Performance Characterization of Distributed Virtual-Reality Applications: A Case Study

Leonidas Deligiannidis, Alva L. Couch, Robert J.K. Jacob
Department of Electrical Engineering and Computer Science
Tufts University, MA
{ldeligia, couch, jacob}@eecs.tufts.edu

Abstract

DLoVe is a new paradigm for designing and implementing distributed and non-distributed Virtual Reality applications using one-way constraints. We introduce a useful measure of asynchronous real-time distributed applications such as Virtual Reality. We define performance as the latency of the data in each rendered frame in addition to the overall frame rate instead of total throughput. This paper looks specifically at performance characterization for VR, and explains its unique requirements. In this context, we are not concerned only with throughput but also with how it looks to the user. Our perspective calls for a new approach to performance measurement. Our experience with DLoVe demonstrated that introducing a higher-level declarative User Interface Description Language (UIDL) provides extra degrees of freedom in the underlying implementation. Rather than causing a performance penalty, the constraint language made possible the increased performance we obtained with parallel processing. We demonstrate performance characteristics in realistic virtual environments using quantitative performance measurements.

Keywords: Performance, Virtual Reality, Distributed Systems

1 Introduction

Constraints in programming languages, user interface toolkits, databases, database queries, simulation packages, and other systems allow programmers to state declaratively a relation that is to be maintained, rather than requiring programmers to write procedures to maintain the relation themselves [12]. Defining a problem in terms of constraints helps define a new style of programming, in that the focus is on computing data values instead of writing methods [22]. DLoVe defines, with the help of constraints, a new style of programming. It introduces a new paradigm of specifying and executing applications, designed in its framework, in a distributed environment for performance improvement. It also gives one the ability to share Virtual Environments among multiple users.

Distributed multi-user applications are complex because pieces of such applications typically run in separate address spaces, often on heterogeneous networks of machines. The pieces must communicate and synchronize with each other, sharing and replicating data as needed, and must handle users' interaction at each site. A well-known example is the Diamond Park [30][7] that was designed at the Mitsubishi Electric Research Laboratory (MERL). Diamond Park is a social virtual reality system in which multiple geographically separate users can speak to each other and participate in joint activities in a mile-square virtual prototype. RENDEZVOUS is a language and architecture to help people build interactive multi-user systems using constraints and callbacks [28][29]. Visual Obliq [16] is a user interface development environment for constructing distributed, multi-user applications using callbacks, and distributed callbacks. Systems such as SimNet are using Dead Reckoning [25], a technique where the position of an object can be extrapolated from its previous position and velocity - developed by DARPA in the early 1980's.

Although Dead Reckoning works well in DIS, it is not suitable for general-purpose VR systems where the behavior of the objects and the users in a Virtual Environment is unpredictable. Dead Reckoning works well in DIS because the position of every object can be extrapolated from its previous location. The History-based approach [32][33] also uses Dead Reckoning but in a more efficient way than DIS. The history-based approach offers smooth, accurate visualizations of remote objects. The effectiveness of the DIS protocol is limited since tracking relies on acceleration information. An object's acceleration can change more rapidly than its position, so, if packets are delayed the DIS algorithm is likely to use out-of-date information to predict object behavior. The position history-based protocol is potentially superior to the DIS protocol for tracking non-

smooth object motion. The DIS protocol is highly sensitive to sudden acceleration changes because the algorithm utilizes only the most recent update information. Better performance on these non-smooth paths makes the history-based protocol more useful than the DIS protocols in virtual reality applications and visual simulations where entities move in unpredictable ways [32] [33]. Another software that implements Dead Reckoning is the Log-Based Receiver-Reliable Multicast (LBRM) [11]. The application chooses a threshold according to the freshness requirement of the data being disseminated. Shortening the threshold results in fresher data, but more network traffic. For entities with strict real-time delivery requirements, the threshold must be small. In the LBRM approach, reliability is provided by a logging server that logs all transmitted packets from the source. When a receiver detects a lost packet, it requests the missing packet from the logging server [11]. In TCP, the buffered data at the sender is effectively a log of transmissions, from which acknowledged packets have been flushed. The DIS system uses multicasting, and that approach seems promising for more general distributed VR application as well. But multicasting is not universally implemented. An effort is underway to link small pockets of multicast-capable machines to each other over the Internet; the result is a "multicast backbone" or MBONE. The MBONE system uses "tunneling"; it wraps the IP packets destined for a multicast address inside another IP packet, which travels through the regular Internet from one MBONE subnet to another [6]. Many others utilized multicasting protocols such as NPSNET [19] and DIVE [5].

