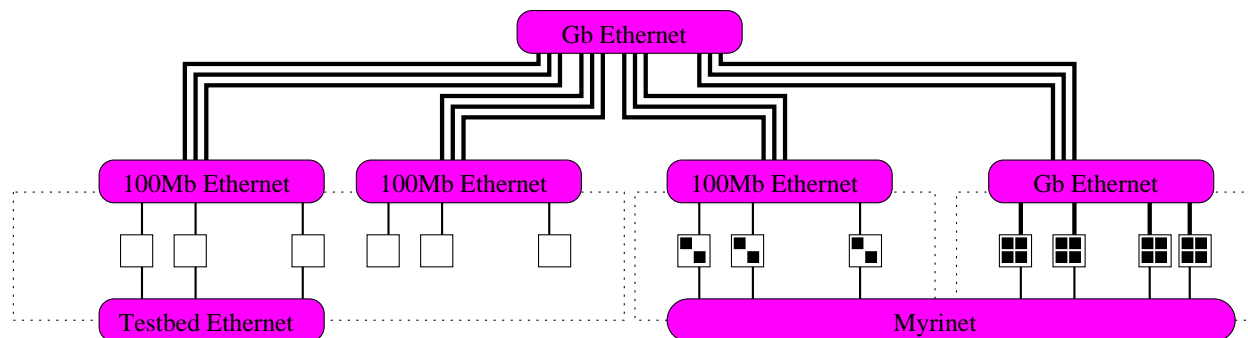


Figure 8. The SciClone cluster. Consists of 64 single processor, 32 dual processor, and 8 quad processor nodes tied together with a fat tree switched Ethernet, as well as Myrinet and a networking testbed. High-performance parallel file servers are also provided. SciClone’s testbed offers opportunities to experiment with the system under controlled conditions, while the cluster is a valuable grid computing resource in its own right for production experiments.

4.3 Evaluation Testbeds

4.3.1 The Cluster Testbed. Initial implementation and experimentation with the optimistic scheduling techniques described here are being performed on the networking testbed portion of the SciClone testbed, shown in Figure 8. The networking testbed provides a 32-node reconfigurable network with production endpoints on which the application can be run under a variety of conditions. This reconfigurability allows us to control the bandwidth and latencies observed by applications by using tools such as NistNet [28] and DummyNet [30].

After initial development on the testbed, the remainder of the SciClone cluster can be used for production runs and reconfigured for brief experiments with limited resources.

4.3.2 The Grid Testbed. The ultimate goal of this project is to demonstrate latency tolerant techniques for applications, such as mesh generation, in a grid environment. After initial development and testing have been done in a more controlled cluster environment, we will proceed with testing in a real-world grid. The exact configuration of the grid resources used is flexible. The general plan, however, is to use grid resources that provide a developmental sequence:

- *Within One Institution:* Using an internal grid for initial testing has the advantage that all resources are co-located and in the same administrative domain. While the number of resources on the grid will be small and close together, our test applications will have to contend with competing traffic for both bandwidth and cycles.
- *Within the Continental U.S.:* A more widely distributed environment will force applications to contend with general Internet traffic.
- *World-Wide:* Institutions from different continents will provide a truly world-wide latency environment in which to demonstrate the potential performance benefits for optimistic computations such as mesh generation.

The goal of this work is to demonstrate superior performance for grid applications using our techniques. To achieve this goal, we will integrate our application tools with all necessary grid services. This will certainly include the use of certificates for security and authorization for using pre-arranged grid resources and code for interacting with Grid Resource Access Managers. It is possible that an information service, such as the Globus MDS, will be used to enable daemons in the DMCS, the Mobile Object Layer, and experimental network measurement tools to discover and interact with one another, but for initial experimental purposes, it may be sufficient to use manual configuration.

5. Summary

We have described the concept of *optimistic grid computation* as a technique to “loosen-up” tightly coupled applications and potentially make them more suitable for execution on grids. While loosely coupled applications are naturally more “grid-friendly”, there are clear scientific and economic motivations to make all applications more “grid-tolerant”. The goal of optimistic computation is to allow applications to synchronize more loosely and better tolerate the dynamic and heterogeneous bandwidths and latencies that will be seen in a grid environment. Based on the observed performance of a world-wide grid testbed, we have estimated target *operating regions* for grid applications derived from the application’s level of concurrency, communication/computation ratio, and the bandwidth-delay product it is currently experiencing.

Mesh generation is our primary test application. We are focusing on Delaunay triangulation methods, and specifically the parallel Bowyer-Watson algorithm. This allows boundary cavities to be expanded optimistically in parallel. When an optimistically generated cavity is found to violate the Delaunay criterion, a setback has occurred and the mesh cavity must be corrected.

To manage the level of optimistic execution and hopefully stay within the application’s operating region, we are integrating *grid performance monitoring and prediction* into the Data Movement and Control Substrate. Using packet train measurement techniques, we are also addressing the issue of *network performance variability*. This information will be used by an Optimistic Control module within the Data Movement and Control Substrate.

Finally, the experience and knowledge of optimistic grid computing gained through using mesh generation will be generalized into a tool that can be applied to other tightly coupled computations in other application domains. This will enable a much larger segment of computational science to effectively realize the potential of grid computing. The resulting research findings and runtime infrastructure should be of interest not only to computational scientist using parallel meshing, but to a broad spectrum of computational scientists and engineers who want to use grid environments for tightly coupled applications.

