A Quality Planning Model for Distributed Multimedia in the Virtual Cockpit

Mark Claypool
John Riedl
{*claypool,riedl*}@*cs.umn.edu*

University of Minnesota
Computer Science Department

ABSTRACT

Tomorrow's multimedia applications will stress all parts of a computer system. To determine the computer resources needed to meet application demands we have developed a new capacity planning model that is based on application quality as perceived by the user. We have applied our model to a Distributed Interactive Simulation flight simulator called the Virtual Cockpit. We investigate the quality of the Virtual Cockpit on existing networks and processors and predict the effects of high-speed networks and high-performance processors on Virtual Cockpit quality. We find processor performance is the current bottleneck in application quality for the Virtual Cockpit, but that higher-speed networks, such as ATM, will be needed to meet network requirements after two to three generations of processor improvement.

Keywords: Communications/Networking/VOD Applications

INTRODUCTION

Today, there are many exciting new distributed multimedia applications. Although current computer systems are often powerful enough for one user, the recent growth of networks has presented the opportunity for multiple users to collaborate using one application. Today, two to tens of users can communicate through a computer audioconference. Tomorrow, tens to hundreds of neuroscientists will explore and contribute to a distributed brain database [1]. Soon, tens, hundreds and perhaps even thousands of soldiers will train for combat in a distributed interactive simulation [4]. With high multimedia system requirements and many users, today's and tomorrow's applications will stress all parts of a computer system.

The study of computer resources needed to meet expected computer demand is called capacity planning. Capacity planning often includes system configuration guidelines. However, system configuration guidelines limit their usefulness by not considering the user perception of the application quality. We have developed a new capacity planning model that is based on application quality as perceived by the user. Like other models, our model incorporates users, applications and hardware into a measure of performance. We enhance the model by including a set of user requirements to get a quantitative measure of quality as perceived by the user. We can then quantitatively predict changes in quality that a user will perceive when the application or hardware is changed.

We start with a distributed, multimedia application. We model the user, application and computer system. We perform some detailed experiments to measure the fundamental components of the application. We validate the detailed experiments through larger experiments. Lastly, we change the model components to reflect alternate configurations and predict the effects on application quality.

Using the above methods, we have applied our model to the specifications and preliminary performance results of a flight simulator called the Virtual Cockpit [19]. The Virtual Cockpit is a low-cost, manned flight simulator of an F-15E built by the Air Force Institute of Technology. A soldier flies the Virtual Cockpit using the hands-on throttle and stick, while the interior and out-the-window views are viewed within a head-mounted display. The Virtual Cockpit research was undertaken to build an inexpensive system for use as a tactics trainer at the squadron level that could participate in a Distributed Interactive Simulation (DIS).

DIS is a virtual environment being designed to allow networked simulators to interact through simulation using compliant architecture, modeling, protocols, standards and databases. The DIS system must be flexible and powerful enough to support increasingly large numbers of simulators. The Advanced Research Projects Agency (ARPA) estimates the need for exercises with 100,000 simulators.

Our objective is to enhance our understanding of the perfor-

[0] This work is sponsored in part by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

mance bottlenecks that arise in the design of a multi-person, distributed multimedia application. Among the factors we investigate are the performance of the Virtual Cockpit on existing networks and processors and the effects of high-speed networks and high-performance processors on Virtual Cockpit quality.

RELATED WORK

In this section, we show how our research complements that of research in related areas.

Quality Application quality research focuses on the user's perception of the applications they are running. Quality researchers have developed some models for measuring application quality [15, 20]. Researchers have studied some factors that affect application performance such as latency and frame rate. Some of the many applications studied include videoconferences and teleoperation [18].

We build upon the reported acceptable tolerance levels for multimedia applications in computing our measures of quality. We expand upon the work of application quality researchers by extending quality models to a new multimedia application.

Geographic Information Systems A Geographic Information System (GIS) is a computer system capable of assembling, storing, manipulating, and displaying geographically referenced information, i.e. data identified according to their locations. GIS are often processor and network intensive. GIS research includes ways to effectively distribute processing time on high performance parallel computer systems while obtaining enough fidelity to support realistic simulations [21], and developing network software architectures to support large scale virtual environments [17].

GIS are the basis for many vehicle simulators. We use performance numbers from the GIS used by the Virtual Cockpit, Software Systems' MultiGen, in our experiments.

Capacity Planning The study of computer resources needed to meet expected computer demand is called *capacity planning*. Many experts agree that the main goal of capacity planning is to maintain a balance between business growth and needs and explicit or implicit service level objectives of computing support. In other words, capacity planning is a method used for projecting computer workload and planning to meet the future demand for computing resources in a *cost-effective* manner.

Capacity planning is a difficult task because no clear economic framework exists to do a cost-benefit analysis of information technology. There have been several general frameworks established in an attempt to manage the growth of information technology [11, 13]. There are also many specific capacity planning solutions designed for specific platforms and intended to plan for specific workloads [24]. Generally, it is still easier to find that a configuration will *not* support a specific level of service than to predict it *will*.

We develop a form of capacity planning that emphasizes the quality of the application as perceived by the user, enabling designers to tradeoff application performance and system cost.

Networks Networks connect the components of distributed applications. Unlike centralized applications of the past, distributed multimedia applications run over one or more networks. Network researchers have studied theoretical limits of past and current networks, as well as the practical limits under normal workloads. Networks that have been studied include Ethernet [22], high-speed gigabyte networks [8], Asynchronous Transfer Mode (ATM) networks [16] and the Internet. Network researchers have also studied how best to use a network's available bandwidth. Network utilization research includes the effects of delay jitter on application performance and transport mechanisms on packet-switched networks [12].

We use quality planning to determine the effects of high-speed networks on the performance of the Virtual Cockpit.

Benchmarks The process of performance comparison for two or more systems by measurements is called *benchmarking*, and the workloads used in the measurements are called *benchmarks*. As early as 1971, Lucas provided a survey categorizing benchmarks [14]. SPEC, the Standard Performance Evaluation Corporation, has sought to create objective series of applications-oriented tests, which can serve as common reference points and be considered during the evaluation process [5]. The SPEC benchmark numbers are the ratio of the time to run the benchmarks on a reference system and the system being tested.

We use SPEC results to make predictions in our quality planning model. Performance results from our research may also be useful to other benchmark researchers.

MODEL

Our model for the quality of a distributed multimedia application incorporates the user, application and hardware. Figure 1 depicts our model.