2 Specification Paradigm

DLoVe is a specification paradigm that allows programmers to easily define and implement Virtual Reality programs in the context of a one-way constraint engine. In addition, it provides a framework where computationally expensive tasks can be performed in parallel in a distributed environment and improve performance. Most of the time, a program utilizing DLoVe's paradigm can be executed in a distributed environment with hardly any code modifications. The DLoVe specification paradigm is described in more detail in [18][24]; this paper is focused on the performance measurement techniques. In concert with its ability to distribute load over several machines, it also provides the mechanics to implement multi-user interfaces. There are two types of machines in the network (when a DLoVe application is executed in the distributed mode): Coordinators, (we have multiple Coordinators in a multi-user environment where multiple users share the virtual environment) render displays and read user input, while Workers perform calculations on behalf of the Coordinators.

DLoVe's constraint engine is mainly based upon incremental demand-driven constrained-solving algorithms found in Eval/vite [10] and algorithms presented in [3]. A modification of these algorithms produced the DLoVe's constraint engine that satisfies the real-time requirements of VR. The underlying elements of DLoVe's constraint engine are C++ objects called Links and Variables. Links implement behavior, and Variables are intermediate value-placeholders.

DLoVe is based upon a two-component model for describing and programming the fine-grained aspects of non-WIMP (non- Window Icon Mouse Pointer) interaction such as virtual environments. Its programming model is based on the notion that the essence of a non-WIMP dialogue is a set of continuous relationships, most of which are temporary. This model combines a data-flow or constraint-like component for the continuous relationships with an event-based component for the discrete interactions, which can enable or disable individual continuous relationships. Other current Graphical User Interfaces (GUIs) or (WIMP) interfaces, also involve parallel, continuous interaction with the user. But, most other user interface description languages and software systems are based on serial, discrete, token-based interaction. DLoVe is designed to provide a fundamentally continuous, rather than discrete, treatment of naturally continuous phenomena such as time and motion. However, it does treat discrete events as such and provides a mechanism for communication between the continuous and the discrete sub-system.

2.1 Continuous Time Subsystem

DLoVe's continuous time sub-system consists of object elements that define the relationship between objects through Links and Variables. The entire set of these elements connected together form a constraint graph. Changes on one end of the graph propagate to the other end. Interaction that is conceptually continuous is encoded directly into these elements, and thus the application does not need to deal with tracking events from conceptually continuous devices. Examples of conceptually continuous interaction include, drinking from a cup in a virtual world, throwing a ball in a virtual park, and driving a car in a virtual city. The body of a Link specifies how its Variables are related. Links can be enabled or disabled in response to user inputs. When a Link is disabled, it is as if the Link was not part of the constraint graph anymore, even tough the global structure of the constraint graph does not vary. By enabling and disabling Links we can quickly change/re-wire the constraint graph on the fly since only a flag needs to be set or cleared to indicate that a Link is enabled or disabled. Figure 1 shows the Variables as circles and the Links as rectangles. Links are enabled by default. A DLoVe graph is read from left to right. For example, the

Variables ‘A’ and ‘B’ are inputs to Link L1, and the Variable ‘C’ is its output Variable.

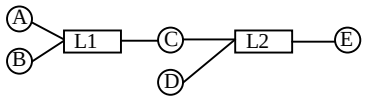

Figure 1. Constraint Graph

A disabled Link is still part of the data structure and when it becomes enabled again it knows how it is supposed to be attached to its Variables. Enabling a Link is similar to asserting a constraint, and disabling a Link is similar to retracting a constraint from the constraint graph. Even though enabling and disabling Links is similar to asserting and retracting constraints respectively, enabling and disabling can be performed rapidly; both operation only mark a Link to indicate whether it currently belongs to the graph or not.

2.2 Discrete Time Subsystem

There are other interactions that are fundamentally discrete (event-based) and are implemented in the event-based component of DLoVe. Such examples include button presses, menu choices and gesture recognition verifications. TBAG applications [4] generally deal with such discrete input events by retracting some existing constraints and asserting new ones. Bramble uses a similar mechanism [20]. DLoVe instead handles the discrete time using Event handlers, which are objects that capture tokens and respond to them. Event handlers contain a user-specified body that describes the response to tokens. The application sends a token to all event handlers, and only those event handlers that are interested in the token execute their body. The responses might include setting Variables, making custom procedure calls, and enabling/disabling Links. Event handlers recognize states and state transitions, and can provide different services depending on the state they are in.

