TITLE:

Efficient Synchronization of Large-scale Network Simulations

AUTHORS/AFFILIATIONS:
Alfred Park, Richard M. Fujimoto, Kalyan S. Perumalla
College of Computing, Georgia Institute of Technology
Atlanta, Georgia, USA 30332-0280
{park,fujimoto,kalyan}@cc.gatech.edu

CORRESPONDING AUTHOR CONTACT INFORMATION:
Alfred Park
College of Computing
801 Atlantic Drive
Atlanta, GA 30332-0280
Fax: (404) 385-2295
Email: park@cc.gatech.edu

KEY WORDS:
Parallel and distributed simulation, conservative synchronization, scalable, federated

ABSTRACT:

Parallel and distributed simulation tools are emerging that offer the ability to perform detailed, packet-level simulations of large-scale computer networks containing millions of end hosts and routers. This paper examines the synchronization mechanism that is required to ensure the correct execution of such parallel/distributed network simulation tools. Specifically, a variation of the classical null message synchronization algorithm is described as well as an analytical model to assess the efficiency and scalability of this synchronization protocol. An implementation of this protocol is developed using an approach to realizing parallel network simulations that reuses existing sequential simulation models. The implementation is based on the popular ns-2 network simulation tool, and is used to validate the accuracy of the analytic model for large parallel network simulations. The performance of the network simulator is evaluated, and compared with an alternate synchronization mechanism based on global reduction computations. Based on results from experiments on large-scale network simulations, the efficiency of the null message protocol is verified. These results suggest that this protocol yields scalable parallel performance when the simulated network can be partitioned among the parallel processors to maintain good load balance, lookahead and a relatively constant number of inter-processor cross-links as the model size and number of processors are increased in proportion.

NOTES:

Earlier versions of portions of this paper appeared in the PADS 2004 and MASCOTS 2003 conferences.

1. Introduction

Simulation is widely recognized as an essential tool to analyze networks. Although analytic methods are useful in many situations, the complexity of modern networks combined with the inability to apply simplifying assumptions in many analysis problems limit the applicability of purely analytic approaches. For example, it is well-known that Markovian traffic assumptions are often inappropriate and can lead to misleading results. Even when analytic methods can be used, simulation is often used to validate the analysis. It can be used to gain insight into the behavior of particular protocols and mechanisms under a variety of network conditions.

In certain cases, simulations of small to medium sized networks may be sufficient to gain critical insights into the behavior of the network. However, in other cases, simulation is needed to understand the detailed effects of a new protocol, policy, service, attack, or application when it is widely deployed on a large network such as the Internet. Interactions between such a newly proposed mechanism, and the intensity and variety of competing traffic on the Internet are important considerations in gaining such an understanding. For example, if one wished to understand the effect of enabling multicast on a large Internet Service Provider (ISP), much larger simulations with a variety of competing traffic are needed to gain credible evidence regarding the effect of this proposed change.

This paper examines the tools necessary for this latter class of research questions, namely, those that require simulations of large networks. Specifically, we are interested in quantitatively characterizing the ability of modern parallel simulation systems to perform detailed simulations of computer networks. We focus primarily on packet-level simulations that model the transmission and queuing of individual packets as they travel through the network. Such packet-level simulation is extensively used for protocol design and evaluation. It is widely used by many network simulation tools such as ns-2 [1], GloMoSim and its commercial version Qualnet [2], and Opnet [3], among many others.

Most studies performed today using packet level simulations are limited to experiments modeling hundreds to thousands of network nodes (e.g., routers and end host computers). A critical reason behind this limitation in configuration size is that the amount of computation time and memory required to perform simulations of much larger networks is prohibitive. The goal of this paper is to explore the limits of packet level simulation using parallel computation techniques and to address conservative synchronization issues when dealing with large-scale simulations distributed across many processors.

First, we will discuss network simulators and how their general architecture can often limit the modeling of large-scale networks. We then introduce methods for overcoming sequential simulator limits through parallel and distributed simulation techniques that allow for increased network model sizes, reduced execution time or both. We continue the discussion on parallel simulation techniques by further exploring conservative synchronization protocols that often accompany parallel discrete event network simulations. An analytical model for an optimized null message algorithm is then presented and verified, and results of an empirical performance evaluation study presented demonstrating scalability for large-scale network simulations modeling up to a million network nodes (routers and end hosts) distributed over hundreds of processors.

2. Scalability Limits of Network Simulators

In practice, two factors that typically limit the scale of packet level simulations that can be performed are the amount of memory required and the amount of computation time needed to complete each simulation run. Because the simulator requires a certain amount of memory to represent the state of each node, memory requirements increase at least linearly with the number of nodes. Memory requirements may increase faster than linearly in some simulators, e.g., to represent routing tables [4]. Thus, the total amount of physical memory on the computer performing the simulation acts as a limiting factor on the number of nodes that can be represented.

At the same time, the amount of time required to complete a packet level simulation usually increases in proportion with the amount of traffic that must be simulated. This is because packet-level simulators are usually based on a discrete event paradigm where the computation consists of a sequence of event computations. Events in a packet-level simulation represent actions associated with processing a packet such as transmitting the packet over a link. Thus the execution time in a packet level simulation is proportional to the number of packets that must be processed. Assuming the modeler is only willing to

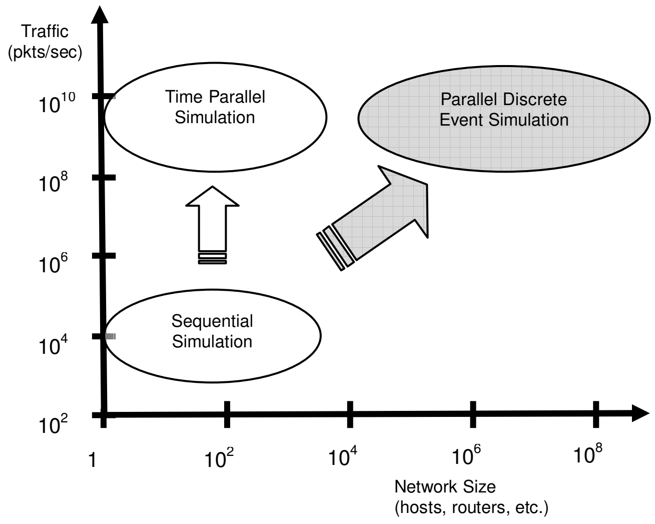

Figure 1. Notional diagram depicting network size (nodes and traffic) that can be simulated in real time

wait a certain amount of time for the simulation to complete, this places a limit on the amount of traffic that can be simulated.

The preceding observations imply specific limits on the scale (size and amount of traffic) that can practically be modeled by a given simulator using a specific hardware platform. A notional diagram illustrating these scalability limitations is shown in figure 1. In this figure we normalize the execution time constraint by requiring that the simulation *execute in real time*, i.e., simulating S seconds of network operation requires no more than S seconds of wallclock time. Each shaded area in Figure 1 represents a class of network simulators. Sequential simulation methods provide a baseline. Our experiments indicate that contemporary network simulators such as ns-2 with suitable optimizations to reduce memory consumption are able to simulate on the order of tens to hundreds of thousands of packet transmissions (transmission of a single packet over a single communication link) per second of wallclock time for networks containing thousands to tens of thousands of nodes on a modern workstation or personal computer.