Users We start with the application users. People interact with touch, sight and hearing. We would like computers to come as close to real-life interaction as possible, even enhancing personal interaction by allowing it across both time and space. The users of the Virtual Cockpit are F-15 pilots training as members of a team. They need the flight simulation to be realistic enough to prepare them for real flight and the response time from other simulators to be fast enough to provide practical "live" training.

Application The application is the software the users will run. The Virtual Cockpit has been used by the ARPA in a series of DIS exercises called Zealous Pursuit. See the Virtual Cockpit paper for more details on the application [19].

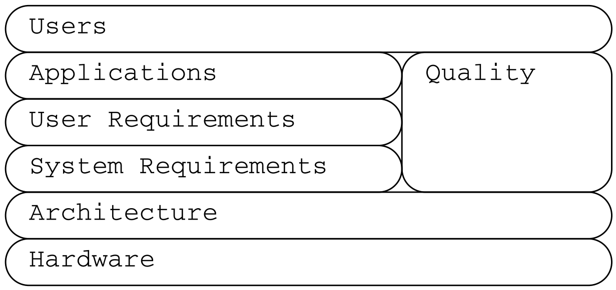

Figure 1: Quality Planning Model. Our model of application performance incorporates users, applications, user requirements, system requirements, architecture and hardware. In addition, we include a measure of application quality as perceived by the user.

User Requirements The application is founded on a set of user requirements that need to be fulfilled for the application to be effective for the user. These include information such as frame rate and frame size, acceptable latency and jitter and tolerance of data loss. The user requirements are the user's interface to our model. The requirements they specify may drive the selection of the underlying system in order to make the application acceptable for the user.

System Requirements The user requirements impose a series of requirements on the system. Some of these include network bandwidth, disk throughput and processor power. The method to determine the system requirements from the user requirements depends upon the application and, to some extent, its implementation. For instance, the workstation can make rendering frames the highest priority, possibly forsaking sending and receiving updates to do so. Or, sending and receiving updates can be the top priority at the expense of a lower frame rate. Packets can be compressed before sending, reducing network bandwidth but possibly increasing processor load from compression and decompression.

Architecture Architecture is the structure of the distributed program which determines the location of data and the distribution of the processing. Architecture can greatly affect the application. For example, a Virtual Cockpit that supports all soldiers on one central mainframe would perform differently than one in which each soldier had a dedicated workstation connected by a network.

Hardware Given the system requirements and architecture, the hardware needed to support the application can be determined. Hardware might range from a low-end workstation with a T1 network up to a high-performance workstation with an ATM network.

Quality The variations in hardware, architecture, system requirements, user requirements and the application all effect the application quality as perceived by the user. The acceptability of the application to the user is determined by how close the application performance matches the user requirements. We are developing a quantitative measure of the difference between application performance and user requirements. We call this the application quality.

QUALITY

The quality of a distributed multimedia application is a measure of the application's acceptability to the user. Although we often think of a multimedia application as a continuous stream of data, the computer system handles multimedia in discrete events. An event may be receiving an update packet or displaying a rendered frame on the screen. The quantity and timing of these events give us measures that affect application quality. We have identified three measures that determine quality for most distributed multimedia applications:

- *Latency.* The time it takes information to move from the server through the client to the user we call *latency*. Latency decreases the effectiveness of applications by making them less like real-life interaction.
- *Jitter.* Distributed applications usually run on non-dedicated systems. The underlying networks are often packet-switched and the workstations are often running multiple processes. These non-dedicated systems cause variance in the latency, which we call *jitter*. Jitter can cause gaps in the playout of a stream such as in an audioconference, or a choppy appearance to a video display for the Virtual Cockpit.
- *Data Loss.* Any data less than the amount determined by the user requirements we call *data loss*. Data loss takes many forms such as reduced bits of color, jumbo pixels, smaller images, dropped frames and lossy compression. Data loss may be done voluntarily by either the client or the server in order to reduce load or to reduce jitter and/or latency.

There may be additional measures that affect application quality that are application specific. For instance, DIS simulators use "dead reckoning" algorithms to compute position. Each simulator maintains a simplified representation of other simulators and extrapolates their positions based on their last reported states. When a simulator determines that other simulators cannot accurately predict its position within a predetermined threshold, it sends a state update packet. The state update contains the correct position and orientation as well as velocity vectors and other derivatives that the other simulators can use to initiate a new prediction. Figure 2 depicts the difference between the actual flight path of a simulator and the dead reckoning flight path computed by the other simulators.

Dead reckoning creates an additional quality measure specific to DIS applications:

- *Missed Updates.* If a simulator is unable to send the update or process an incoming update, the accuracy of the simulation decreases.

Figure 3 shows the effects of missed updates on accuracy. The horizontal line is the position threshold, set at 1 meter. When the inaccuracy surpasses the threshold, an update is sent, bringing inaccuracy back to 0. Inaccuracy increases

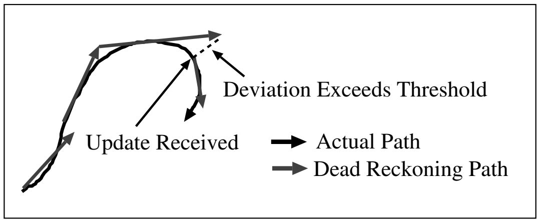

Figure 2: Actual Path versus Dead Reckoning Path. The solid line represents the actual flight path of the simulator. The grey line represents the flight path as computed by the other simulators using a simple dead-reckoning algorithm based only on direction and velocity. The dashed line represents a time when the perceived path deviated from the actual path by more than a pre-set threshold. At this time, a packet updating position, orientation and direction is sent and the dead reckoning resumes from this new posture.

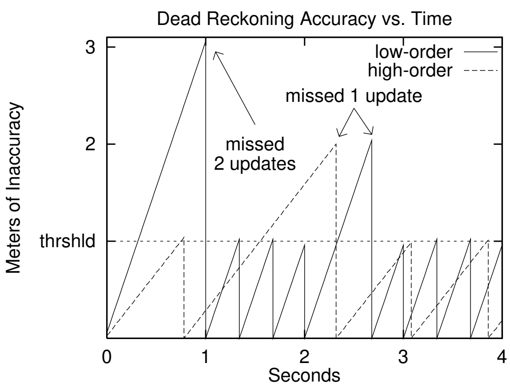

Figure 3: Dead Reckoning Accuracy. The horizontal line is the position threshold, set at 1 meter. The triangular-shapes represent areas of inaccuracy. When they reach the threshold, an update is sent, bringing inaccuracy back to 0. Inaccuracy increases by as much as a meter on the average for each update missed. These missed updates are shown by the taller triangles.

one meter on average for each update missed. These missed updates are shown by the taller triangles. The total inaccuracy for the simulation is represented by the sum of the areas of the triangles. The larger the sum, the more inaccurate the simulation. As we would expect, consecutive missed updates affect accuracy more than non-consecutive missed updates. There are two different simulations depicted in the graph. One uses a high-order dead reckoning algorithm while the other uses a low-order dead reckoning algorithm. The total accuracy is independent of the dead reckoning algorithm used; changing the dead reckoning algorithm only affects the number of updates sent and not the total accuracy.