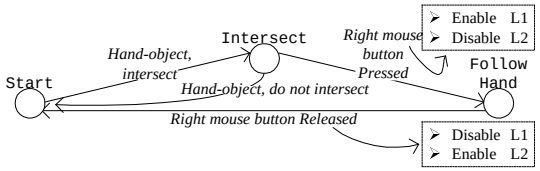

Figure 2. State Diagram of the Event Handler

For example, in figure 2, the user might intersect his/her virtual hand with a virtual object. The event handler will receive an INTERSECT token and will change state to its ‘intersect’ state. In this state if the user presses the left mouse button, the event handler enables one or more Links and changes state to the ‘dragging’ state. As a result, the object will be now attached to the virtual hand and the object will follow the movement of the virtual hand. When the user releases the mouse button, the event handler disables the previously enabled Links, transitions to the ‘start’ state, and the continuous relationship hand-object is terminated

3 Parallel Processing

There are three main differences between DLoVe and other parallel systems. While DLoVe’s tasks appear externally similar to those in PVM [35], task allocation is done at compile time, so that there is not appreciable overhead for task management. The purpose of task allocation, in DLoVe, is to allow the Coordinator to always request the same Variables from the same Workers. In other words, the queries the Coordinator sends to the Workers are partitioned, so that the Workers can execute multiple different queries in parallel. DLoVe’s task handling, unlike PVM or MPI [21] [23] [26] [27], is designed to support multi-user, multi-input application development by simply adding Coordinators. The third difference concerns performance requirements. DLoVe is designed primarily for Virtual Reality applications and thus requires high frame rate and also that the frames must be as fresh, or otherwise up-to-date, as possible. Distributed applications using DLoVe’s framework are characterized by real-time computations and constraints. Thus, not only number of evaluations or else total throughput, but also timing factors need to be taken into account when evaluating DLoVe [14]. Timing constraints in DLoVe arise from interaction requirements between the Coordinator and the user, and between the Coordinator and the Workers. The communication between the Coordinator, the user, and the Workers is described by three operations: sampling, processing, and responding. The Coordinator continually samples data from the input devices. Sampled data is sent to the Workers that process it immediately. Then the Workers send the processed data back to the Coordinator in response to its request. All three operations must be performed within specified times; these are the timing constraints [14] [34].

4 Communication Protocol

After all Workers connect to the Coordinator, the Coordinator partitions the constraint graph and assigns sub-graphs of it to different Workers. This partition of the graph enables the Coordinator to request Variables in parallel. Figure 3 shows a simple graph that is partitioned for three Workers[18] [17].

The Coordinator can request the Workers to ‘`set`’ and ‘`get`’ up-to-date Variables. It can also request the Workers to ‘`enable`’ or ‘`disable`’ Links, which re-wire the constraint graph. Each of these operations is a message and can be sent to the Workers individually. However, to utilize better the network, the Coordinator bundles multiple of these messages, set, get, enable, disable, into a single message called a multi-message, since it contains multiple messages/requests.

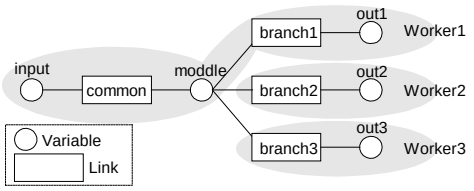

Figure 3. Partition into three sub-graphs

The Coordinator sends these multi-messages asynchronously to the Workers without blocking for replies. If the replies do not reach the Coordinator on time, it uses the previous values of the Variables to render the display. Similarly, to better utilize the bandwidth, Workers bundle multiple replies in a multi-message and send it to the Coordinator.