Concurrent execution of the simulation computation on multiple processors can improve the scalability of the simulator both in terms of network size and execution speed, depending on the method that is used. Time parallel simulation methods such as those described in [5-11] parallelize a simulation of a fixed sized network by partitioning the simulation time axis into intervals, and assigning a processor to simulate the system over its assigned time interval. These methods increase the execution speed of the simulator enabling more network traffic to be simulated in real time, but do not increase the size of the network being simulated (see Figure 1). For example, performance on the order of 10^{10} simulated packet transmissions per second are reported in [11]. Parallel discrete event methods utilize a space-parallel approach where a large network is partitioned and the nodes of each partition are mapped to a different processor, offering the potential for both network size and execution speed to scale in proportion to the number of processors. As discussed later, several parallel simulators using this approach have been developed. This approach is the primary focus of this paper. Finally, in addition to these "pure" time and space parallel approaches, some approaches combine both time and space parallelism, e.g., see [12].

3. Parallel Network Simulation

Several parallel discrete event simulation systems have been developed to improve the scalability of network simulations. The traditional approach to realizing such a system is to create the parallel simulator "from scratch," where all the simulation software is custom designed for a particular parallel simulation engine. The simulation engine provides, at a minimum, services for communication and synchronization. Examples of parallel network simulators using this approach include GloMoSim [2], TeD [13, 14], SSFNet [15], DaSSF [16], TeleSim [17], and the ATM simulator described in [18], among others. One advantage of this approach is the software can be tailored to execute efficiently in a specific environment. Because new models must be developed and validated, this approach requires a significant amount of time and effort to create a useable system.

Another approach to parallel/distributed simulation involves interconnecting existing simulators. These federated simulations may include multiple copies of the same simulator (modeling different portions of the network), or entirely different simulators. The individual simulators that are to be linked may be sequential or parallel. This approach has been widely used by the military to create simulation systems for training or analysis, and several standards have been developed in this regard [19-23]. An approach linking multiple copies of the commercial CSIM simulator to create parallel simulations of queues is described in [24]. The federated approach offers the benefits of model and software reuse, and provides the potential of rapid parallelization of existing sequential simulators. It also offers the ability to exploit models and software from different simulators in one system [25].

Here, we focus on the latter approach. Specifically, we are concerned with the Parallel/Distributed Network Simulator (*PDNS*) based on the widely used ns-2 network simulation tool. *PDNS* uses the underlying runtime infrastructure (RTI) software that provides services for communication and synchronization (see Figure 2). The RTI and *PDNS* software will be described next.

3.1. RTI-KIT

RTI-Kit is a component of the Federated Simulations Development Kit (FDK). It provides a set of libraries for realizing runtime infrastructure (RTI) software that provides communication and synchronization services for parallel/distributed simulations. Specifically, the MCAST library provides group communication services for implementing publication/subscription communication among simulators (federates). The current version uses reliable, point-to-point communication to implement group communications. The TM-Kit library includes software to synchronize the computation, as will be discussed in greater detail shortly. The other modules shown in figure 2 implement mechanisms such as buffer management, priority queues, and low level communications support. The RTI-Kit software has been extracted from FDK and ported into a standalone library tuned for extensibility, high performance, and scalability. This library designed specifically for parallel and distributed applications is called *libSynk/RTI* [26, 27], and offers all RTI-Kit services detailed in figure 2 along with a basic RTI/HLA interface for backwards compatibility. Our simulation study utilizes *libSynk/RTI.*

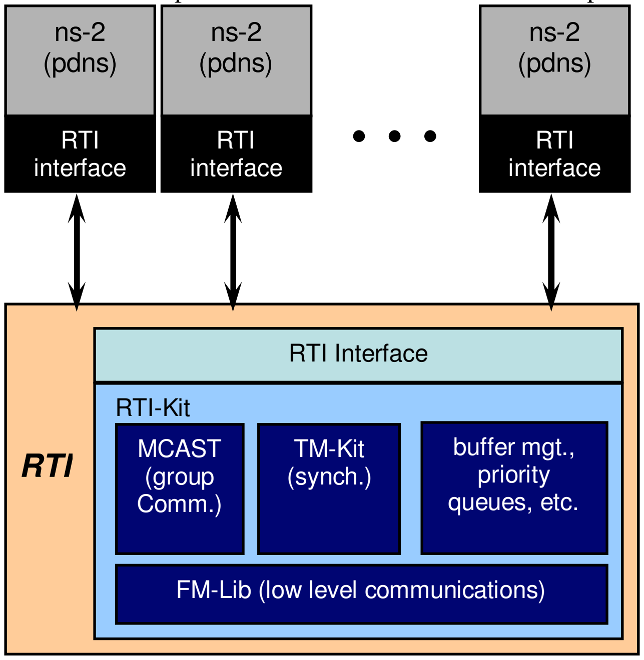

Figure 2. *PDNS*/RTI-KIT architecture

Synchronization mechanisms for parallel discrete event simulation can be broadly classified as conservative or optimistic. The principal responsibility of the synchronization mechanism is to ensure that each simulator (called a *federate*) processes events in time stamp order. Conservative mechanisms use blocking to ensure an event (message) is not processed until it can be guaranteed no event with a smaller time stamp will later be received. Optimistic mechanisms allow events to be processed out of time stamp order, but use some mechanism (e.g., rollback) to recover from such errors. These synchronization paradigms are discussed in greater detail in [28].

The initial version of *libSynk/RTI* used a synchronization mechanism based on using a distributed snapshot algorithm to compute the lower bound on time stamp of future message that might be received by processors. In addition to computing a global minimum, this algorithm must account for messages that have been sent, but have not yet been received, i.e., *transient messages*. Specifically, a variation of Mattern's algorithm [29] is used for this purpose. At the heart of this version of TM-Kit is the computation of reductions (global minimums) that account for transient messages using message counters. This computation and performance optimizations that have been developed are described in [26]. However, the *libSynk/RTI* software infrastructure can accommodate multiple synchronization schemes developed as modules. Another TM module based on null messages to synchronize the distributed simulation was developed with large-scale simulations in mind, as will be described later.

3.2. *PDNS*

PDNS is based on the ns-2 simulation tool, and is described in greater detail in [30]. Each PDNS federate differs from ns-2 in two important respects. First, modifications to ns-2 were required to support distributed execution. Specifically, a central problem that must be addressed when federating sequential network simulation software in this way is the global state problem. Each *PDNS* federate no longer has global knowledge of the state of the system. One ns-2 federate cannot directly reference state information for network nodes that are instantiated in a different federate. In general, some provision must be made to deal with both static state information that does not change during the execution (e.g., topology information), and dynamic information that does change (e.g., queue lengths). Fortunately, due to the modular design of the ns-2 software, one need only address this problem for static information concerning network topology, greatly simplifying the global state problem. *PDNS* uses a facility called *remote links* to refer to link endpoints that reside in a different processor, and a technique called NixVector routing to reduce the size of routing tables that are required to perform the simulation.