Ideally, we would like there to be no latency, jitter, data loss or missed updates. Unfortunately, on a variable delay network and non-dedicated computer this can not be achieved. To compute the application quality, we use the above quality components in a process depicted by Figure 4. The user requirements for the application define the acceptable latency,

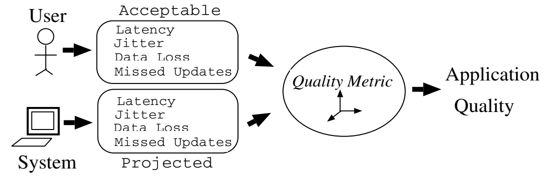

Figure 4: The process for computing application quality. The user defines the acceptable latency, jitter, data loss and missed updates and the system determines the actual values. Based on the acceptable values specified in the user requirements, a quality metric computes the application quality from the actual values.

jitter, data loss and missed updates. The system determines the projected latency, jitter, data loss and missed updates. Acceptable and projected data are fed into a *quality metric* for the application. The quality metric is a function, based on the acceptable components and dependent upon the projected components, that computes the application quality.

In order to quantitatively compare application quality for different system configurations, we need a reasonable quality metric. To form our quality metric, we build upon the work of Kleinrock and Naylor [15]. Using each quality component as one axis, we create a 4-dimensional quality space. We place the best quality value for each axis at the origin and scale each axis so that the user-defined minimum acceptable values have an equal weight. An instantiation of the application lies at one point in this space. We compute the application quality by taking the Euclidean distance from the point to the origin. All points inside the region defined by the user-defined minimums have acceptable quality while points outside do not.

There can be many possible quality metrics for a given application. In fact, there may be many quality metrics that agree with a user's perception of the application. Mean opinion score (MOS) testing can be used to determine if a metric agrees with users' perception. The MOS is a five-point scale where a MOS of 5 indicates perfect quality and a score of 4 or more represents high quality. MOS has been used extensively in determining the acceptability of coded speech. MOS testing is beyond the scope of this paper, so we cannot be certain our quality metric fits user perceptions. However, the metric we chose has several useful characteristics. First, it treats the axes symmetrically which seems appropriate in the absence of user studies to the contrary. Second, the Euclidean distance fits our intuition about changes in quality: the measure increases total quality with any increase in quality along one axis. Third, the metric produces a convex region of acceptable quality, which avoids certain anomalies. The rest of our model is independent of the quality metric chosen. If new metrics are developed, they can be used in place of our quality metric.

One limitation to quality metrics is that after scaling, the upper limits on the axes have different characteristics. The “data loss” and “missed updates” have a finite upper-limit of 100%, while the “delay” and “jitter” axes each have an infinite bound. Comparing application quality for two different configurations at the upper-limit of any of the axes may not match user perception. Fortunately, this limitation only arises when comparing two unacceptable configurations. The metric is most valuable for determining whether a configuration provides “acceptable” or “unacceptable” application quality.

Note that the user-defined acceptability limits along each axis are greatly dependent upon the application and must be re-evaluated for each new applicatio. For example, the acceptable latency for an audio broadcast application of a radio program may be far more than the acceptable latency for the Virtual Cockpit.

For the parameters in our metric, we define the acceptable delay, jitter, data loss and missed updates for the Virtual Cockpit:

1. Latency is the time from when an update is sent until it is processed and displayed by the other simulators. The DIS steering committee had defined latency as 100 to 300 milliseconds as acceptable for DIS applicationsn [4]. We use 300 milliseconds as the maximum acceptable latency for the Virtual Cockpit.

2. Jitter is the variance in the latency. We assume 10% jitter is the maximum acceptable for the Virtual Cockpit.

3. For data loss, we note that research in remote teleoperator performance states that task performance is virtually impossible below a threshold of 3 frames per second [18]. We use 3 frames per second as the minimum acceptable frame rate. The conventional rate of 30 frames per second would provide a more realistic simulation, possibly influencing training effectiveness. We assume that frame rates above 30 frames per second provide no more useful data.

4. For missed updates, we assume the same thresholds of 1 meter position and 3 degrees of accuracy used in SimNet [9]. We assume the simulation must be 95% accurate to be acceptable.

Figure 5 depicts the quality space for the Virtual Cockpit. The user requirements summarized in Table 1 determine a region of acceptable application quality, depicted by the shaded region. All points inside the shaded region have acceptable quality, while those outside the region do not. An instantiation of the application and the underlying computer system would lie at one point in this space. Note that the graph does not show the fourth axis of quality, missed updates, because of the difficulty in depicting the 4th dimension.

Latency	300 milliseconds
Jitter	10%
Frame Rate	3 frames/second
Position Threshold	1 meter
Orientation Threshold	3 degrees
Accuracy	95%

Table 1: User Requirements for the Virtual Cockpit.

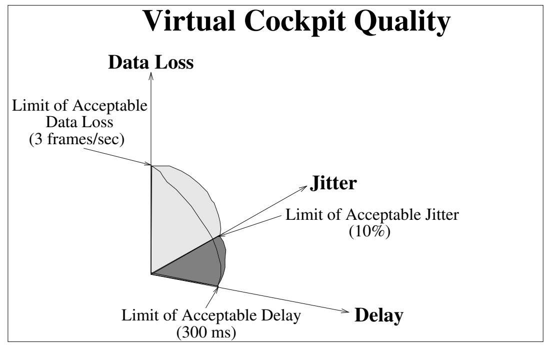

Figure 5: Virtual Cockpit Quality. The axes represent application quality components. The limits along the axes are defined by the user. The shaded region depicts acceptable application quality. An instantiation of the Virtual Cockpit and the underlying computer system would lie at one point in this space. If the point was inside the shaded region, the application would have acceptable quality.

