Analytical Simulation for Performance Analysis of Distributed Virtual Reality Systems

Shaun Bangay
Peter Clayton
Computer Science Department
David Sewry
Information Systems Department
Rhodes University, P.O. Box 94, Grahamstown, 6140, South Africa

cssb@cs.ru.ac.za

Abstract

Existing performance analysis techniques have limitations when used on distributed virtual reality systems, including a lack of support for the measurement of interaction latency. Results tend to be numerical in nature, limiting their usefulness when comparing models. This paper describes an approach to using simulation to generate analytic forms of the performance measures required for virtual reality systems.

The execution of each process is simulated, keeping track of the execution time of each process as a symbolic expression. Cycle times and latencies can be found by measuring the appropriate time intervals during simulation.

The application of analytical simulation is illustrated by applying it to common issues in distributed virtual reality systems: network performance, database distribution and measurement of interactive performance. Comparing the analysis results with an implementation of the model demonstrates the accuracy of the predictions.

1. Introduction

The requirements of performance analysis for distributed virtual reality systems differ from those for other networked and parallel systems in that the former place a very much greater emphasis on interaction. It is no longer sufficient merely to examine speedup, and throughput. A value of great importance is interaction latency, the time delay between a user providing input to the system and experiencing the result of that action. This, together with the rate at which the system cycles, the frame rate or its reciprocal, the cycle time, is essential for describing the performance of virtual reality systems [7] [15] [5].

This paper describes a performance analysis technique capable of producing the metrics relevant to virtual reality systems. The approach is based on simulation of a model of the system, but produces results in analytic form, as expressions relating the variables in the model.

The following sections describe the analytical simulation approach and illustrate its use by applying it to common issues in distributed virtual reality systems: network performance, database distribution and measurement of interactive performance.

2. Related Work

A number of studies have been done describing different distributed distributed virtual reality systems and contrasting the different parallel decomposition strategies [16] [21] [10] [20]. An element lacking in these studies was a discussion of the relative performance characteristics of the various approaches.

This may be due to the lack of a suitable method for performing this comparison. Common approaches to performance prediction in parallel systems are Petri Nets and Data Flow Graphs. Stochastic Petri Nets are not considered suitable for analysis of large, real-time systems [1] and the Markovian analysis for large nets, with both stochastic and deterministic times, becomes impractical [8]. Data Flow graphs offer the prospect of determining cycle times and latencies but require simulation to calculate the latter value [18] [13]. Both approaches resort to simulation to solve large problems, producing numerical values from their analyses which are not particularly conducive to comparison across architectures.

Other approaches to performance prediction are also unable to provide an analytical solution or are specific to a particular form of parallel decomposition [2]. Studies in which analytical solutions are derived are limited to predefined variables [12].

The desired analysis technique should possess the following properties to be suitable for the comparison of distributed virtual reality systems:

- Provide measures of latency and cycle time
- Be capable of modelling the decomposition strategies used for virtual reality systems
- Produce results suitable for comparison purposes

The limitation of simulation is that it is not general enough to cover results that are not explicitly simulated. Analytic performance modelling on the other hand is either ad hoc, extremely complex and computationally intensive, or does not provide the desired performance measures. The next section describes an approach which satisfies these requirements

3. Analytical Simulation

3.1. The Analytical Simulation Algorithm

This approach will be introduced by way of a simple example. Consider a simple client-server system with two clients each which send a request to the server, get a reply and processes it for a period represented by variable X. The server picks up a request, takes period Y to service it and returns a reply. Communication is synchronous; both processes must be ready to take part in an exchange before a message can be passed. This system is simple enough to be analyzed by hand, and as may be expected, the result depends on the relative sizes of X and Y.

Simulating the execution of the program will produce the process activity versus time diagram in Figure 1, related to the Single Graph Play diagrams used for Data Flow analysis in [18]. The intervals at which processing occurs in each process are shown as solid blocks. During the remainder of the time the process is blocked waiting for communication with another process.