If '`get`' operations involve the evaluations of computationally expensive Links, then the Workers may not be able to catch up with the requests coming in from the Coordinator. The immediate problem in this case would be a congested network, since the Coordinator transmits requests as fast as possible – sending messages asynchronously without the notion of flow control. That is why the Coordinator keeps track of the number of pending '`get`' requests that exist in the network. If the number of the pending '`get`' requests exceeds a threshold, it simply discards the multi-message and continuous. That is fine in Virtual Reality as long as we drop below the threshold soon and replies keep coming from the Workers satisfying the VR timing constraints, in which case users stay immersed in the Virtual Environment. This technique is similar to TCP-probing[36], a modified version of the current TCP Transport layer protocol. When TCP-probing detects that data segments are lost, instead of re-transmitting and adjusting the congestion window, it initiates a probe cycle and suspends transmission. It resumes transmission as soon as the probe-acknowledgments get back to the sender in time. In DLoVe, the Coordinator instead of entering a Probe state where transmission is suspended, it simply drops the multi-message, in the Application layer, and never transmits it. This is acceptable and tolerable in Virtual reality as long as there are not too many dropped messages, since the Coordinator will request the same Variables at the next loop iteration. DLoVe can tolerate this kind of drops. However, if a multi-message contains `enable`/`disable` requests, which re-wire the constraint graph and thus these messages are considered important, the Coordinator does send the multi-message even if it detects that there are many pending message/requests on the network. That is also acceptable since Enable/Disable requests are triggered by the discrete time subsystem and not the continuous. These events/requests are triggered for example when the user grabs an object, or when he/she is looking at an object for over 5 seconds and the object becomes selected. TCP-Real [37] is a modification to the TCP protocol that could be of potential use for DLoVe. The TCP-Real receiver determines the size of the congestion window by estimating the level of contention within the network, thereby, avoiding unnecessary transmission gaps. In DLoVe, the Coordinator keeps track of pending 'get' requests and adjusts its transmission rate accordingly. However, the Coordinator does not distinguish between a congested network and busy Workers.

5 Performance Evaluation

Distributed computing attempts to increase processing speed by computing several tasks on otherwise autonomous computers at the same time [14][15]. To evaluate how well a distributed system behaves one must evaluate its performance, by comparing it against other ways of achieving the same result. In practice, these comparisons are difficult to make. The most often quoted measure of performance in parallel and distributed systems is *speedup*. The speedup is computed by dividing the time to compute a solution to a certain problem using one processor, by the solution time using N processors.

Amdahl's Law [1] [8] [26] states that the speed of a program in execution on a multiple-processor computer is limited by its slowest sequential part. A program contains two types of calculations, those that must be done serially, and those that can be executed in parallel on an arbitrary number of processors. If the time taken to do the serial calculations is some fraction β of the total time τ, $0 < \beta \leq 1$, then the parallelizable portion is $(1-\beta)\tau$ of the total time τ. Amdahl's law was challenged by John Gustafson and Ed Barsis, who showed that for some problems the regularity of the problem can feed as many parallel processors as are needed. Therefore, by adding processors, the size of matrix calculations can grow without bound [8] [2]. Gustafson and Barsis [13] show that problem size s and N are not independent of each other. Gustafson-Barsis law is relative to Amdahl's law but with an assumption about the problem size. Gustafson-Barsis law says that the size of the problem and the number of processors increases together, thus by adding more processors the speedup can increase accordingly.

Two factors influence the goodness of a graph partition. A given graph should be partitioned into concurrent modules to obtain the shortest possible program execution time. Secondly, one must choose the best size for each concurrent module that will result in fastest execution, while using a minimal number of processors. The grain-packing problem is related to optimal scheduling where an "optimal" scheduler executes in minimum time, well known to be NP complete in general. DLoVe's partition algorithm may be compared against load balancing and scheduling, but in reality it is someplace in between. Load Balancing, and more precisely Dynamic Load Balancing,

tries to keep all processors equally busy, but does not try to reduce overall execution time. Only when the calculations are more expensive than the communication, the applications may run faster [9]. Dynamic Load Balancing handles process migration and reacts to conditions that vary in the network. DLoVe's partition algorithm determines at compile time how to partition the constraint graph. DLoVe does not modify the partition of a given graph dynamically to efficiently implement Dynamic Load Balancing. It tries to partition the graph assuming that all Links are equally computationally expensive, which is not a great idea.

DLoVe assigns tasks to Workers in a manner similar to a scheduler. A single task in a scheduler corresponds to a single Link where in DLoVe a task corresponds of a set of interconnected Links. DLoVe uses the constraint graph itself to give extra information, making determining task allocation easier. Though a real scheduler allows tasks to have any cost, DLoVe assumes that all Links comprising a task have the same cost. This may cause DLoVe to create unbalanced task schedules but seems to work fine for many constraint graphs used in Virtual Reality.