By federating sequential simulators such as *PDNS* via an RTI, one can develop larger models using existing software and tools. This approach favors conservative over optimistic synchronization because one need not add a rollback mechanism to the original simulator. Therefore, conservative synchronization issues must be scrutinized to ensure that parallel and distributed network simulations run efficiently over a large number of processors. An inefficient time management (synchronization) implementation or the application of an inappropriate synchronization protocol can dramatically reduce performance and prohibit instantiation or large-scale network models. We will now turn our attention to issues with conservative synchronization methods provided by the time management module in the RTI-Kit.

4. Conservative Synchronization Overview

As discussed earlier, in conservative PDES, an event cannot be processed unless it is safe to do so. Since all federates do not have a consistent view of the state of the entire system, federates must exchange information to determine when events are safe to process. Moreover, federates must process events in time stamp order so that the local causality constraint is preserved. This process of synchronizing federates has been the subject of much past research [28].

Conservative synchronization algorithms can be broadly classified as using global or local synchronization computations. Global algorithms rely on reduction computations such as that described earlier to synchronize with other federates in the simulation. Processors using these algorithms alternate between event computations and participation in global synchronization computations. During the event computation phases, the distributed simulation does useful work, and forward progress in simulation time is made. Synchronization computations involve all federates to compute a global minimum value referred to as the Lower Bound on Timestamp (LBTS). The LBTS value is a guarantee that there will be no future incoming events that will have a timestamp less than the LBTS value. This allows federates to safely process events in its event queue in time stamp order with timestamps less than the LBTS value. Although each federate computes its own individual LBTS value, the global minimum of LBTS values is computed over all federates through the reduction computation and this global minimum is used as the LBTS value by all federates. These global reductions may be performed synchronously, in which case all federates block until the reduction computation has completed, or asynchronously, “in the background” while federates perform event computations. The *libSynk/RTI* implementation uses the latter approach. By only processing those which have a timestamp less than LBTS one can guarantee that each federate only processes events in time stamp order. Well-known examples of algorithms using global synchronization computations include YAWNS [31], deadlock detection and recovery [32], and the bounded lag algorithm [33].

Local algorithms do not require global synchronization computations. Rather, such computations are performed in an asynchronous manner by different processors based on locally available information. A well-known example is the null message algorithm for deadlock avoidance originally developed independently by Chandy and Misra [34] and Bryant [35]. The original Chandy-Misra-Bryant (CMB) null message algorithm specifies that after processing an event, the federate must send a time stamped message containing a lower bound on the time stamp of future messages that can later be sent to neighboring federates. These null messages essentially contain LBTS values between neighboring federates instead of a global minimum between all federates in the previous (global synchronization) case. Consequently, simulations utilizing asynchronous null message algorithms are logically composed of federates that synchronize locally with other federates with which it directly interacts. Variants to the original CMB null message algorithm to improve efficiency have been devised and evaluated [36-39].

The focus here is to improve scalability of large-scale network simulations over many processors by optimizing the well-known CMB null message algorithm used for time management. Although previous studies have evaluated the performance of conservative synchronization algorithms for inefficiencies and overhead (e.g., see [40-42]), the scale of these studies have been, for the most part, limited to modest-sized configurations, with most using fewer than 100 processors. As the number of federates that participate in a parallel and distributed simulation increases, performance conclusions based on small-scale simulation studies may not apply to a large-scale context and the appropriateness of a particular synchronization method can shift from one algorithm to another. Here, we address scalability concerns in order to compare the local CMB null message algorithm with a global reduction-based algorithm for large-scale network simulations.

While there has been much research comparing optimistic and conservative synchronization protocols, there has been comparatively little work devoted to comparing the performance of local and global conservative synchronization algorithms for large numbers of processors. Bagrodia and Takai provide extensive performance comparisons, but do not consider large numbers of processors [36]. Nicol describes scalability through analysis of model characteristics, partitioning, workload balance, and model complexity in a broad PDES context [43]. However, our work specifically analyzes the detailed costs associated with the operation of a lazy null message algorithm. Further, we show that better scalability can be achieved using CMB, compared to global reduction algorithms, in scaled network models with constant fan-in/fan-out. This is performed with an emphasis on quantitative results supporting our findings. Nicol and Liu [44] combine asynchronous channel scanning and synchronous global barrier algorithms with a good amount of experimental data examining fan-in/fan-out and lookahead up to 8 processors. We discuss performance implications on larger configurations, specifically comparing lazy null message-based and global reduction-based algorithms. These algorithms are applied to regular and irregular network models with varying load distributions and asymmetric lookahead values, on up to 128 processors. Nicol also presents models to provide a lower-bound on the performance of a synchronous global reduction algorithm that can apply to large numbers of federates [45, 46]. Our analytic model differs from this work in its focus on scalability concerns for the CMB algorithm. While analyzers described in [47] illustrate speedup differences between local and global algorithms, our focus lies in identifying detailed CMB performance characteristics, such as null message send frequency and overhead, and applied to analytical models for large-scale network simulation. Synchronization challenges are described by Naroska and Schwiegelshohn [48] for large number of processes. While their work focuses on VLSI design, we focus on network simulations. The work presented in [49] contains models that exclude lookahead values [50]. However, their approach requires extensive knowledge of various metrics and has no means to measure the effect of increasing scale.

5. Analytical Models for CMB

Synchronization based on the CMB algorithm depends upon null messages exchanged only between neighboring federates. The frequency and time stamp of these null messages dictate the behavior and performance of the simulation; this simple observation will drive the formulation of the analytical CMB model. We will iteratively build this model. The end result will be a model for a version of the CMB null message algorithm that can approximate simulator performance in structured large-scale network simulations.

In the following we assume the parallel simulator consists of a collection of federates that communicate by exchanging messages. Additionally, we relax the constraints assumed in the original null message algorithm. Specifically, a federate need *not* send messages in time stamp order. We do assume that the communication channel delivers messages in the same order in which they were sent (FIFO). These assumptions imply that all simulation time information for computing new LBTS values resides in the time stamps of null messages.

5.1. Effect of Null Messages on Simulation Behavior

Lookahead is one of the most important factors to achieving good performance in conservative PDES. Each federate is allowed to process events without re-synchronizing with other federates up to its LBTS value. In general, conservative PDES performs well when the difference between successive LBTS values is large relative to the simulation time between successive events, because each federate can process many events before it must re-synchronize with other processors.

In a traditional CMB algorithm, after an event is processed, null messages are sent to every output channel to neighboring federates. It is well known that overly eager null message transmission schemes produce an excessive number of null messages, leading to other algorithms that improve CMB performance by sending null messages less often. However, a sufficient number of null messages must be sent in order to avoid deadlock. The frequency of null message sends can have a large effect on performance. As the number of null messages increases, the overhead to synchronize increases proportionately, and less useful work is accomplished each second of wallclock time. Lookahead, null message frequency, time advance conditions, and remote communication are all contributing factors in the analytical model characterizing the performance of algorithms based on null messages.

5.2. A Lazy CMB Null Message Algorithm

This variant of the CMB algorithm, which is traditionally referred to as *lazy CMB*, attempts to minimize the number of null messages by only sending them when absolutely necessary. Specifically, null messages are only sent when the federate reaches the end of its safe processing time, i.e., only when the federate must block. Failing to send null messages while in this blocked state could lead to deadlock situations.