In this case the clients are identical, so the one which is first serviced by the server will be labelled C1 and the second C2. The second synchronization between C1 and the server occurs once C1 has finished processing for the period X, and after the server has completed processing C2's request, its second delay of period Y. This can be calculated as:

From C1's point of view: Y + X

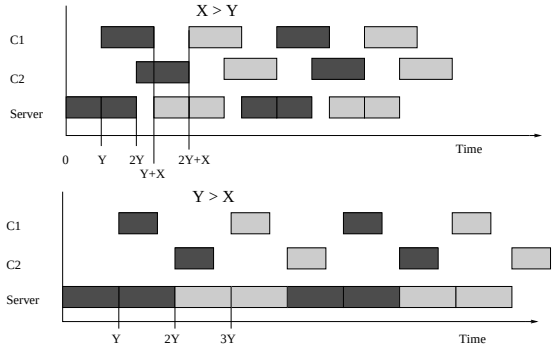

Figure 1. Process Activity versus Time diagram for client-server system

From the servers point of view: Y + Y

Clearly this time depends on the relative sizes of X and Y. If X is larger, the client will delay the synchronization, otherwise the server is the delaying factor.

A trace of the times at which C1 finishes its processing is given below:

For X≥Y : X+Y 2X+2Y 3X+3Y ... n(X+Y)

For Y≥X : X+Y X+3Y X+5Y X+7Y ... 2nY+X-Y

The latency and cycle times for C1 can now be easily calculated. In this case, it is assumed that input arrives just before the client issues a request to the server, and output occurs just after client processing (X) is complete. Since latency is the period from input to corresponding output, latency and cycle time are the same in this example, and given by the time between two successive cycles:

For X≥Y : Latency = Cycle time = X + Y

For Y≥X : Latency = Cycle time = 2Y

This technique will be illustrated for a slightly more complex model in which latency and cycle time differ, before the general algorithm is presented.

Consider a common parallel processing topology, the pipeline. The one modelled will have three nodes and will run the processes as shown in Figure 2, where two markers are included to show where input (Latency 1) and output (Latency 2/Output) occur. It is assumed that input data is always immediately available and the output data can be delivered without delay. The latency in this system is the time taken for any specific datum to be transformed from input to output. The cycle time is the time taken per result produced. As with the previous example, a process activity versus time diagram shows that several decisions need to be made regarding the relative sizes of the variables, in this

```
process 1
  do forever
    Marker:  Latency 1
    consume input
    process for A seconds
    send data to 2
process 2
  do forever
    receive data from 1
    process for B seconds
    send data to 3
process 3
  do forever
    receive data from 2
    process for C seconds
    produce result
    Marker:  Latency 2 / Output
```

Figure 2. Three Processor Pipeline

case A, B and C, in order to draw the process activity versus time diagram. The possible situations are slightly more numerous in this example, in fact there are infinitely many of them. The various cases are listed in Figure 3

The complex inequality arises from a transient which works its way out of the system after a certain number of iterations, dependent on the relative values of A, B and C.

By marking certain points in the program and calculating the time at which they occur, it is possible to find values for latency and cycle time. The values given in Figure 3 show the time at which program execution reached each marker for cycle *n* and the resulting cycle times and latencies. An algorithm for deriving the various cases associated with the analytical simulation technique as well as the sequence of time values at various points is presented in Figure 4. This algorithm performs essentially the same process as illustrated in the previous examples. A process activity versus time diagram is constructed by simulating the program. The synchronization points are analyzed to determine the time of synchronization. This analysis may result in various constraints on the variables being set up, to guide further simulation. By working symbolically as illustrated in the examples in this section, analytic solutions can be obtained with constraint regions which specify the variable values for which each solution applies.

The approach in its current form has several problems. The last line in Figure 4 assumes the system is going to settle into a stable state after some point. This line requires a user specified cutoff, at a point where sufficient analysis has been performed. Some systems may never reach a point after which all possible relationships have been enumerated. Such a system was illustrated in the pipeline example earlier. The problem of choosing the cutoff point is discussed