5.1 Strategy for analysis

Two sets of experiments were conducted to measure the performance of DLoVe. The first set tested the performance of a Virtual Park simulating 32 Humanoids, entities that walk, run, interact with each other and the user, and play with a ball [18]. We measured the results of the non-distributed and the distributed versions using up to three Workers and to assess performance we counted the number of Link evaluations. We also measured the resulting frame rate at which the Coordinator was able to render the graphics on the display. We purposely implemented a computationally expensive collision prevention and behavior algorithm for the Humanoids in order to overcome the network bandwidth bottleneck. This experiment employed a traditional approach for measuring performance: the overall computational throughput for the system. These experiments, however, did not measure how up-to-date each rendered frame was during each computational cycle. These experiments were conducted on Silicon Graphics workstations using SGI Performer as the graphics-rendering library. The four machines involved had differing hardware architectures with different processor type, memory size and cache. In addition, all Workers mounted the Performer run-time libraries of the Coordinator via a Network File System (NFS), adding even more overhead to the network. The Network was a shared 10Mbps LAN. The scheduler's task assumption that all Workers possess the same throughput is false, so that scheduling is less optimal than it would be in a homogeneous environment.

Because traditional techniques for performance analysis do not accurately describe VR performance we performed a second set of experiments. These experiments used a new approach for measuring the performance of VR systems, by quantifying the accuracy of each frame rendered on the display. We defined the *accuracy* of a frame as the difference between the wall clock time and the minimum time since all output Variables (used for rendering the display) were last updated. Unlike the first set of experiments, these ones were conducted within a homogeneous environment. For this set of experiments we used a very simple simulation executed on Sun ultra 5 workstations running Solaris 2.7. We measured the latency of each network message as well as the number of requests initiated, the number of replies, and the number of messages that were discarded. This allowed us to compute the accuracy of each frame rendered.

5.2 Analyzing the Results

In DLoVe, when most of the replies from the Workers to the Coordinator are too late, users lose the feeling of real-time interaction and can become disoriented and confused even if the frame rate stays high, since rendering is performed using old values. In this case frame rate alone does not accurately describe the efficiency of the rendering system. We found that by adding more Workers the total throughput of the system increases but we get a decrease in frame rate as a penalty [18].

DLoVe utilizes the TCP transport protocol for all its communication. The Coordinator needs to send all multi-messages to all Workers, which is essentially a simulation of multicasting. Since the Coordinator has to send all these additional multi-messages to the additional Workers it spends more time transmitting and less time rendering the display. In all cases, the distributed version outperformed the non-distributed version, whose frame rate and total throughput were very low compared to the distributed version, because one machine had to calculate all computationally expensive collisions and behavior, and also render the display. If DLoVe outperforms the non-distributed version using TCP/IP for point-to-point communication, it promises even greater performance if re-implemented using multicasting.

The analysis of the Virtual Park, even though it illustrated some of the positive and negative aspects of DLoVe, was conducted within a heterogeneous environment. The outcome of the experiments might have been different if all machines were exactly of the same architecture and all possessed a local copy of the Performer run-time libraries. Clearly we needed a more carefully conducted procedure for conducting the experiment to get more conclusive and more accurate results. To address this, we implemented a very simple application that was executed in a homogeneous environment, and made accurate measurements of its performance. We were able to measure latency of individual messages sent by the Coordinator, and the number of messages that were built and discarded by the Coordinator. This benchmark

program contained four Links and five Variables. It was designed so that DLoVe can partition it into at most three Workers (since we only had four machines of the same type in our possession – one for the Coordinator and three for the Workers). Figure 3 shows the Links and Variables within this application and how they were partitioned to the three Workers:

5.3 State machine of the run time system

The Coordinator can be thought of as possessing five independent states as shown in Figure 4.

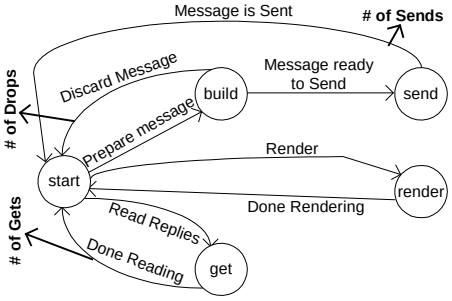

Figure 4. State machine

The Coordinator builds a message (multi-message) consisting of several requests, and sends it to all of the Workers. Initially, the Coordinator is in the '`start`' state. When it sees a need to '`set`' or '`get`' a Variable's value or to enable or disable a Link, it transitions to the '`build`' state within which it builds the message that it may be sent to the Workers. At the end of the main loop, just before sending the message, it checks internal counters that indicate the number of pending requests on the network to each Worker. If a counter is below a chosen threshold, it transitions to '`send`' state, sends the message, and returns to the '`start`' state. Else, it discards the message and returns to the '`start`' state. While in the '`start`' state, it checks to see if any replies came back from the Workers. If replies have arrived, it transitions to the '`get`' state and processes all the replies. When all replies are processed, it transitions back to the '`start`' state where is starts building messages all over again. The '`start`' state is a state indicating idle time and it is used as a starting point in describing the functionality of the Coordinator. The fifth state is the '`render`' state in which the Coordinator transitions to render the display.