Figure 3 shows the progression of the simulation using the lazy CMB algorithm where each federate has the same lookahead. Note that "LBTS" in the figure represents when a new LBTS value is computed for time management. This lazy null message execution resembles that of a time-stepped simulation since each federate is in perfect synchrony with each other. With a minimal number of null messages exchanged between federates, this scheme appears to be an efficient synchronization protocol. One can calculate the number of null messages N_{Φ} sent as:

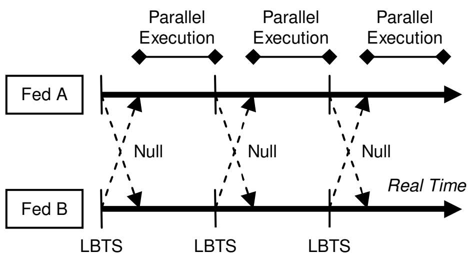

Figure 3. Execution Timeline of a Lazy CMB Algorithm

$$N_{\Phi} = m \times n \times \left(\frac{s}{lookahead} \right) \qquad (1)$$

where m is the number of federates, n is the number of output channels per federate, and s is the length of the run in simulation time. The number of execution windows (s divided by *lookahead*), gives the number of rounds or time-steps for which null messages are sent. Multiplying this result by the number of output channels yields the number of null messages sent each round. The product of this result and the total number of federates gives the total number of null messages exchanged during the entire simulation. It is important to emphasize that formula 1 pertains to a CMB algorithm applied to a regular, structured network model. In other words, the lookahead of each federate is assumed to be the same.

In our sample simulation, let $m = 8$, $n = 2$, $s = 25$, and *lookahead* = 0.2 seconds. By equation 1, the total number of null messages would be 2,000. We can modify equation 1 to approximate the number of synchronization messages (N_{Ψ}) sent when using global reductions based on a butterfly barrier (assuming that m is a multiple of 2):

$$N_{\Psi} = m \times \log_2 m \times \left(\frac{s}{lookahead} \right) \qquad (2)$$

This equation indicates the same simulation using global reductions and a butterfly barrier mechanism results in 3,000 synchronization messages. More generally, if the number of output channels does not increase with the number of federates, the number of null messages in CMB increases linearly with the number of federates, yielding lower overhead than the approach using global reductions.

It is important to note, however, that minimizing the number of null messages does not necessarily maximize performance. In particular, a modest load imbalance may lead to large waiting times that could be avoided by sending additional null messages. To see this, consider the example in figure 4, showing two federates A & B with output channels between them. Here, federate A determines that it can no longer process events safely. Federate A sends a null message to federate B and blocks. However, federate B is busy processing events within its own LBTS limit. Further, when federate B completes processing these

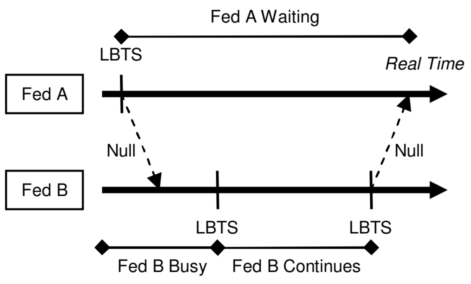

Figure 4. Blocking in a Lazy CMB Algorithm

events, it can process the null message from federate A, compute a new LBTS value, and begin processing events based on this new value, all *without* sending a null message to federate A. Because no new null message is sent to federate A, federate A will remain blocked. A null message will be sent to federate A only after federate B has finished processing these new events and is forced to block. Moreover, the delay to transmit this null message from federate A to federate B further increases the amount of time federate A remains blocked.

A solution to this problem is to have each federate send null messages more frequently. One must generate more null messages, but not so many that the system is burdened with an excessive number. This is discussed next.

5.3. An Optimized Null Message Algorithm

One approach to generating additional null messages is to designate that each federate should send null messages every *f* units of simulation time advance, where *f* is less than the federate's lookahead (continuing to assume, for the moment, that all links have the same lookahead; we will relax this assumption later). By increasing the frequency of null messages sent to neighbors, potentially long blocking times can be avoided.

A second optimization is to eliminate null messages that carry no new useful information. For example, if two successive null messages carry the same time stamp it is clear there is no reason to send the second. Thus, we can cancel (suppress) any superfluous null messages that convey no new information, by not sending those messages at all.

To include these two changes, we can modify equation 1 to yield:

$$N_{\Phi} = m \times n \times \left(\frac{s}{f}\right)(1-c) \qquad (3)$$

where c is the proportion of null messages cancelled and f is the frequency at which null messages are sent. Here, f is used in place of the lookahead value in equation 1. Due to the non-deterministic nature of the modified lazy null message algorithm, equation 3 provides only a lower-bound estimate on the number of null messages sent if lookahead is used for f. The quantity c cannot equal 1 nor can f equal 0 (which means all null messages are cancelled or no null messages are sent, respectively). If either of these conditions were true, then the deadlock avoidance guarantee of the CMB algorithm would be violated.

We now turn to the more general case of non-uniform lookahead on output channels. It is useful to construct an analytical model for which the simulation models are irregular. Here, we assume a lookahead value is assigned to each link. Increased concurrency in simulation execution can be gained using local synchronization if there exists a non-trivial difference in lookahead for multiple output channels per federate. By tailoring null messages to specific lookahead values for each output channel, we can offer better LBTS guarantees which are not artificially limited by other output channels on a federate which have smaller lookaheads. In the case of different lookahead values for different output channels to different federates, formula 3 can be modified to yield:

$$N_{\Phi} = \sum_{j=1}^{m}\sum_{k=1}^{n} \frac{s}{f_{jk}}\left(1-c_{jk}\right) \qquad (4)$$

where f_{jk} represents either the minimum lookahead on the outgoing channel to federate jk or the frequency of null message sends.

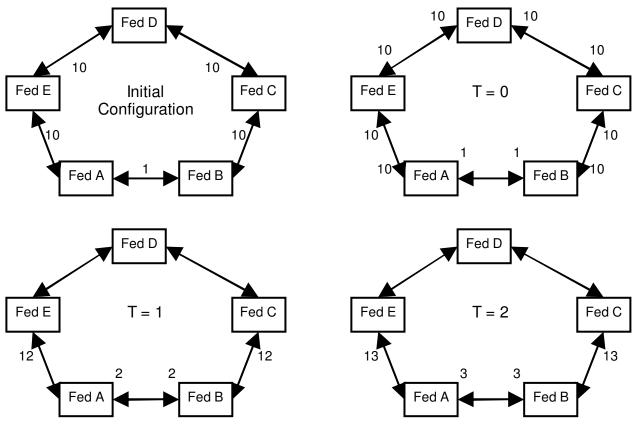

Figure 5. Example Illustrating the Operation of Null Message Exchange with Non-Uniform Lookahead

From a federate *j*, if there are multiple outgoing links to a neighboring federate *x*, they are consolidated into a single output channel with the lookahead assigned as the minimum of all outgoing links to *x*. In figure 5, a sample federate topology is constructed where the lookahead on all output links between federates is 10, with the exception of the link between federate A and federate B. Although the lookahead is 10 between federates A & E and federates B & C, we can see that both federate C and federate E receive null message updates from federates A & B at 1 time unit intervals. Therefore, the "localized effect" of non-uniform lookahead must be carefully considered before applying an estimate using equation 4. Equation 4 is applicable even in the case where all lookahead values are the same among all output channels, in which case the equation would then be equivalent to equation 3.