Latency 1	Latency 2/Output
$A \geq B, A \geq C$:	
(n - 1) A	nA + B + C
$A \geq B, C \geq A$ and $B + kC \leq (k+1)A$: $(k+1)A \leq (k+2)A \leq B + (k+1)C,\ k = 0..\infty$:	
(n - 1) A, n≤k+2	A + B + nC
A + B + (n-3)C, otherwise	
$B \geq A, B \geq C$:	
0, n = 1	A + nB + C
A + (n-2)B, otherwise	
$B \geq A, C \geq B$:	
0, n = 1	A + B + nC
A, n = 2	
A + B + (n-3)C, otherwise	
Latency	Cycle Time
$A \geq B, A \geq C$:	
A + B + C	A
$A \geq B, C \geq A$ and $B + kC \leq (k+1)A$: $(k+1)A \leq (k+2)A \leq B + (k+1)C,\ k = 0..\infty$:	
B + nC - (n - 2) A, n≤k+2	C
3C, otherwise	
$B \geq A, B \geq C$:	
A+B+C, n = 1	B
2B + C, otherwise	
$B \geq A, C \geq B$:	
A+B+C, n = 1	C
B+2C, n = 2	
3C , otherwise	

Figure 3. Latency and cycle time for a 3 processor pipeline

```
REPEAT
    Simulate processes keeping
    track of the running time of
    each process
    When two processes need to
    synchronize, compare their
    running times
    IF the relation between
    times cannot be determined
    THEN
        FOR all possible
        relationships between
        the times
            Assume that
            relationship
            holds and
            simulate with
            that assumption
    ELSE
        Synchronization will
        occur at the later of
        the two times
UNTIL sufficient data has been
accumulated
```

Figure 4. Algorithm for Analytical Simulation

further in section 3.3.

3.2. Implementing the Analytical Simulation algorithm

The full extent of the Analytical Simulation method has yet to be described, however it is useful to describe a number of implementation issues at this point before any additional complexity is introduced.

An implementation of the Analytical Simulation algorithm was constructed to perform the analysis of the systems described later in this paper. The emphasis for these models was to provide facilities for the analysis of parallel processes communicating using message passing. Other architectures may also use the algorithm, however, all distributed virtual reality systems known to the authors are based around a message passing paradigm [3]. The simulation need only model the duration of sequential processing, and inter-processor communication to provide results suitable for performance analysis. The simulation language consists of a few simple commands:

send	send a message to another process
receive	receive a message from another process
think	the process performs sequential processing for a specified time

A variable name can be associated with each command to represent the duration of sequential processing, or the communication time. These variable names will appear in the output of the analysis.

A separate set of variables, evaluated by the simulator, may be used as arguments for these commands. The receive command can be used non-deterministically by using an uninstantiated variable as the name of the source process. In this case the message received will be from the first process to attempt to send. In the case of more than one process fulfilling this requirement, all possible alternatives will be examined. A number of simple flow-control constructs are provided as well. The system is assumed to consist of a number of processes each consisting of an infinite loop surrounding a sequence of these commands.

Synchronization occurs when two processes attempt to communicate. The run times of each of the processes is a linear expression consisting of the sum of a number of *think* times and, possibly, communication times. Deciding which is the greater involves comparing the two in the presence of assumptions about the interrelationships of various other expressions.

Making the comparisons turns out to be a reasonably complex problem and the solution is discussed in the next section.

3.2.1. Comparing run times

Implementation of the Analytical Simulation algorithm depends on the ability to compare the local times of the various processes when represented as a linear expression, constrained by a number of inequalities, in turn composed of linear expressions. This comparison is required not only where shown explicitly in the algorithm, but also for selecting the first process on non-deterministic *receives* as well as for identifying extremes in the results.

The method for comparing linear expressions is presented below:

$$\text{Let A} = \sum_{i=1}^{n} \mathrm{e}_i \mathrm{X}_i$$

$$\text{Let B} = \sum_{i=1}^{n} \mathrm{f}_i \mathrm{X}_i$$

The problem is to determine if A≥B, A≤B or no known relationship exists, given

$$\begin{aligned} \sum_{i=1}^{n} \mathrm{g}_{ij} \mathrm{X}_i &\geq \sum_{i=1}^{n} \mathrm{h}_{ij} \mathrm{X}_i \qquad \mathrm{j} = 1 \ldots \mathrm{m} \\ \mathrm{X}_i &\geq 0 \qquad\qquad \mathrm{i} = 1 \ldots \mathrm{n} \end{aligned}$$