5.4 What is Measured

The experiments were run for 60 seconds, where DLoVe collected two kinds of performance statistics every second. Statistics included the number of requests initiated by the Coordinator, the number of messages that were discarded, and the number of replies that came back to the Coordinator. Additional performance data included latency of each request. Every request was time-stamped with the time of initiation before being sent to the Workers. The Worker that responded to the requests preserved the timestamp in its reply. The Workers also time-stamped each reply message with the amount of time it took the Worker to update the requested Variable. When the Coordinator received the reply, it calculated the elapsed time of the request and subtracted the amount of time the Worker needed to update the requested Variable – measuring the time a message spends on the network. Figure 4 shows the state transitions where we measured how many messages the Coordinator sends, discards, and receives every second.

Performance in time required measuring the elapsed time between pairs of state transitions as shown in figure 5.

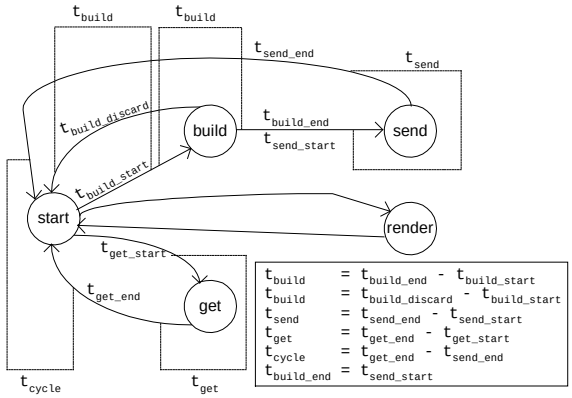

Figure 5. Message Measurements

t_{build} is the time it takes the Coordinator to build a message before sending it to the Workers or discarding it, which is the difference between t_{build_end} and t_{build_start} , or between t_{build_start} and $t_{build\_discard}$ representing the time at which the build ended and started. t_{send} is the time it takes the Coordinator to put the message on the network, which is the difference between t_{send_end} and t_{send_start} indicating the time the Coordinator completed the transmission of the message and the time it started transmitting. t_{get} is the time it takes the Coordinator to process the replies from the Workers. t_{cycle} indicates the time it takes a request to come back as a reply, which is the difference between t_{send_end} and t_{get_end}. t_{cycle} is the latency of each message and this is what in which we are the most interested. For the Coordinator to receive many replies from the Workers is a goal we would like to achieve. However, if the messages are all too old, then user interaction is minimized and this is not what we want since DLoVe is designed for real-time applications such as Virtual Reality.

5.5 Analyzing the Results

Throughput of a Virtual Reality system does not describe how well it performs. A more critical factor in VR is the high frame rate it can achieve, in addition to how accurately each frame represents the virtual world. Accuracy of rendition is difficult to measure. In DLoVe, the Coordinator issues '`get`' requests to update certain Variables to render the display. These requests are sent to the Workers that update the requested Variables, which then send the results back to the Coordinator. These replies are not instantaneous, but arrive with some latency that depends on performance upon the network. So, one measure of accuracy is message latency. Another is how up-to-date the frames are when rendered, relative to user input and the real state of the virtual world.

5.6 Frame Validity and Statistical Skew

Latency shows how long a message spends on the network to get to the Workers and then come back to the Coordinator - time required by a Worker to evaluate a Link is not counted. However, this does not show how accurate the frame rendering is and it does not capture the interactive performance of such user interfaces. The system might be doing a great deal of processing by generating many frames, but displaying each of them late. It would score well by traditional parallel processing measures, but would seem very sluggish to the user. We therefore used total throughput along with total latency of a displayed video frame to measure performance. That is, when a frame is displayed, what is the oldest piece of data that was used to generate that frame? We believe that frame latency, in conjunction with frame rate, captures the interactive performance of a VR user interface better than total processing throughput. Frame latency shows how valid or how late the information is at the time the Coordinator renders the display.

To perform this analysis, we tracked each message through the network. Message latency shows how long a message spends on the network to get to the Workers and then to come back to the Coordinator (minus the time required by a Worker to bring the requested Variable up-to-date). Message latency is the time this request spent in the network. However, this does not show how accurate a rendered frame is. To visualize how valid the frames are, we used a statistical clock skew that plots the oldest information used to render the display:

```
skew = (wall clock) - min(time of request of all
                requested Variables)
```

For every frame, the minimum time of request of all output Variables is subtracted from the current time (wall clock). This skew describes the worst difference between what is rendered and what the user is doing. Figure 6 is one of many graphs that shows this (see [17] for many additional performance tests).