Equations 1-4 include lookahead and null message frequency to give an approximation of the number of null messages sent during the course of the simulation. As mentioned earlier, the number of null messages is useful, but it does not in itself offer an estimate on the efficiency of the synchronization algorithm. In addition to blocking, the type of communication used by the hardware platform on which the simulator executes (e.g. shared memory vs. Ethernet) and other related factors provide a better approximation on the overhead of a null message algorithm.

$$\alpha_{\Phi} = \frac{1}{a}\sum_{i=1}^{a}\ln\left[\frac{p_i w_i}{m}\left(\sum_{j=1}^{m}\sum_{k=1}^{n}\frac{s}{f_{jk}}\left(1-c_{jk}\right)\right)\right] \qquad (5)$$

Equation 5 represents an overhead index for the lazy null message algorithm. Here *a* represents the number of different types of remote communication used in the simulation, *p* is the percentage and *w* is the weight given to a specific type of remote communication. The quantity *w* is a normalization factor for different communication media. For instance, if we had remote communication over both shared memory (w_1) and Ethernet (w_2), a possible normalization would be w_1=1 and w_2=10 to represent shared memory being faster by an order of magnitude compared to Ethernet. The natural logarithm is applied to represent the index as a monotonically increasing function of the product of the types of remote communication used and the average amount of null messages sent and received per federate.

Utilizing these equations, one could compute not only the number of null messages that would be sent over all federates, but also the relative messaging overhead incurred with a particular null message algorithm or simulation scenario. This can be particularly useful when attempting to predict performance of a new scenario against an established baseline (e.g., scaled experiments).

As mentioned earlier, it is important to emphasize that raw null message numbers sometimes may not always provide the best approximation for synchronization overhead. Other factors such as load imbalance can also reduce parallel and distributed simulator performance. However, the analytic model allows for a simple synchronization overhead estimation with good accuracy as we will show later in our validation study.

5.4. Synchronizing Large-Scale Network Simulations

Equation 5 predicts that the overhead associated with these variations of the null message algorithm will remain relatively constant as a simulation scales, assuming the number of output channels for each federate does not increase with the number of federates, and all other factors (such as relative amount of remote communication and lookahead) remain unchanged. On the other hand, time management algorithms using global reduction operations have larger overheads as the simulation scales to a larger number of processors.

Notice that the only change from equation 1 to equation 2 is the substitution of $\log_2 m$ for *n*. For a global algorithm, the number of synchronization messages increases at a rate of $m\log_2 m$, compared to a rate of *m* for CMB. In addition, this cost for synchronous algorithms does not include the penalty of jointly interrupting and restarting *all* federates that increases linearly as the number of federates increases, contributing to the overhead of time management for *every* LBTS computation.

6. Applying the Analytical Null Message Model to Predict Overhead

Using the analytical model formulated in the previous section, we can forecast null message activity and overhead indices. These indicators can then be used to predict behavior and performance for different simulation models. We will first estimate null message activity and overhead and then verify our numbers through experimental measurements. In our experimental performance study, one federate corresponds to one physical processor (CPU).

6.1. Baseline Model

Here, we used benchmark networks developed at Dartmouth College as a set of baseline models for the network modeling and simulation community [51]. This baseline configuration was created to facilitate and demonstrate network simulator scalability.

Each portion of the network is referred to as a Campus Network (CN). Figure 6 shows the topology for the CN. Each CN consists of 4 servers, 30 routers, and 504 end hosts for a total of 538 nodes. The CN is comprised of 4 separate sub-networks. Net 0 consists of 3 routers, where node 0:0 is the gateway router for the CN. Net 1 is composed of 2 routers and 4 servers. Net 2 consists of 7 routers, 7 LAN routers, and 294 clients. Net 3 contains 4 routers, 5 LAN routers, and 210 clients.

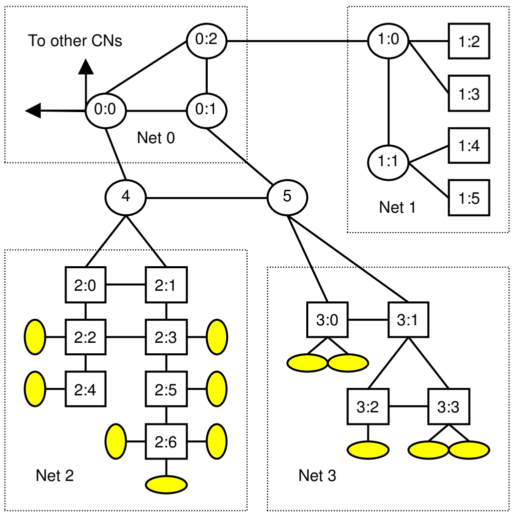

Figure 6. A Campus Network

All non-end host links have a bandwidth of 2Gb/s and have a propagation delay of 5ms with the exception of the Net 0 to Net 1 links, which have a delay of 1ms. End hosts are connected point-to-point with their respective LAN router and have links with 100Mb/s in bandwidth and have a delay of 1ms.

Multiple CNs may be instantiated and connected together to form a ring topology. This aspect of the network allows the baseline model to be easily scaled to arbitrarily large sizes. Multiple CNs are interconnected through a high latency 200ms "ring" link with 2Gb/s bandwidth via their Net 0 gateway router (node 0:0). The baseline model also contains chord links between CN *i* and CN *i + 4* where *i mod 4* is zero. A second set of chord links exist between CN *i* and CN *i + 10* where *i mod 2* is zero and *i* is less than half of the length of the ring. For our tests, we instantiated 7 CNs per federate, which is the maximum number of CNs representable within the total memory available on each processor. Thus, these experiments scale the size of the simulation model in proportion with the number of processors. Results from experiments containing up to 512 processors are reported, corresponding to a simulation modeling over 1.9 million network nodes.

In our analysis and performance evaluation, we focus on pure TCP traffic transferred to and requested by end hosts from server nodes. We use the short transfer case of the baseline model, where clients request 500,000 bytes from a Net 1 server. TCP sessions start at a time selected from a uniform distribution over the interval from 0 and 10 seconds of simulation time. The baseline model specifies for TCP traffic to be requested from the neighboring CN that is one ring link hop away. This traffic model exercises only the ring links and no chord links, consequently, we have modified the traffic scenarios for some of our test cases to allow end hosts to request data from any Net 1 server in the ring network at random.

6.2. Scenarios

The baseline model was enhanced to test the effects of irregularity and asymmetry on time management performance. The modifications are intended to exercise irregularity of both traffic and topology. The five scenarios (see Table 1) represent five contrasting configurations, providing a range of benchmarks. Random server selection and varying propagation delay on ring and chord links were parameters that were modified to create scenarios used in the performance study.