This problem can be restated more simply by assigning $\mathrm{a}_i = \mathrm{e}_i - \mathrm{f}_i$ and $\mathrm{b}_{ij} = \mathrm{g}_{ij} - \mathrm{h}_{ij}$ as:

Determine if $\sum_{i=1}^{n} \mathrm{a}_i \mathrm{X}_i \geq 0$, ≤ 0 or if no known relationship exists, given:

$$\begin{aligned} \sum_{i=1}^{n} \mathrm{b}_{ij} \mathrm{X}_i &\geq 0 \qquad \mathrm{j} = 1 \ldots \mathrm{m} \\ \mathrm{X}_i &\geq 0 \qquad \mathrm{i} = 1 \ldots \mathrm{n} \end{aligned}$$

The inequality $\sum_{i=1}^{n} \mathrm{a}_i \mathrm{X}_i \geq 0$ will hold if it can be written as a linear combination of the assumptions with only positive coefficients. The case $\sum_{i=1}^{n} \mathrm{a}_i \mathrm{X}_i \leq 0$ can be reduced to the previous case if rewritten as $\sum_{i=1}^{n} -\mathrm{a}_i \mathrm{X}_i \geq 0$. Thus it is only necessary to attempt to solve the first case.

A simple transformation reduces the problem to one with a known solution. Attempting to find the desired linear combination produces an expression of the form shown in (1), where $\mathrm{w}_i \geq 0$ $\mathrm{v}_i \geq 0$, and A, B, V, W and X are matrices whose components are the a_i, b_{ij}, v_i, w_i and x_i respectively. The w_i are the required coefficients and the v_i are slack variables to enforce the inequality.

$$X\,(IV + BW) = XA \tag{1}$$

Solving for the v_i, and using the requirement that each $\mathrm{v}_i \geq 0$, produces (2). This is the standard form of the constraints in a linear programming problem [19]. The complete solution to the linear programming problem is not required, however, it will suffice to find a single point in the feasible region. The existence of such a point implies the existence of a positive coefficient linear combination of the assumptions.

$$BW \leq A \tag{2}$$

$$\mathrm{w}_i \leq 0 \tag{3}$$

Finding this single point uses the Two-Phase Method described in [9]. This method consists of introducing two sets of slack variables and applying the simplex method to remove one set. The success or failure of the simplex method is dependent on the existence of a solution.

If a relationship between A and B can be found then the next step in the simulation of the model is well defined. If no relationship exists, then an additional constraint is introduced, specifying the relationship between A and B. Since two possibilities exist (A≥B, or B≥A), each must be introduced in turn and both branches simulated. Resolving non-determinacy can cause a number of constraints to be introduced at one point, splitting the simulation path in more than two ways.

3.3. State space extensions to the Analytical Simulation technique

The Analytical Simulation approach as described previously suffers from two significant limitations:

- Non-determinism causes the possible simulation paths to increase, often exponentially. This makes thorough analysis of the results time consuming and increases the computing resources required to perform the analysis.
- The simulation is only performed for a limited number of steps. Any characteristics of the model that are not present in this portion of the execution trace will be ignored.

The systems being modelled in this paper, virtual reality systems as well as real-time systems in general, are usually cyclic. This periodic nature means that the states of the program will repeat. The reachability graph of program states will be finite, and thus the problem of selecting a cut-off point for the Analytical Simulation algorithm falls away.

Non-determinism creates states with multiple outgoing arcs. If the resultant nodes do not occur in cycles then this behaviour is only transient, and can be identified as such.

When the node occurs in a cycle, then it can be identified as a recurring state. Since the complete state space can be explored, all aspects of the model can be examined.

Before examining the methods for analyzing state space graphs a working definition of the state of a parallel program is given.

3.3.1. Defining the state of a parallel program

The state of a conventional sequential program consisting of a sequence of instructions can be specified by providing values for the program counter and all variables defined in the program. These variables include the registers and stack used for executing the program.

If one considers a number of sequential programs running simultaneously, a parallel program in which no interaction occurs between processes, then the state of the parallel program can be given as a tuple, containing the states of the individual processes. Once synchronization constructs are introduced however, then this is no longer sufficient. A field giving the local time of each process is required.