MICRO EXPERIMENTS

Experiments that measure processor performance of components of an application we call *micro experiments*. We do micro experiments to allow us to predict the effects of systems on applications built with those components. The fundamental components for the Virtual Cockpit are: send a dead reckoning update packet, receive update packets, update dead reckoning status, read GIS information from the database, perform dead reckoning, render the frame and display the frame.[1] After carefully measuring the processor, network and disk loads induced by each component, we can predict the load of an application built with those components. Changes in application configuration or changes in hardware are represented by modifying the individual components and observing how that affects the application performance. Examples of previous micro experiments appear in [2, 3].

The fundamental components of the Virtual Cockpit are depicted in Figure 6. Send is the processor load for sending updates to the other soldiers. Receive is the processor load for receiving updates. Update is the processor load for updating dead reckoning status structures. Read is the processor load for reading from a file. Reckon is the processor load for do-

[1] We assume that the user interface contributes an insignificant amount of processor load.

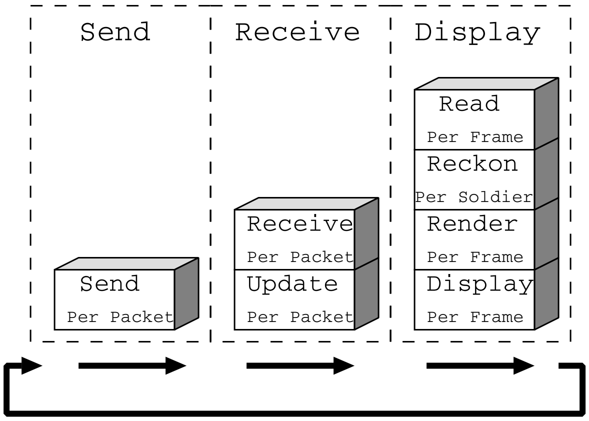

Figure 6: Virtual Cockpit software scheme. The 3-D boxes above all contain a fundamental component of the Virtual Cockpit. The larger, dashed boxes place the fundamental components into conceptual groups.

Component	msec	Low	High	Per Unit
Display	0.194	0.193	0.195	frame
Read	0.000327	0.000325	0.000329	byte
Render	305	304	306	frame
Update	0.00813	0.00808	0.00818	soldier
Compute	0.949	0.940	0.958	soldier
Unicast Receive	0.579	0.230	0.928	update
Multicast Receive	1.12	1.11	1.13	update
Unicast Send	1.58	1.57	1.59	update
Multicast Send	1.57	1.56	1.58	update

Table 2: Processor load breakdown for the fundamental components of the Virtual Cockpit. Send is the processor load for sending updates to the other soldiers. Receive is the processor load for receiving updates. Update is the processor load for updating dead reckoning status structures. Read is the processor load for reading from a file. Reckon is the processor load for doing a dead reckoning computation to determine the position of another soldier. Render is the processors load for generating a frame to be displayed. Display is the processor load for displaying the frame on the screen. "Low" and "High" are the lower and upper bounds respectively from the 95% confidence interval.

ing a dead reckoning computation to determine the position of another soldier. Render is the processors load for generating a frame to be displayed. Display is the processor load for displaying the frame on the screen. Send is done once for each update packet sent. Receive and Update are done once for each update packet received. Read, Render and Display are done once per frame. Reckon is done once per soldier per frame.

We use a high-order dead reckoning algorithm as an upper bound on the processor load required.

We compared the load for sending and receiving unicast packets to that of multicast so we could better analyze the benefits of multicast in Section "Predictions" below. The update packet size is 144 bytes, as defined by the DIS standard [17].

We used a process that increments a `long integer` variable in a tight loop to measure the processor load for the components in the Virtual Cockpit model. We did not use the Unix `time` command because the reporting of per-process processor consumption by most operating systems is unreliable. Often, the system gives an incorrect account of interrupt level processing and fails to capture processor degradation from Direct Memory Access (DMA).

To obtain a baseline for our counter, we ran the counter process on a machine with a minimum of system processes running. This gave us the processor potential for the machine. We then ran the counter process with each of the fundamental components. The difference between the bare count and the count with the component process is the component-induced load.

We ran our experiments on a dedicated network of SGI Personal Iris workstations. Each workstation had a 20 MHz IP6 R3000 processor, 16 Mbytes RAM and 96 Kbytes cache.

Table 2 gives the processor times for the Virtual Cockpit components. All points are shown with a 95% confidence interval. The times for multicast send and unicast send are indistinguishable at the 95% confidence level. The time for multicast receive is significantly more than the time for unicast receive.

Although we present only the measurements of the processor load, in the past, workstation performance has scaled with processor performance [10]. We assume that processor performance will continue to reflect the workstation performance.

MACRO EXPERIMENTS

Experiments that measure performance of applications built with micro experiment components we call *macro experiments*. We do macro experiments to validate micro experiment-based predictions of application performance. Examples of macro experiments appear in [2, 3].

We ran macro experiments for the Virtual Cockpit model for 2-15 simulated soldiers. Each soldier ran a Virtual Cockpit on an SGI Personal Iris, one workstation per soldier. The workstations were connected by an Ethernet. Each Virtual Cockpit sent 3 updates packets per second, the average for most vehicles in a typical DIS exercise [17].

Our micro experiments measured the processor load for the fundamental components of the Virtual Cockpit. Using these measurements, we can predict the number of occurrences of each of the fundamental components in the macro experiments. For example, in a 10 second, two-soldier simulation, we would expect each soldier to send and receive 30 update packets. Table 3 gives the predicted versus actual count for

Component	Predicted	Actual
Receive	27	25
Send	3	3
Update	27	25
Compute	27	28
Render	3	3
Display	3	3
Read	6265	6354

Table 3: Component predictions for a Virtual Cockpit simulation with 9 Soldiers. "Predicted" are the predicted performance numbers based on the performance of the individual components. Actual are the actual performance numbers reported in the experiment. All predicted values are for one second.

all the Virtual Cockpit components for an experiment with 9 soldiers. "Predicted" are the predicted numbers of times each component occurred per second based on the performance of the individual components. "Actual" are the actual numbers of times each component was recorded per second in the experiment. Other macro experiments with 2-15 soldiers had similar results, but we do not show them here to avoid redundancy.

All of the predicted performance results are within 10% of the actual performance results. We regard future comparisons of predicted performance results that combine the Virtual Cockpit components into alternate configurations to be significantly different if they vary by more than 10%.

Our micro experiments give us the performance results for the fundamental components of the Virtual Cockpit. Our macro experiments validate our ability to combine the values from our micro experiments into accurate Virtual Cockpit predictions.