Three scenarios were drafted where no chord links were used. The *Baseline* model consists of standard ring link delays and standard server selection. Traffic restrictions for *Baseline-R* are not limited to the next neighbor but any server in the ring network. *Baseline-2ms* is similar to the previous scenario, but has a single 2ms ring link that spans two federates. The other two scenarios contain chord links to add more paths to the network topology. *Chord-R* contains standard 200ms ring and chord delays with random server selection. *Chord-Asym* contains asymmetric, random chord delays from 10-50ms.

Table 1. Benchmark Scenarios

Scenario	Traffic	Chord?	Special Properties
Baseline	Std.	No	None
Baseline-R	Rand.	No	None
Baseline-2ms	Rand.	No	Single 2ms ring link
Chord-R	Rand.	Yes	None
Chord-Asym	Rand.	Yes	Asym. Chord delays

6.3. Software and Hardware Platforms

The aforementioned *PDNS* network simulator was used in the performance study. *libSynk/RTI* was used as the underlying communications and time management library to manage the *PDNS* federates. The five benchmark scenarios were tested on two different hardware platforms:

Intel Pentium III Linux Cluster (P3) – Consists of 16 8-way 550MHz Pentium II Xeon SMP machines with 4GB RAM connected via Gigabit Ethernet. The operating system is Red Hat Linux 7.3 running a customized 2.4.18-27.7.xsmp kernel. *libSynk/RTI* and *PDNS* (based on ns-2.26) were compiled using gcc-3.2.3 with compiler optimizations for the Intel Pentium III architecture.

Intel Itanium 2 Linux Cluster (IA64) – Consists of 30 2-way 900MHz Itanium 2 SMP machines with 6GB RAM connected via Gigabit Ethernet. The operating system is Red Hat Advanced Workstation 2.1 running a 2.4.18-e.31smp kernel. *libSynk/RTI* and *PDNS* (based on ns-2.26) were compiled using Intel's ia-64 C/C++ compiler 7.1 (ecc).

Scalability tests for the *Baseline* model were also conducted on the Pittsburgh Supercomputing Center's "Lemieux" *Compaq Alpha Tru64 Cluster* (PSC). The cluster has 750 4-way 1GHz Alpha ES45 SMP nodes with 4GB RAM connected via a Quadrics switch. *libSynk/RTI* and *PDNS* (based on ns-2.1b9) were compiled using gcc-3.0.

6.4. Approximating Null Message Transmissions

Using the analytical model for CMB algorithms, the number of null messages sent can be estimated. Table 2 shows null message activity estimates for the *Baseline* scenario along with measurements from the parallel simulator. In this scenario, s=25, *lookahead*=0.2, f=(0.2/3), n=2, and c=0.5.

Table 2. Null Message Send Estimations

Number of CPUs	Approximation	Measurement
4	1,500	1,424
8	3,000	2,974
16	6,000	6,064
32	12,000	11,890
64	24,000	23,586
128	48,000	48,628
256	96,000	96,034
512	192,000	193,862

Table 2 shows that our analytical model predicts the null message counts quite accurately. The discrepancy between the approximation and simulation numbers are due to the estimation of the null frequency and null message cancellation rate. It is difficult to predict the exact interaction between event processing and the effects on null message frequency and cancellations, yet the approximation is very close to the actual null message activity.

6.5. Quantifying Simulator Performance

Just as computer hardware designers utilize metrics such as the number of floating point operations a machine can process per second of wallclock time, it is useful to have a quantitative metric to characterize the performance of packet-level network simulators. Since the bulk of the simulation computation in a packet level simulator involves simulating the transmission and processing of packets as they travel from the source, through intermediate nodes (routers), to its destination, it is convenient to use the number of *Packet Transmissions that can be simulated per Second of wallclock time* (or PTS) as the metric to specify simulator speed. This metric is useful because given the PTS rate of a simulator, one can estimate the amount of time that will be required to complete a simulation run if one knows the amount of traffic that must be simulated, and the average number of hops required to transmit a packet from source to destination.

Specifically, the execution time T for a simulation run can be estimated by the equation

$$T = \frac{N_T}{PTS_S} \qquad (6)$$

where N_T is the number of packet transmissions that must be simulated and PTS_S is the number of simulated packet transmissions that can be simulated per second of wallclock time by simulator S. N_T depends on several factors. A first order estimate is

$$N_T = N_F \times P_F \times H_F \qquad (7)$$

where N_F is the number of packet flows that must be simulated, P_F is the average number of packet transmissions per flow, and H_F is the average number of hops a packet must traverse in traveling from source to destination. It is important to note, however, that this is an approximation because it does not consider packet losses nor other non-user traffic packets that must be sent by the protocol being used, e.g., acknowledgement packets in TCP. Nevertheless, when these considerations are accounted for, these equations provide a reasonable means to estimate execution time.

When the objective is to achieve real-time performance, the execution time constraint is PTS_S must be at least as large as

$$N_F \times R_F \times H_F \qquad (8)$$

where R_F is the average rate of packets transmitted (packets per second) per flow, again with the same caveats concerning losses and protocol-generated packets. For example, consider a simulation of 500,000 active UDP flows in real time where each flow produces traffic at a rate of 1 Mbps. Assuming 1 KByte packets, this translates to 125 packets/second. If the average path length is 8 hops, the simulator must execute at a rate of 500 Million PTS to achieve real time performance.

6.6. Approximating Null Message Overhead

Utilizing the previous estimation results for the number of null messages sent and equation 3 derived from the analytical model, it is possible to estimate the synchronization overhead incurred for each scenario.

Simply comparing the overhead index for a particular scenario to its corresponding run time performance (wallclock seconds) is inadequate. This is because the elapsed time to run a simulation does not accurately gauge the amount of work performed by the network simulator. Instead, we will utilize the PTS metric defined previously to determine work performed by the simulator, and PTS is adequate to compare against the overhead index as the PTS rate is directly affected by time management overhead.

The overhead indices and PTS rates for 32 CPUs are shown in Table 3. For each of the scenarios, it is known that 28 CPUs (87.5%) send null messages through shared memory buffers and the remaining 4 CPUs (12.5%) communicate via TCP/IP over Ethernet. Using the values m=32, p_1=0.875, p_2=0.125, w_1=1, and w_2=10, the overhead index for the *Baseline* scenario is 6.25.

Table 3. Overhead Approximation

Scenario	Overhead Index	PTS
Baseline	6.25	1,512,410
Baseline-R	8.52	865,385
Baseline-2ms	10.54	621,008
Chord-R	9.43	699,848
Chord-Asym	12.21	684,630

Clearly, an inverse relationship between the overhead index and PTS metric exists. The lower the overhead costs for the null message algorithm, the higher the performance of the particular scenario. However, the *Baseline-2ms* and *Chord-Asym* models appear to exhibit conflicting results. The *Chord-Asym* model has a higher overhead index and still manages to have a higher PTS rate than the *Baseline-2ms* model. This anomaly illustrates that the overhead index measures only penalties associated with time management, not for other performance degrading parameters that are a result of the model itself. Although the overhead index does capture lookahead, it does not consider other factors such as load imbalance and the amount of remote communication due to event transmission. Nevertheless, the overhead index can be used as an approximation tool to predict time management efficiency and thus, simulator performance.