As explained previously, the periodic nature of the programs can produce cyclic state space graphs of the program execution. Adding in a field giving the absolute time for each process would prevent this, since time is monotonically increasing. Instead relative values are used. One process is used as a reference and set as the origin of the time axis in each state. The times of the other processes are given relative to the reference.

Report markers in the program are used to mark states at which important events occur, such as the beginning or end of a timing period for measurement of latency or cycle time.

3.3.2. Exploring state space

Cycle times can be found by calculating the time it takes for a state containing a state marker to recur. Usually a state marker will indicate a point at which the system will produce output.

Latency is slightly more complex to calculate. Latency can be found by calculating the time taken to go from a state where a first marker occurs, to a corresponding state where a second marker occurs. Latency measures the time taken for information to move from one state to another. Thus the notion of corresponding markers requires that there be a message sent from the process with the first marker, after that marker is executed, and before it is executed again. This message must arrive at the process with the second marker. Time measurement ceases with the first execution of the second marker after the message arrives.

3.4. Use of Analytical Simulation

3.4.1. Area of application

The development of the analytical simulation approach has produced a number of enhancements to the analysis process, which have tended to limit the applicability of the approach to specific categories of programs. Ultimately, the approach is intended for use on message passing, virtual reality systems, and is capable of that in all its manifestations. In its least sophisticated versions, analytical simulation is applicable to many other areas.

The original algorithm is applicable to any architecture and model. At this stage, the only requirement is the ability to simulate the program, and to determine the length of the execution path for each process when synchronization occurs. With this very general approach there is no indication of when the analysis should terminate.

The next refinement, discussed when considering the implementation details, was to limit the simulation to models of message passing architectures. This relied on the fact that all virtual reality system surveyed had used message passing as their communication method. Having made this decision, the modelling language could be specified and the simulation engine could be implemented.

The next refinement allowed for automatic termination of the analysis and allowed for finite analysis in the presence of non-deterministic constructs. This state space analysis requires that the region of state space that could be reached by the model (reachability graph) is finite. For real-time systems which do not terminate, this requirement means that the model must be periodic.

3.4.2. Limitations

The Analytical Simulation approach to performance modelling does have limits in its applicability.

- Analysis requires the presence of repeated values, thus the state space of the systems being modelled must be cyclic.
- The size of the search through the state space can be substantial, especially if there are many points at which a non-deterministic choice is possible.
- Human intervention is still required to determine effects of different numbers of processors.

3.4.3. Advantages

Given that the analytical simulation approach has some limitations, it also has several features which are not found in other performance analysis and prediction tools.

- It generates metrics suitable for performance analysis of virtual reality systems.
- It produces output as a symbolic expression, allowing the effects of the variables in the model to be clearly identified.
- Automatic constraint generation removes the need to specify limits, or distributions for the variables.
- It supports non-deterministic constructs, which other approaches [11] have trouble with.
- Transient analysis is straightforward using analytical simulation.

4. Verifying the Analytical Simulation Technique

This section will apply the analytical simulation approach to a common problem in distributed virtual reality systems, that of collision detection. This problem provides a thorough test of the support of a virtual reality system for distributing the database representing the virtual world, and of its ability to control and synchronize independent processes.

A number of approaches to collision detection are discussed in [6]. These can be implemented in a numbers of ways on a parallel architecture. A common technique for distributing data in virtual reality systems is to use a client-server approach [3] [17]. An outline of an algorithm for performing collision detection, distributed using a client-server approach is given in [4]. It simulates a collection of point molecules in a closed container.

The server maintains a database of the position of every molecule. Each molecule is controlled by its own process, one of the clients. Each client calculates the time at which it will collide with each of the other objects, requesting a copy of the database from the server to do so. The earliest time is selected and sent to the server which relays the time of the earliest collision back to all the clients. Each client can then simulate the motion of its molecule until the time of this first collision. The server database is then updated, and the process repeated.

This section will examine performance of the client server model on Ethernet. It is assumed that only the processes in the system being modelled have access to the network cable. The model does not implement other Ethernet protocols such as packet collision detection and random backoff. Monitoring of cable transmission shows that these effects occur extremely infrequently when the communication medium is used for single, synchronized systems such as this.