References

- [1] Eitan Altman, Kostia Avrachenkov, and Chadi Barakat. A stochastic model of tcp/ip with stationary random losses. In *Proceedings of ACM SIGCOMM 2000*, 2000.
- [2] T. Baker. Delaunay triangulation for three dimensional mesh generation. Technical Report 1733, Princeton University, MAE, 1985.
- [3] Sabyasachi Basu, Amarnath Mukherjee, and Steve Klivansky. Time series models for internet traffic. Technical Report GIT-CC-95-27, Georgia Tech, 1995.
- [4] Adrian Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.
- [5] George E. P. Box, Gwilym M. Jenkins, and Gregory Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, 3rd edition, 1994.
- [6] Bruce Carter, Chuin-Shan Chen, L. Paul Chew, Nikos Chrisochoides, Guang R. Gao, Gerd Heber, Antony R. Ingraffea, Chris Myers Roland Krause and, Démian Nave, Keshav Pingali, Paul Stodghill, Stephen Vavasis, and Paul A. Wawrzynek. Parallel fem simulation of crack propagation – challenges, status. In *Lecture Notes in Computer Science*, volume 1800, pages 443–449. Springer-Verlag, 2000.
- [7] Robert L. Carter and Mark E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, 27 and 28, October 1996. also appears as Boston University BU-CS-96-006.
- [8] P. Chew. Guaranteed-quality triangular meshes. Technical Report TR 89-983, Cornell University, Department of Computer Science, 1989.
- [9] Nikos Chrisochoides and Démian Nave. Parallel delaunay mesh generation kernel. *IJNME*, To appear, 2003.
- [10] J. G. Cleary and H. S. Martin. Estimating bandwidth from passive measurement traces. In *Passive and Active Measurement Workshop (PAM-2001)*, 2001.
- [11] Peter Dinda, Thomas Gross, Roger Karrer, Bruce Lowekamp, Nancy Miller, Peter Steenkiste, and Dean Sutherland. The architecture of the remos system. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC 10)*, August 2001.
- [12] Peter A. Dinda. *Resource Signal Prediction and Its Application to Real-time Scheduling Advisors*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 2000. Available as Carnegie Mellon University Computer Science Department Technical Report CMU-CS-00-131.
- [13] Andriy Fedorov. Runtime substrate for grid-aware adaptive computations. cs710 Project Report, Computer Science Department, College of William and Mary, Williamsburg, VA 23197, Spring 2003.
- [14] Anja Feldmann, Anna C. Gilbert, Polly Huang, and Walter Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM 1999*, pages 301–313, 1999.
- [15] I. Foster and K. Kesselman. Globus: A metacomputing infrastructure toolkit. *Journal of Supercomputer Applications*, 11(2):115–128, 1997.
- [16] S. Ben Fredj, T. Bonald, A. Proutiere, G. Regnie, and J. Roberts. Statistical bandwidth sharing: A study of congestion at fbw level. In *Proceedings of ACM SIGCOMM 2001*, pages 111–122, 2001.
- [17] P. L. George and H. Borouchaki. *Delaunay Triangulation and Meshing: Applications to Finite Element*. Hermis, Paris, 1998.

- [18] Matthias Grossglauser and Jean-Chrysostome Bolot. On the relevance of long-range dependence in network traffic. *Transactions on Networking*, 7(5):629–40, October 1999.
- [19] D.A. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.
- [20] Guojun Kin, George Yang, Brian R. Crowley, and Deborah A. Agarwal. Network characterization server (NCS). In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC 10)*. IEEE, August 2001.
- [21] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 2001.
- [22] Kevin Lai and Mary Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM 2000*, pages 283–294, 2000.
- [23] C. Lawson. Transforming triangulations. *Discrete Mathematics*, 3:365–372, 1972.
- [24] C. Lee and J. Stepanek. On future global grid communication performance. *10th IEEE Heterogeneous Computing Workshop*, May 2001.
- [25] Bruce Lowekamp, David O’Hallaron, and Thomas Gross. Direct queries for discovering network resource properties in a distributed environment. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 38–46. IEEE Computer Society, August 1999.
- [26] Bruce Lowekamp, David R. O’Hallaron, and Thomas Gross. Topology discovery for large ethernet networks. In *Proceedings of SIGCOMM 2001*. ACM, August 2001.
- [27] Nancy Miller and Peter Steenkiste. Collecting network status information for network-aware applications. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.
- [28] National Institute of Standards and Technology. NistNet. <http://snad.ncsl.nist.gov/itg/nistnet>, 2001.
- [29] S. Owen. A survey of unstructured mesh generation. Technical report, ANSYS Inc., 2000.
- [30] L. Rizzo. DummyNet, 1999. http://www.iet.unipi.it/~luigi/ip_dummynet.
- [31] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.
- [32] Aimin Sang and San qi Li. A predictability analysis of network traffic. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.
- [33] Srinivasan Seshan, Mark Stemm, and Randy H. Katz. SPAND: Shared passing network performance discovery. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 135–46, December 1997.
- [34] J. Shewchuk. *Delaunay refinement mesh generation*. PhD thesis, CMU, May 97.
- [35] The Globus Team. The Globus Metacomputing Project. <http://www.globus.org>, 2001.
- [36] Ajay Tirumala and Jim Ferguson. Iperf: The TCP/UDP bandwidth measurement tool. <http://dast.nlanr.net/Projects/Iperf/>, May 2001.
- [37] David F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.
- [38] Rich Wolski. Dynamically forecasting network performance using the network weather service. Technical Report CS-96-494, UCSD, 1996.
- [39] Rich Wolski, Neil Spring, and Chris Peterson. Implementing a performance forecasting system for metacomputing: The network weather service. In *Supercomputing '97*, 1997.