6.7. Scalability of the CMB Algorithm

In section 5.4, we stated that, provided all parameters for the scenario remain constant, the overhead due to the null message algorithm does not change as the simulation model increases in proportion with the number of processors. We verify the same experimentally here for the *Baseline* scenario. Due to the dynamic nature of the other scenarios, it was impossible to keep other factors such as remote communication due to events and lookahead constant between scalability runs.

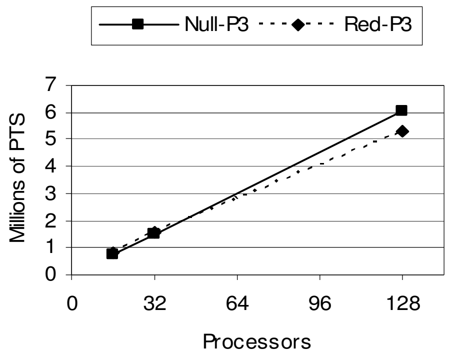

Figure 7. Effect of Scale on PTS Rate

Table 4. Run times for the *Baseline* Scenario

Number of CPUs	Null Messages	Reductions
16	784	736
32	783	747
128	787	892

The run time using a CMB null message algorithm remains constant when the number of processors is increased from 16 to 128, as shown in Table 4, indicating the computation does scale as the model size is increased in proportion with the number of processors. Execution times based on global reductions steadily increase from 16 to 128 processors while remaining relatively constant for null messages. Figure 7 ("Red" denotes global reductions) shows the PTS rate for the two synchronization protocols used in the *Baseline* scenario.

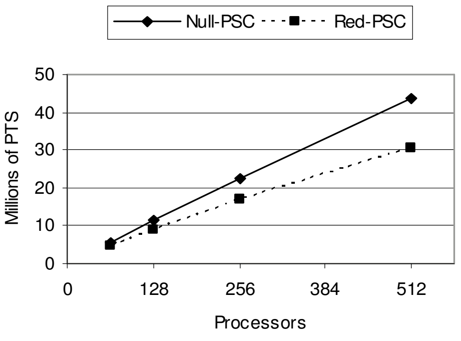

Figure 8 Large-Scale Simulation PTS Rate

Table 5. Large-Scale Run times for the *Baseline* Scenario

Number of CPUs	Null Messages	Reductions
64	420	508
128	414	523
256	420	563
512	436	620

Table 5 shows the runtimes of the *Baseline* scenario on the Compaq Alpha Tru64 cluster. The corresponding PTS rates are plotted in Figure 8. The execution time of the baseline scenario using global reductions increases somewhat as the model scales with the number of processors. In contrast, the null message algorithm performance exhibits a relatively constant execution time up to 512 processors. These experimental results provide strong support to verify the overhead model and proposition put forth in section 5.4.

The experimental results presented in this section have only investigated synchronization behavior of a structured, regular network (e.g. *Baseline* model). In the next section, we will compare the performance of the local and global synchronization algorithms for network models of both regular and irregular structure.

7. Regular and Irregular Network Models

Due to the asynchronous nature of the null message algorithm, synchronization occurs "locally" and remains constant even as the simulation model scales. This is the reason why CMB algorithms tend to scale better than schemes based on global reductions, if all other model factors remain constant. We will show that even with smaller scale network models, an optimized null message algorithm can provide improved performance for irregular networks.

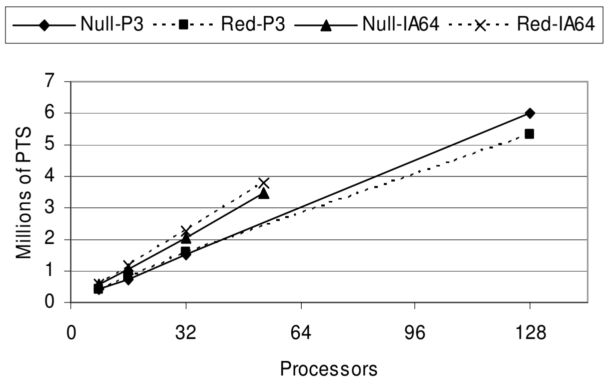

Figure 9. Performance of the *Baseline* Scenario

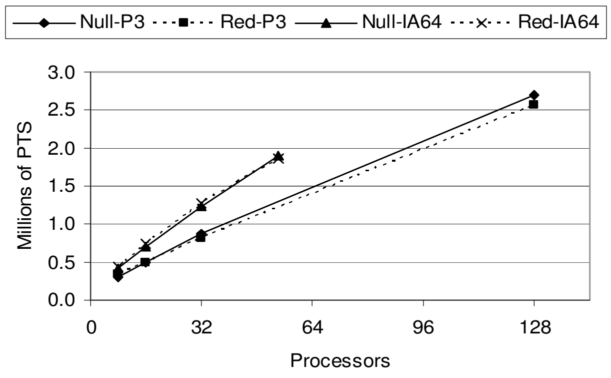

Figure 10. Performance of the *Baseline-R* Scenario

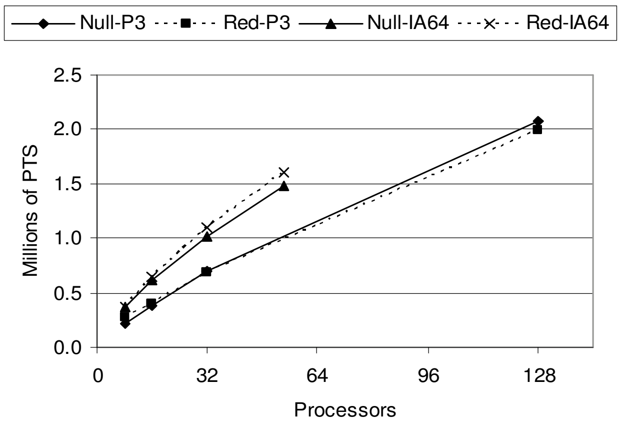

Figure 11. Performance of the *Chord-R* Scenario

7.1. Simulation Performance of Structured, Regular Networks

First we examine the performance of local and global synchronization algorithms in the context of relatively regular network models. The performance of the *Baseline* model on both the Intel Pentium III and Itanium 2 cluster is shown in Figure 9 as was discussed in the previous section.

The *Baseline-R* scenario is another network considered as a regular model with random server selection. Figure 10 illustrates that the trends of the *Baseline-R* scenario do not differ significantly from that of the *Baseline* model. The random Net 1 server selection pushes the performance of the asynchronous null message algorithm ahead of the global reduction algorithm at 16 CPUs for the P3 data and at 54 CPUs for the IA64 data.

The final regular network is the *Chord-R* scenario shown in figure 11. Although the addition of chords add asymmetry to the network, the structure and regularity of the network model is preserved by identical lookahead on all chord and ring links. Similar trends to the first two scenarios re-emerge in *Chord-R*.

These scalability runs thus far show that there exist only small gains in using one synchronization algorithm over the other in small- to medium-scale regular networks. However, previous large-scale simulation results suggest that the CMB algorithm with a constant number of output channels is more appropriate for large-scale simulations even in scenarios composed of nearly all consistent structure.