N	Cycle time/[ms] Practice	Cycle time/[ms] Theory	% Theory / Practice
1	22.7	22.1	97.7
2	26.6	25.8	97.0
3	31.9	29.6	92.6
4	38.1	35.2	92.5
5	44.7	40.8	91.2

Table 1. Performance of buffered Ethernet Client-Server system

The analytical simulation implementation uses synchronous communication. Asynchronous communication, such as for Ethernet, is modelled using buffer processes. The client-server algorithm is such that each message requires an answer, so each client needs to be able to buffer only one message, and the server needs to buffer at most N, where N is the number of clients.

Measuring communication time (C) gave a value of 2.9ms, with a packet size of 1000 bytes. At the specified bandwidth of 10Mbits/s, transmission should have taken only about 1.0ms. The extra 1.9ms measure included extra time required to get the message through the hardware and a rather extensive array of network drivers. Two values S1 and S2 are introduced into the model to represent overheads on sending and receiving messages from the network respectively.

Examination of the packet transmission for a variety of different communication patterns revealed further interesting behaviour introduced by the underlying network software. This software contained some complex buffering mechanisms which complicated the inter-packet transmission times. The behaviour of the send operation depends on the time of the last communication. If the network driver was still occupied in sending the last message, the new message was placed straight into a buffer and the sending process could continue immediately. If the driver was idle, the sending process was required to block for a period S1 before the message was placed on the wire and the sender was allowed to continue. The buffered message could not be sent as soon the wire is idle again, instead it had to wait an additional S1 seconds.

This asymmetric communication complicates the model. Fortunately this is limited to the server process, since the client processes never send two messages in quick succession. The enhanced model is shown in Appendix A.1. The time required for simulation of particle motion in the client is represented by the variable X, the response time of the server by variable Y.

A comparison between the measured and predicted results from this model are shown in Table 1.

The theoretical values are less than those measured as

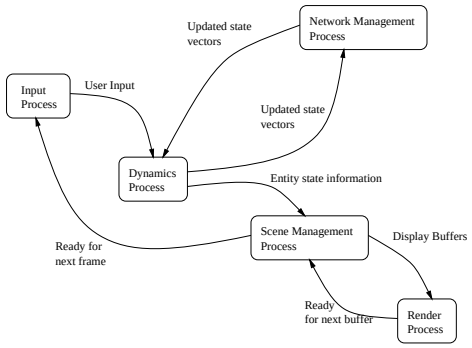

Figure 5. Data flow in an NPSNET node

may be expected, since the model ignores various small overheads. Some of the differences are due to the precision in the measurement of the variables; the accuracy of measurement is estimated at about 5%. There is inhomogeneity in the machines involved. Variations in the values measured on the machines (X and Y) were about 10%.

Results of equivalent or better accuracy are achieved using other architectures and communication protocols [4].

5. Simulation of a Virtual Reality System

The section describes the analytical simulation of a model of a single node of a virtual reality system. It illustrates the power of the approach for providing simulation results which are applicable to all values of the variables in the system.The model is that of the software running on a single machine of the well known NPSNET virtual reality system [14]. The relationships between the components of the system are illustrated in Figure 5. The interactive performance of the single node is of interest, inter-processor communication is modelled very coarsely.

The model for this is given in Appendix A.2. The variables D, S and R represent the time spent simulating the dynamics, managing the scene and rendering each frame of graphical output respectively. The variable C is the time between updates from the other nodes in the network. The two values of interest in a virtual reality system are the latency and the cycle time. The cycle time for this node is the interval between recurrences of the *output* report marker, the latency is the time taken for data created at the point indicated by the *input* report marker to reach the *output*.

The results for the analytical simulation of this model are as follows:

$S \geq C$, $D+S \geq R$:	Cycle Time = D+S
	Latency = R+S+D
$S \geq C$, $R \geq D+S$:	Cycle Time = R
	Latency = 2R
$C \geq S$, $D+C \geq R$:	Cycle Time = D+C
	Latency = R+C+D
$C \geq S$, $R \geq D+C$:	Cycle Time = R
	Latency = 2R

The results clearly show for which variable values the various performance characteristics will apply. The variables which affect the performance in any region can also be easily identified, and their effect clearly seen.