PREDICTIONS

By modifying the fundamental Virtual Cockpit components, we can predict performance on alternate system configurations. This allows us to evaluate the potential performance benefits from expensive high-performance processors and high-speed networks before installing them. Moreover, we can investigate possible performance benefits from networks and processors that have not yet been built.

Our approach for evaluation of each alternative system is the same: We modify the parameters of our performance model to fit the new system, then evaluate the resulting model to obtain performance predictions. These analyses are intended to provide a sense of the relative merits of the various alternatives, rather than present absolute measures of their performance.

Network

We first investigate the network bandwidth requirements for a DIS simulation. Network bandwidth requirements depend

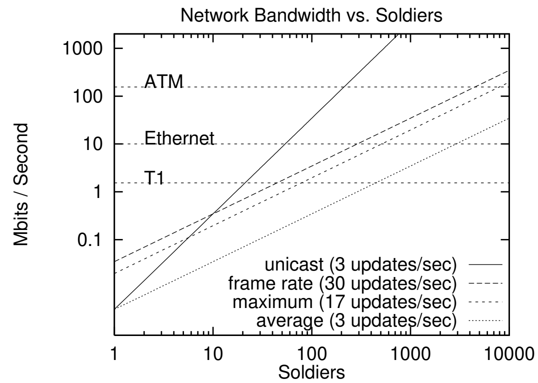

Figure 7: Network Bandwidth versus Soldiers. This graph shows the predicted network bandwidth as the number of soldiers increases. The upward sloping lines are the predicted bandwidth for the frame rate, maximum rate, and average update rate. The steeply sloped line is the network bandwidth required for unicast with an average update rate. The horizontal lines are the maximum bandwidths for a T1, Ethernet and ATM network. Both the horizontal axes and vertical axes are in log scale.

upon the frequency with which each Virtual Cockpit broadcasts its position. In the absence of dead reckoning, each simulator would have to send an update every frame, for a maximum of 30 updates per second. However, with dead reckoning, the average rate is 3 updates per second [17], with a maximum of 17 updates per second [9].

Figure 7 depicts the predicted network bandwidth versus the number of soldiers in a Virtual Cockpit exercise. The upward sloping lines are the predicted network bandwidth for updates sent at the frame rate without dead reckoning, the maximum update rate with dead reckoning, and average update rate with dead reckoning. The line with the steepest slope represents the network bandwidth required for a unicast simulation at an average update rate. The horizontal lines are the maximum bandwidths for a 1.5 Mbps T1, 10 Mbps Ethernet and 155 Mbps ATM network.

Without dead reckoning, a T1 network becomes saturated at about 50 soldiers and an Ethernet at about 200 soldiers. With dead reckoning and the maximum update rate, a T1 can support nearly 100 soldiers and an Ethernet can support nearly 400 soldiers. With dead reckoning and the average update rate, a T1 can support about 400 soldiers and an Ethernet can support over 2000 soldiers. We use the maximum update rate for situations in which we are concerned about peak network bandwidth. We use the average update rate in situations in which we are concerned about network throughput.

Multicast is crucial for soldier scalability. Even at the average update rate, a Virtual Cockpit exercise using unicast rapidly exceeds the bandwidth available on T1 and Ethernet networks.

Processor

Existing networks are capable of supporting many soldiers, but what about existing processors? We investigate the performance results Virtual Cockpit exercises with more powerful processors.

We compare the performance of the SGI Personal Iris to other processors by comparing performance results from the Systems Performance Evaluation Cooperative (SPEC) benchmark integer suite. In past work, we have found SPEC results correlate inversely well with execution times for processor-intensive tasks. Since the largest Virtual Cockpit component, render, is largely processor-intensive, we assume that the Virtual Cockpit processor performance will correlate well with SPEC results. The SPEC int92 value for a SGI Personal Iris is 22.4 and the SPEC int92 value for the 150 MHz SGI Indigo 2 is 92.2 [6]. Roughly, the Indigo 2 is 6.5 times faster than the Personal Iris, so we assume it does all the Virtual Cockpit components measured in Section "Micro Experiments" 6.5 times faster.

Swartz et al. did experiments to measure the effects of frame rate on on the ability to perform tasks through Unmanned Aerial Vehicles (UAVs) [23]. UAVs are used to conduct a variety of reconnaissance missions with human operators interpreting the transmitted imagery at ground stations. The experiments found that human performance with 4 frames per second is significantly better than 2 frames per second, but not consistently worse than 8 frames per second. Therefore, we assume a rate of 4 frames per second in our processor load predictions.

We break the processor load into three pieces: send, receive and display (see Figure 6 for the components in each piece). Figure 8 depicts the predicted processor load in seconds on a SGI Personal Iris versus the number of soldiers. The graph reads from the bottom. The load of each piece is the sum of the pieces below it. Thus, the total processor load is indicated by the "display" line at the top. The horizontal lines are the maximum processor throughputs for SGI Personal Iris' and SGI Indigo 2s, as indicated.

The processor is the bottleneck. The SGI Personal Iris is unable to support even the minimal frame rate for any soldiers. Processors significantly more powerful than SGI Personal Iris' are needed to realize the army's goals of hundreds and thousands of distributed soldiers interacting in a simulation. Indigo 2s, powerful workstations, can support adequate frame rates while communicating with 1000s of other soldiers.

Quality

If we have a system with the more powerful processors required for acceptable frame rate, we can investigate the predicted user perception of the Virtual Cockpit – what is the *application quality*? In order to predict application quality, we need to predict latency, jitter, data loss and missed updates for a variable number of soldiers.

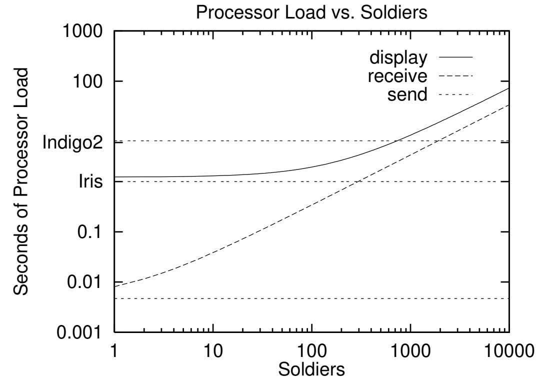

Figure 8: Processor Load versus Soldiers. This graph depicts the predicted processor load in seconds on a SGI Personal Iris versus the number of soldiers. The graph reads from the bottom. The load of each piece the sum of the pieces below it. Thus, the total processor load is indicated by the "display" line at the top. The horizontal lines are the maximum processor throughputs for an SGI Personal Iris and an SGI Indigo 2. Both axes are in log base 10 scale.