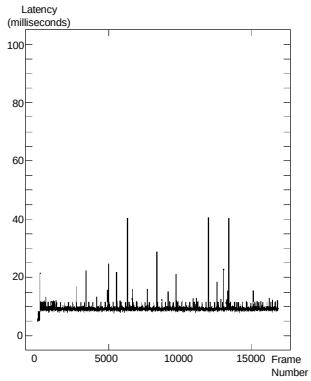

Figure 6. Frame Latency

6 Conclusions

The experiments demonstrated that DLoVe not only describes specification of Virtual Reality programs well, but also improves overall performance of applications designed in its framework by dramatically increasing the validity of the rendered frames. In addition, DLoVe supports mechanics for implementing or transforming single user programs into multi-user programs. DLoVe can be used to implement large-scale Virtual Reality applications, and when speed is required, DLoVe's framework is able to provide the additional CPU cycles needed by real time application by utilizing multiple machines.

We showed the need for a different method of measuring performance that describes DLoVe accurately, where traditional methods fail by providing seemingly acceptable performance measures that in actuality reflected poor performance. Throughput is not enough to describe performance, but throughput and frame rate in conjunction with statistical skew, which measures the freshness of each rendered frame, do accurately describe DLoVe's performance.

More experiments need to be conducted to understand how DLoVe behaves in high-speed networks and how multiple Coordinators influence the performance of DLoVe. However, because appropriate equipment was unavailable, these experiments must be deferred for future investigation.

References

1. Amdahl, G.M. Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485.
2. Berman, Kenneth A. and Paul, Jerome L. "Fundamentals of Sequential and Parallel Algorithms". PWS Publishing Company, 1997.
3. Brad Vander Zanden, Brad A. Myers, Dario A. Guise, Pedro Szekely., "Integrating Pointer Variables into One-Way Constraint Models". ACM Transactions on Computer-Human Interaction, v1, n2 June 1994 p161-213.