6. Conclusions

This Analytical Simulation approach to performance analysis has been described in detail. It has been shown that it possesses characteristics that make it suitable for application to distributed virtual reality systems. Extensions to the initial algorithm were discussed which improve the performance and allow a complete analysis of the performance characteristics of the model.

A comparison between the results predicted by an analysis of a complex client-server model and those achieved in practice was presented. The predicted values agree extremely well with those measured from the implementations, demonstrating the accuracy of the approach.

The approach was demonstrated by simulating a node in a distributed virtual reality system. The critical metrics for virtual reality systems, latency and cycle time, were easily obtained. The results obtained characterized the performance of the system for all variable values in the model.

References

[1] Wil M.P. van der Aalst, Using Interval Timed Coloured Petri Nets to Calculate Performance Bounds, *Proceedings of the 7th International Conference of Modelling Techniques and Tools for Computer Performance Evaluation*, G. Haring and G. Kotsis (eds), Lecture Notes in Computer Science, Springer-Verlag, New York, Vol 794, 1994, 425-444.

[2] Vikram S. Adve, Charles Koelbel and John M. Mellor- Crummey, Performance Analysis of Data Parallel Programs, Technical Report CRPC-TR94405, Center for Research on Parallel Computation, 1994.

[3] Shaun Bangay, Parallel Implementation of a Virtual Reality System on a Transputer Architecture, MSc Thesis, Department of Computer Science, Rhodes University, November 1993.

[4] Shaun Bangay, Modelling Parallel and Distributed Virtual Reality Systems for Performance Analysis and Comparison, PhD Thesis, Department of Computer Science, Rhodes University, November 1996.

[5] Rich Gossweiler, Robert J. Laferriere, Michael L. Keller, and Randy Pausch, An Introductory Tutorial for Developing Multiuser Virtual Environments, *Presence, Teleoperators and Virtual Environments*, 3(4), 255- 264.

[6] Philip M. Hubbard, Interactive Collision Detection, *Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

[7] Roger Hubbold, Alan Murta, Adrian West and Toby Howard, Design Issues for Virtual Reality Systems, *Proceedings of the First Eurographics Workshop on Virtual Environments*, Barcelona, September 1993.

[8] K. Kant, *Introduction to Computer System Performance Evaluation*, Mc-Graw-Hill, New York, 1992.

[9] Bernard Kolman and Robert E. Beck, *Elementary Linear Programming with Applications*, Academic Press, New York, 1980.

[10] Michael R. Macedonia and Michael J. Zyda, A Taxonomy of Networked Virtual Environments, *Proceedings of the 1995 Workshop on Networked Realities*, Boston, MA, October 1995.

[11] Celso L. Mendes, Performance Prediction by Trace Transformation, *Fifth Brazilian Symposium on Computer Architecture*, Florianopolis, Brazil, September 1993.

[12] Celso L. Mendes, Jhy-Chun Wang and Daniel A. Reed, Automatic Performance Prediction and Scalability Analysis for Data Parallel Programs, CRPC/Rice Workshop on Automatic Data Layout and Performance Prediction, Houston, April 1995.

[13] Praveen Murthy, On the Optimal Blocking Factor for Blocked, Non-Overlapped Schedules, Memo. No. UCB/ERL M94/46, Electronics Research Laboratory, College of Engineering, University of California at Berkeley, June 1994.

[14] David R. Pratt, A Software Architecture for the Construction and Management of Real-Time Virtual Worlds, PhD Thesis, Naval Postgraduate School, Monterey, California, June 1993.

[15] Matthew Regan and Ronald Pose, A Low Latency Virtual Reality Display System, Technical Report 92/166, Department of Computer Science, Monash University, Monash, 1992.

[16] Gurminder Singh, Luis Serra, Willie Png, Audrey Wong and Hern Ng, BrickNet: Sharing Object Behaviours on the Net, *1995 IEEE Annual Virtual Reality International Symposium*, Research Triangle Park, North Carolina, March 1995.

[17] Michael Snoswell, Documents on Cyberterm, available via anoymous ftp from ftp.adelaide.edu.au as /pub/cybertem/ctdocs.zip.