The macro experiments we ran allow predictions of latency and jitter. We compute latency by adding: the time between frames; the time spent sending the update; and the network propagation time. Figure 9 shows the latency data and Figure 10 shows the jitter data for a variable number of soldiers. We obtain predictions for latency and jitter by fitting the recorded data points with a least-squares line fit. The curves in the graphs are 95% confidence intervals around the lines.

Data loss and missed updates for the Virtual Cockpit are determined by the processor load. If the processor power is limited there are two choices: reduce processor load by reducing the frame rate (data loss), or reduce processor load by dropping updates (missed updates).[2] The processor load for updates is low compared to the processor load for displaying frames. For example, in a simulation exercise with 10 soldiers, a Virtual Cockpit would need to drop about 300 updates in order to equal the processor load for dropping 1 frame. It is likely that dropping 300 updates would adversely affect quality greater than dropping 1 frame. Future work is needed to completely understand the quality tradeoffs between dropping frames and retaining updates, but in this paper, we assume the processor processes all updates before rendering frames.

To compute application quality, we use the methods described in Section "Quality" above. We examine the application quality for Virtual Cockpit exercises with SGI Indigo 2s because SGI Personal Iris' were incapable of achieving the minimum acceptable frame rate. Figure 11 depicts our quality predic-

[2]There may be a way of using voluntary data loss in the GIS database engine to reduce processor load, but we leave that as future work.

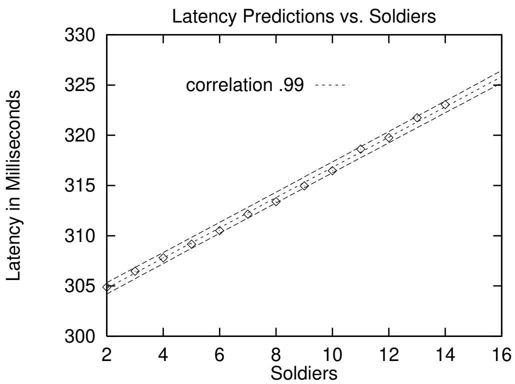

Figure 9: Latency versus Soldiers. The above graph shows latency graphed versus the number of soldiers in the Virtual Cockpit simulation. The middle line is a least-squares line fit. The correlation is 0.99. The curves are 95% confidence intervals around that line.

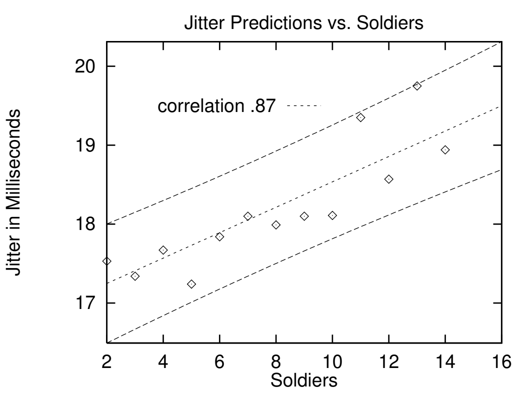

Figure 10: Jitter versus Soldiers. The above graph shows jitter graphed versus the number of soldiers in the Virtual Cockpit simulation. The middle line is a least-squares line fit. The correlation is 0.87. The curves are 95% confidence intervals around that line.

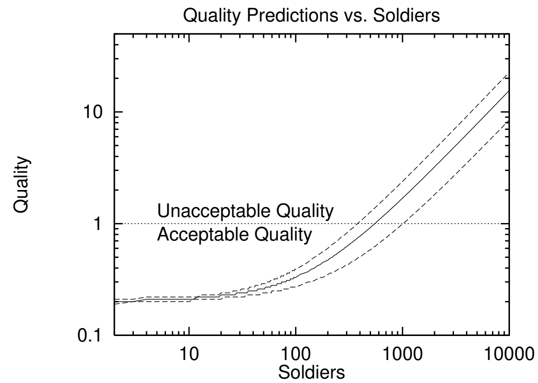

Figure 11: Virtual Cockpit Quality versus Soldiers. The middle line represents predicted application quality as the number soldiers increases. The upper and lower lines represent best and worst case quality scenarios respectively. The horizontal line marks the acceptable/unacceptable quality limit.

tions.

SGI Indigo 2s provides acceptable quality for Virtual Cockpit exercises with about 500 soldiers. Notice that in Figure 8, we had predicted that the Indigo 2s could support 1000 soldiers when we analyzed the frame rate alone. Application quality is determined by latency, jitter and missed updates as well as frame rate. Latency, jitter and missed updates all increase as the number of soldiers increase.

High-performance Processors

The quality of the Virtual Cockpit was improved by using high-performance processors. Is there further benefit from higher-performance processors?

We assume the network has sufficient bandwidth to handle all necessary updates in order to minimize the effects of the network. We compare the quality of the Virtual Cockpit with SGI Personal Irises and SGI Indigo 2s to the quality of the Virtual Cockpit with processors 15 times more powerful than the Indigo 2.[3]

Figure 12 shows the quality predictions for the Virtual Cockpit with different processors. The top curve is an SGI Personal Iris. The second curve is an SGI Indigo 2. The bottom curve is a processor 15 times more powerful than the Indigo 2. The horizontal line represents the acceptable quality limit. The “knee” in the curve for the 15x processor is where the processor decreases the frame rate in order to handle the updates from the other soldiers.

High-performance processor are crucial for acceptable Virtual Cockpit quality. SGI Personal Iris’ are unable to deliver acceptable application quality. More powerful SGI Indigo 2s can deliver acceptable application quality for up to 500 soldiers. 15x’s provides better application quality than Indigo

[3] Processor performance approximately doubles every year [10]. The 15x processor will come along in about 8 years.

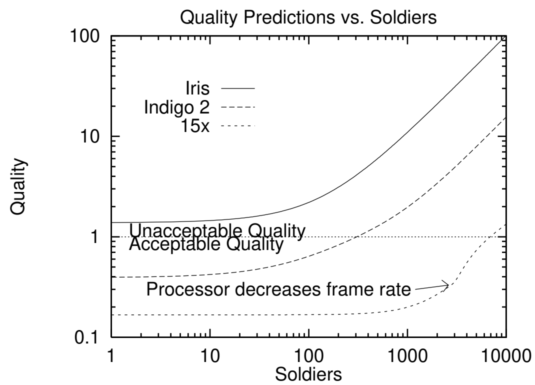

Figure 12: Virtual Cockpit Quality versus Soldiers. The three curves represent the quality predictions for three different processors. The top curve is an SGI Personal Iris. The second curve is an SGI Indigo 2. The bottom curve is a processor 15 times more powerful than the Indigo 2. The horizontal line represents the acceptable quality limit.