4. C. Elliott, G. Schechter, R. Yeung and S. Abi-Ezzi. “TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications”, In Proc. ACM SIGGRAPH ‘94, pages 421-434, August, 1994.
5. Carlsson, C., and O. Hagsand. “DIVE – A platform for multi-user virtual environments”. Computers and Graphics 17(6): 663-669, 1993
6. Casner Steve, Henning Schulzrinne, and David M. Kristol “Frequently Asked Questions on the Multicast Backbone (MBONE)”,’94, ftp://venera.isi.edu/mbone/faq.txt
7. David B. Anderson, John W. Barrus, John H. Howard, Charles Rich, Chia Shen, Richard C. Waters. “Building Multi-User Interactive Multimedia Environments at MERL”, IEEE MultiMedia, 2(4):77-82, Winter 1995.
8. Hesham El-Rewini & Ted G. Lewis, “Distributed and Parallel Computing”. Mannining Publications Co. 1998.
9. Hesham El-Rewini, Theodore G. Lewis, and Hesham H. Ali. “Task Scheduling in Parallel and Distributed Systems”. PTR Prentice Hall, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1994.
10. Hudson, A., “Incremental Attribute Evaluation: A Flexible Algorithm for Lazy Update”, ACM Transactions on Programming Languages and Systems, v13, n3, July 1991, pp. 315-341.
11. Hugh W. Holbrook, Sandeep K. Singhal, and David R. Cheriton, “Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation”, In Proceedings of ACM SIGCOMM Aug. 1995 p328-341. Published as Computer Communications Review, Vol 25, No. 4 Oct. 95.
12. Ivan Sutherland. “Sketchpad: A Man Machine Graphical Communication System”. Ph.D thesis, Massachusetts Institute of Technology, January 1963.
13. J. L. Gustafson, “Reevaluating Amdahl’s Law” chapter book, Supercomputers and Artificial Intelligence, Edited by Kai Hwang, 1988.
14. Jeffrey J. P. Tsai, Yaodong Bi, Steve J. H. Yang, and Ross A. W. Smith, “Distributed Real-Time Systems”, John Wiley & Sons, Inc. 1996.
15. Kevin Dowd & Charles R. Severance, “High Performance Computing” (Second Edition) RISC Architectures, Optimization & Benchmarks. O’Reilly & Associates, Inc, July 1998.
16. Krishna Bharat, and Marc H. Brown., “Building Distributed Multi-User Applications by Direct Manipulation”, UIST ‘94, November 2-4 1994 pages 71-81.
17. L. Deligiannidis, "DLoVe: A specification paradigm for designing distributed VR applications for single or multiple users" Doctoral dissertation, Tufts University, Feb. 2000
18. L. Deligiannidis, R. Jacob "DLoVe: Using Constraints to Allow Parallel Processing in Multi-User Virtual Reality", Proc. IEEE Virtual Reality 2002 Conference, IEEE Computer Society Press, March 2002.
19. Macedonia, Michael, R. Michael J. Zyda, David R. Pratt, and Paul T. Barham, “Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments”, Proceedings of IEEE Virtual Reality Annual International Symposium, 2-10, 1995.
20. Michael Gleicher. “A Graphical Toolkit Based on Differential Constraints”. UIST '93 November 1993, pages 109-120.
21. MPI Message Passing Interface Forum. “MPI: A Message-Passing Interface Standard”. International Journal of Supercomputer Applications and High Performance Computing, 8(3/4), 1994.
22. Myers, B. A., Giuse, D. A., and Vander Zanden, B. 1992. “Declarative programming in a prototype-instance system: Object-oriented programming without writing methods”. Sigplan Not. 27, 10 (Oct.), pages 184-200.
23. Peter S. Pacheco, Parallel “Programming with MPI”, Morgan Kaufman Publishers, Inc. 1997.
24. R.J.K. Jacob, L. Deligiannidis, and S. Morrison, "A Software Model and Specification Language for Non-WIMP User Interfaces," ACM Transactions on Computer-Human Interaction, vol. 6, no. 1, pp. 1-46, March 1999.
25. R.S. Johnston, "The SimNet Visual System," Proc. 9th Interservice/Industry Training Equipment Conf., Defense Technical Information Center, Washington. DC. Nov. ‘92.
26. Rajkuma Buyya, “High Performance Cluster Computing” Vol. 1 (Architectures and Systems), Prentice Hall PTR, New Jersey 1999.
27. Rajkuma Buyya, “High Performance Cluster Computing” Vol. 2 (Programming and Applications), Prentice Hall PTR, New Jersey 1999.
28. Ralph D. Hill, Tom Brinck, Steven L. Rohall, John F. Patterson, and Wayne Wilner, “The Rendezvous Architecture and Language for Constructing Multiuser Applications”. Transactions on Computer-Human Interaction v1,n2 Jun 1994 p81-125.
29. Raph D. Hill, “The Abstaction-Link-View paradigm: Using constraints to connect user interfaces to applications”. CHI ‘92, May 3-7, 1992 pages 335-342.
30. Richard C. Waters, David B. Anderson, John W. Barrus, David C. Brogan, Michael A. Casey, Stephan G. McKeown, Tohei Nitta, Ilene B. Sterns, William S. Yerazunis, “Diamond Park and Spline: A Social Virtual Reality System with 3D Animation”, Spoken Interactions, and Runtime Modifiability. Presence: Teleoperators and Virtual Environments, Nov 1996.
31. Roger Tagg and Chris Freyberg, “Designing Distributed and Cooperative Information Systems”, International Thomson Computer Press, MA 1997.
32. Sandeep K. Singhal, and David R. Cheriton, “Exploiting position history for efficient remote rendering in networked virtual reality”, Presence: Teleoperators and Virtual Environments, 4(2) p169-193, Spring 1995.
33. Sandeep K. Singhal, and David R. Cheriton, “Using a Position History-Based Protocol for Distributed Object”, Technical Report STAN-CS-TR-94-1505, Dep. Of Computer Science, Stanford University, Feb. 1994.
34. Selic, Bran and Gullekson, Garth and Ward, Paul T. “Real Time Object Oriented Modeling”. John Wiley & Sons, Inc., 1994.
35. V. S. Sunderam, PVM: A Framework for Parallel Distributed Computing, Concurrency: Practice and Experience, 2, 4, pp 315--339, December, 1990.
36. V. Tsaoussidis, H. Badr, "TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains" *The 8th IEEE Conference on Network Protocols*, ICNP 2000, Osaka, Japan, November 2000.
37. C. Zhang and V. Tsaoussidis, "TCP-Real: Improving Real-time Capabilities of TCP over Heterogeneous Networks", *11th IEEE/ACM NOSSDAV 2001*, ACM Press, Port Jefferson, NY, June 2001.