[18] Sukhamoy Som, Roland R. Mielke and John W. Stoughton, Prediction of Performance and Processor Requirements in Real-Time Data Flow Architectures, *IEEE Transactions on Parallel and Distributed Systems*, Volume 4, Number 11, November 1993.

[19] Gilbert Strang, *Linear algebra and its applications*, Academic Press Inc., New York, 1976.

[20] Martin R. Stytz, Distributed Virtual Environments, *IEEE Computer Graphics and Applications*, Volume 16, Number 3, May 1996.

[21] Qunjie Wang, Mark Green and Chris Shaw, EM - an Environment Manager for Building Networked Virtual Environments, *Proceedings of IEEE Virtual Reality International Symposium '95*, North Carolina, March 1995.

A. Appendix

A.1. Model of Ethernet Client-Server System

```
replicate N
 process serrec#
  receive clisen# MESSAGE
  think S2
  send server MESSAGE
  receive client# MESSAGE
  send medium wantmedium
  send medium givemedium
  send serrec# MESSAGE
 process cliserrec#
  receive comm somedata
  think S2
  send client# somedata
endreplicate
process buffersend
 send bufferlist get
 receive bufferlist DEST
 send numsending bufinc
 receive numsending ok
 think S1
 send comm DEST
```

```
process bufferlist
 receive CLIENT MESSAGE
 if MESSAGE == get
  if LENGTH == 0
   assign READY [READY+1]
  endif
  if LENGTH != 0
   send buffersend BUF[HEAD]
   assign HEAD [((HEAD+1))%(N-1)]
   assign LENGTH [LENGTH-1]
  endif
 endif
 if MESSAGE != get
  if READY == 0
   assign BUF[TAIL] MESSAGE
   assign TAIL [((TAIL+1))%(N-1)]
   assign LENGTH [LENGTH+1]
  endif
  if READY != 0
   assign READY [READY-1]
   send buffersend MESSAGE
  endif
 endif
process numsending
 receive CLIENT MESSAGE
 if MESSAGE == get
  send [CLIENT] [COUNT]
 endif
 if MESSAGE == inc
  assign COUNT [COUNT+1]
 endif
 if MESSAGE == bufinc
  if COUNT != 0
   assign ISWAIT 1
  endif
  if COUNT == 0
   assign COUNT 1
   send buffersend ok
  endif
 endif
 if MESSAGE == dec
  if ISWAIT == 0
   assign COUNT [COUNT-1]
  endif
  if [ISWAIT] != 0
   send buffersend ok
   assign ISWAIT 0
  endif
 endif
process comm
 receive SOMEONE DEST
 send medium wantmedium
 send medium givemedium
 send [DEST] somedata
 send numsending dec
process odi
 receive server DEST
 send numsending get
 receive numsending SENDING
 if [SENDING] == 0
  send numsending inc
  think S1
  send comm DEST
  send server sent
 endif
 if [SENDING] != 0
  send bufferlist DEST
  send server sent
 endif
replicate [N]
 process client#
  report start_client#
  send clisen# reqdata S1
  receive cliserrec# somedata
  send clisen# reqsummary S1
  receive cliserrec# somedata
  think x
  send clisen# reqsummary S1
  receive cliserrec# somedata
endreplicate
process server
 receive CLIENT MESSAGE
 if MESSAGE == reqdata
  think y
  send odi cli[CLIENT]
  receive odi sent
 endif
  assign COUNT [COUNT+1]
  if COUNT == [N+1]
   replicate N
    think y
    send odi cliserrec#
    receive odi sent
   endreplicate
   assign COUNT 1
  endif
 endif
process medium
 receive SOMEONE wantmedium
 think c
 receive [SOMEONE] givemedium
```

A.2. Model of NPSNET node

```
process input
 report input
 send dynamics data
 receive scene next
process network
 think C
 send dynamics data
 receive dynamics update
process dynamics
 receive input data
 receive network data
 think D
 send network update
 send scene state
process scene
 receive dynamics state
 think S
 receive render ready
 send render display
 send input next
process render
 send scene ready
 receive scene display
 think R
 report output
```