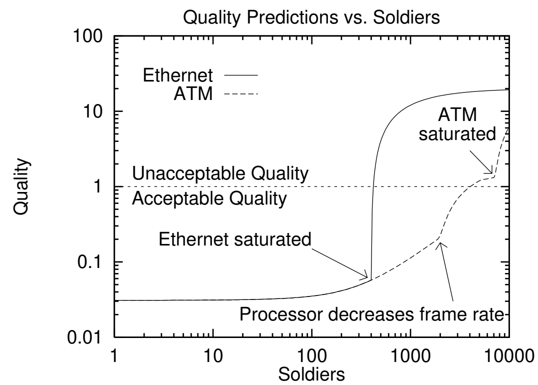

Figure 13: Virtual Cockpit Quality versus Soldiers. The two curves represent the quality predictions for an Ethernet and an ATM network. The horizontal line represents the acceptable quality limit.

2s and can deliver acceptable application quality for up to 7000 soldiers.

High-speed Networks

With the Virtual Cockpit running on processor 15 times more powerful than the SGI Indigo 2, a T1 network will become saturated while supporting just 100's of soldiers. How much quality benefit will then be gained from a high-speed network? We compare the quality of the Virtual Cockpit with an Ethernet to that of the Virtual Cockpit with an ATM network. The ATM network transmits the update packets faster (155 Mbits/second versus 10 Mbits/second for an Ethernet). Past work has found jitter and missed updates in the ATM network are the same as jitter and missed updates in the Ethernet [7]. We assume jitter and missed updates remain the same in high-speed networks. In order to make the network bandwidth more significant, we assume the maximum 17 updates per second.

Figure 13 shows the quality predictions for the Virtual Cockpit with different networks. The top curve is the quality predictions for an Ethernet. The lower curve is the quality predictions for an ATM. The steep increase in the Ethernet curve occurs when the Ethernet becomes saturated. At this point, the Virtual Cockpit begins to increasingly miss updates. The first bend in the ATM curve occurs when the processor must decrease the frame rate in order to process all updates. The second bend in the ATM curve occurs when the ATM becomes saturated.

High-speed networks are unimportant for the Virtual Cockpit quality until existing networks reach saturation. The quality prediction curves for the Ethernet and the ATM are indistinguishable until the Ethernet becomes saturated. At this point, the ATM network greatly increases scalability. The network as the present bottleneck in application quality has been removed by the DIS use of dead reckoning and multicast.

CONCLUSIONS

Multi-person distributed multimedia applications stress all parts of a computer system. We have developed a quality planning model for distributed multimedia applications that allows us to investigate potential bottlenecks in application quality. We have applied our model to the Virtual Cockpit, a flight simulator used in Distributed Interactive Simulation (DIS). DIS is a virtual environment within which simulators may interact through simulation at multiple networked sites. In order for DIS to be effective for military training, simulation exercises must support up to 100,000 soldiers.

Our objective in identifying application bottlenecks is to understand the system limits that will prevent applications from meeting users' needs. After identifying each bottleneck, we explore ways to reduce the effect of the bottleneck through improving system resources. We then explore the new bottlenecks that arise in the enhanced system. Our analysis at each stage is likely to overstate system performance, because we assume maximum possible performance of each system component. However, the bottlenecks we identify are likely to be bottlenecks in practice, and the design principles suggested by the analysis should ameliorate these bottlenecks in practice.

In applying our model to the Virtual Cockpit, we discovered:

- High-performance processors are crucial for acceptable Virtual Cockpit quality. Low-end processors are unable to deliver acceptable application quality. More powerful processors can deliver acceptable application quality for 1000's of soldiers.
- High-speed networks are unimportant for Virtual Cockpit quality until existing networks reach saturation. T1 and Ethernet networks will saturate after 2-3 generations of processor improvement. The network as the present bottleneck in application quality has been removed by the DIS use of dead

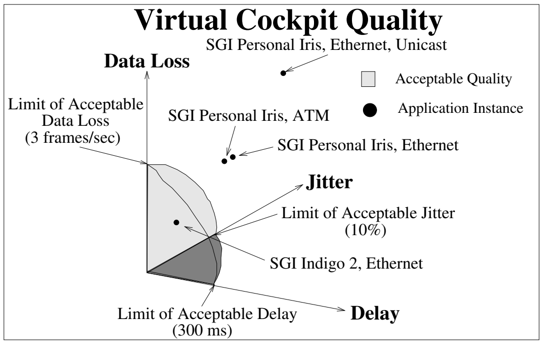

Figure 14: Virtual Cockpit Quality. The user defines the acceptable latency, jitter and data loss. These values determine a region of acceptable application quality, depicted by the shaded region. All points inside the shaded region have acceptable quality, while those outside the region do not. Four Virtual Cockpit instantiations are shown as points in this space. Note that the graph does not show the fourth axis of quality, missed updates, because of the difficulty in depicting four dimensions.

reckoning and multicast.

- Dead reckoning greatly benefits the Virtual Cockpit quality, primarily due to its reduction of network load. The small update message size plus the infrequent number of updates reduces the network load over a scheme where all simulators update their position with each frame. With dead reckoning, a T1 can support 10's of simulators, an Ethernet can support 100's of simulators and an ATM can support 1000's of simulators.
- Multicast is a crucial to maintain reduced network bandwidth. Even at the average update rate, a Virtual Cockpit exercise using unicast rapidly exceeds the bandwidth available on T1 and Ethernet networks.

We depict the above conclusions in Figure 14. The figure shows our measure of quality for the Virtual Cockpit. The user defined acceptable latency, jitter, data loss and missed updates determine a region of acceptable application quality, depicted by the shaded region. Each point is an instantiation of the application and the underlying computer system. All points inside the shaded region have acceptable quality, while those outside the region do not. There are four 100-soldier Virtual Cockpit instantiations shown: SGI Personal Iris' with Ethernet, SGI Personal Iris' with ATM, SGI Indigo 2s with Ethernet and SGI Personal Iris' with Ethernet and unicast. Only the Indigo 2s with Ethernet have acceptable application quality. The ATM does not significantly improve the quality of the Virtual Cockpit with the Personal Iris'. Using unicast instead of multicast greatly decreases the application quality for the Personal Iris' with Ethernet. Note that the graph does not show the fourth axis of quality, missed updates, because of the difficulty in depicting four dimensions. We chose to eliminate the missed updates axis because the predicted number of missed updates is constant for all system configurations.

FUTURE WORK

An interesting processing decision arises in the processing of dead reckoning updates. Dead reckoning allows a tradeoff among three factors: network bandwidth, processor load and simulation accuracy. After a dead reckoning computation, a simulator will have the perceived position of other simulators. This perceived position will be inaccurate up to a pre-set threshold. Lower thresholds provide a more accurate representation of other simulators, but increase the number of updates required, causing more network load. Higher order dead-reckoning decreases the number of updates sent, but increases processor load. Our model could be used to determine the affect on application quality for the various dead reckoning parameters, possibly recommending a specific accuracy and algorithm to use.

Applications that have changing user requirements present another challenge. For example, users doing remote problem-solving via a video link, may want to maximize frame rate at the expense of frame resolution while they are identifying the location of the problem. Once the problem is located, they may want to maximize frame resolution at the expense of frame rate (perhaps even wanting a still image) to best identify the problem. As presented, our model does not allow specification of dynamic user requirements. One possible solution would be to apply a separate quality planning model to each set of user requirements specified. The model that had the poorest quality for a given system configuration could then be examined more closely to determine the application quality bottleneck within.

For some applications, there is potential for interaction effects among the quality events. For example, 3-dimensional graphics applications have multiple factors affecting users' perception of objects and different combinations of requirements may yield satisfactory results. Such applications may even have a non-convex region of acceptable quality. Future research into new quality metrics appropriate for these applications may be required.

We assume that processor performance will continue to reflect workstation performance. If other workstation components, in particular the data bus, do not continue to scale with processor performance, future work would include extending our model to discover which workstation components are bottlenecks to application quality.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. Their suggestions have helped improve the quality of this work.

REFERENCES

1. J. Carlis, J. Riedl, A. Georgopoulos, G. Wilcox, R. Elde, J. H. Pardo, K. Ugurbil, E. Retzel, J. Maguire, B. Miller, M. Claypool, T. Brelje, and C. Honda. A zoomable DBMS for brain structure, function and be-

havior. In *International Conference on Applications of Databases*, June 1994.

2. M. Claypool, J. Riedl, J. Carlis, G. Wilcox, R. Elde, E. Retzel, A. Georgopoulos, J. Pardo, K. Ugurbil, B. Miller, and C. Honda. Network requirements for 3D flying in a zoomable brain database. *IEEE JSAC Special Issue on Gigabit Networking*, 13(5), June 1995.

3. Mark Claypool and John Riedl. Silence is golden? The effects of silence deletion on the CPU load of an audio conference. In *Proceedings of IEEE Multimedia*, Boston, May 1994.

4. The DIS Steering Committee. The DIS vision - a map to the future of distributed interactive simulation. Technical report, Institute for Simulation and Training, May 1994.

5. Standard Performance Evaluation Corporation. SPEC primer.
July 1994. The SPEC primer is frequently posted to the newsgroup comp.benchmarks. SPEC questions can also be sent to spec-ncga@cup.portal.com.

6. gsnow@clark.edu Gary Snow. Specint and specfp 1992 numbers. *comp.benchmarks*, November 1993.

7. Joe Habermann and John Riedl. Using real-time priorities to eliminate jitter in a multimedia stream. Technical Report Technical Report, University of Minnesota Department of Computer Science, January 1996. The author can be contacted about this paper at haberman@lcse.umn.edu.

8. Charles Hansen and Stephen Tenbrink. Impact of gigabit network research on scientific visualization. *IEEE Computer*, May 1993.

9. Edward P. Harvey and Richard L. Shaffer. Capability of the distributed interactive simulation networking standard to support high fidelity aircraft simulation. In *Proceedings of the 13th Interservice/Industry Training Systems Conference*, 1993.

10. John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

11. Magid Igbaria and Snehamay Banerjee. Computer capacity planning management: Definitions and methodology. *Journal of Information Technology*, (9):213 – 221, 1994.

12. K. Jeffay, D.L. Stone, and F.D. Smith. Transport and display mechanisms for multimedia conferencing across packet-switched networks. *Computer networks and ISDN Sytems*, 26(10):1281 – 1304, July 1993.

13. Sanjay K. Jha and Bruce R. Howarth. Capacity planning of LAN using network management. In *Proceedings of Conference on Locacl Computer Networks*, pages 425 – 430, Washington D.C., 1994.

14. Henry C. Lucas Jr. Performance evaluation and monitoring. *Computing Surveys*, 3(3):78 – 91, September 1971.

15. Leonard Kleinrock and William E. Naylor. Stream traffic communication in packet switched networks: Destination buffering considerations. *IEEE Transactions on Communications*, 30(12):2527 – 2534, December 1982.

16. Mengjou Lin, Jenwei Hsieh, David H.C. Du, Joseph P. Thomas, and James A. MacDonald. Distributed network computing over local atm networks. *ATM LANs: Implementation and Experience with An Emerging Technology*, 1995.

17. Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Paul T. Barham, and Steven Zeswitz. NPSNET: A network software architecture for large scale virtual environments. *Presence*, 3(4):265 – 287, October 1994.

18. Michael J. Massimino and Thomas B. Sheridan. Teleoperator performance with varying force and visual feedback. In *Human Factors*, pages 145 – 157, March 1994.

19. W. Dean McCarty, Steven Sheasby, Philip Amburn, Martin R. Stytz, and Chip Switzer. A virtual cockpit for a distributed interactive simulation. *IEEE Computer Graphics and Applications*, January 1994.

20. Radhika R. Roy. Networking contraints in multimedia conferencing and the role of ATM networks. *AT&T Technical Journal*, July/August 1994.

21. Shashi Shekhar, Sivakumar Ravada, Greg Turner, Douglas Chubb, and Vipin Kumar. Load balancing in high performance GIS: Partitioning polygonal maps. In *Proceedings of the Internationall Symposium on Large Spatial Databases*, 1995.

22. John F. Shoch and Jon A. Hupp. Measured performance of an Ethernet local network. *Communications of the ACM*, 23(12):711–720, December 1980.

23. Merryanna Swartz and Daniel Wallace. Effects of frame rate and resolution reduction on human performance. In *Proceedings of IS&T's 46th Annual Conference*, Munich, Germany, 1993.

24. Brian L. Wong. *Capacity Planning for Solaris Servers*. Prentice Hall, 1996.