7.2. Simulation Performance of Asymmetric, Irregular Networks

The two irregular network scenarios, *Baseline-2ms* and *Chord-Asym* will highlight differences and deficiencies between local and global time management algorithms.

Figure 12 shows a significant difference in PTS rate between the null message algorithm and global reductions for the *Baseline-2ms* network model. Moreover, the null message trends remain linear while the global reduction algorithm yields a much smaller increase in PTS rate as the number of processors is increased. This scenario shows a pathological case for global reductions, as the time management system must perform a global synchronization every 2ms of simulation time.

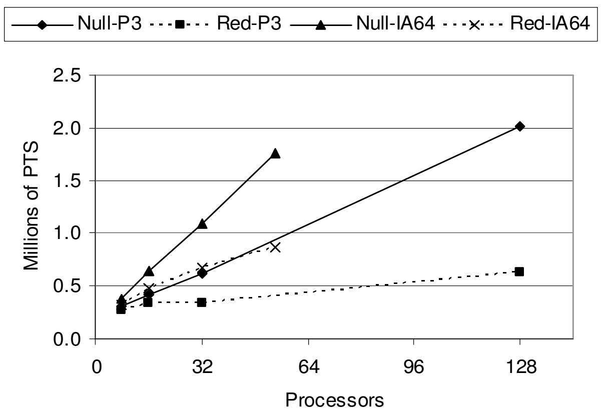

Figure 12. Performance of the *Baseline-2ms* Scenario

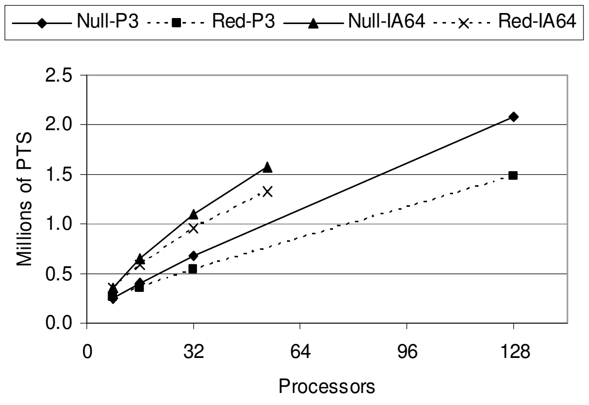

Figure13. Performance of the *Chord-Asym* Scenario

The *Chord-Asym* scenario shown in Figure 13 demonstrates the benefits of a null message algorithm in a typical irregular network. A null message algorithm can take advantage of the varying lookahead values for each output channel while a global algorithm must interrupt execution of the simulation at intervals dictated by the global minimum lookahead.

8. Large-Scale Conservative Synchronization Summary

We presented an analytical model for an asynchronous lazy null message algorithm for synchronization. The proposed equations can be used to approximate null message activity and a corresponding overhead index. In particular, the overhead index predicts that the optimized null message algorithm ensures no appreciable increase in overhead if the fan-in/fan-out of channels is held constant as the simulation is scaled with the number of processors.

We also showed that a null message algorithm offers more flexibility for irregular and asymmetric network models. An optimized CMB algorithm can tailor synchronization messages according to output channel lookahead values to local neighbors only, in contrast to the approach using reduction computations in a global synchronization algorithm. These properties prove to be advantageous in large-scale network simulations.

9. Future Research

The goal of this paper is to try to characterize quantitatively the capability of parallel simulation tools to simulate large-scale networks, and to highlight that the ability now exists to simulate large networks. This is by no means to imply scalable network simulation is a "solved problem"! Much additional research and development is required to effectively exploit these capabilities.

On the modeling side, creating realistic models of the Internet remains an extremely challenging problem. Some of the key issues that must be addressed are described in [52]. For example, the topology, configuration, and traffic of the Internet *today* is not well understood and is constantly changing, let alone the Internet of tomorrow that is targeted by most simulation studies. Methodologies to effectively validate and experiment with large-scale network simulations must be developed. Much work is required to make the parallel simulation tools easily accessible and usable by network researchers who are not experts in parallel processing. Robust parallel performance is needed; modest changes to the configuration of the network being simulated may result in severe performance degradations. These and many other issues must be addressed before large-scale network simulation tools can reach their fullest potential.

Acknowledgments

This research was supported under DARPA contract N66001-00-1-8934 and NSF grants ANI-0136939 and ANI-9977544. Large-scale computations designated as "PSC" were performed on the National Science

Foundation Terascale Computing System at the Pittsburgh Supercomputing Center. The assistance of the staff at the Pittsburgh Supercomputing Center is also gratefully acknowledged.

References

1. Breslau, L., et al., *Advances in Network Simulation.* IEEE Computer, 2000. **33**(5): p. 59-67.
2. Zeng, X., R. Bagrodia, and M. Gerla, *GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks*, in *Proceedings of the 1998 Workshop on Parallel and Distributed Simulation.* 1998. p. 154-161.
3. Chang, X., *Network Simulations with OPNET*, in *Proceedings of the 1999 Winter Simulation Conference*. 1999.
4. Riley, G.F., R.M. Fujimoto, and M.A. Ammar, *Stateless Routing in Network Simulations*, in *Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 2000.
5. Lin, Y.-B. and E.D. Lazowska, *A Time-Division algorithm for Parallel Simulation.* ACM Transactions on Modeling and Computer Simulation, 1991. **1**(1): p. 73-83.
6. Andradottir, S. and T. Ott, *Time-Segmentation Parallel Simulation of Networks of Queues with Loss or Communication Blocking.* ACM Transactions on Modeling and Computer Simulation, 1995. **5**(4): p. 269-305.
7. Greenberg, A.G., B.D. Lubachevsky, and I. Mitrani, *Algorithms for Unboundedly Parallel Simulations.* ACM Transactions on Computer Systems, 1991. **9**(3): p. 201-221.
8. Wu, H., R.M. Fujimoto, and M. Ammar, *Time-Parallel Trace-Driven Simulation of CSMA/CD*, in *Proceedings of the Workshop on Parallel and Distributed Simulation*. 2003.
9. Jones, K.G. and S.R. Das, *Time-Parallel Algorithms for Simulation of Multiple Access Protocols*, in *Ninth International Symposium on modeling, Analysis, and Simulation of Computer and Telecommunication Systems.* 2001.
10. Greenberg, A., et al., *Efficient Massively Parallel Simulation of Dynamic Channel Assignment Schemes for Wireless Cellular Communications*, in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation.* 1994. p. 187-194.
11. Fujimoto, R.M., I. Nikolaidis, and A.C. Cooper, *Parallel Simulation of Statistical Multiplexers.* Journal of Discrete Event Dynamic Systems, 1995. **5**: p. 115-140.
12. Szymanski, B.K., Y. Liu, and R. Gupta, *Parallel Network Simulation under Distributed Genesis*, in *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*. 2003. p. 61-68.
13. Perumalla, K., R. Fujimoto, and A. Ogielski, *TeD - A Language for Modeling Telecommunications Networks.* Performance Evaluation Review, 1998. **25**(4).
14. Poplawski, A.L. and D.M. Nicol, *Nops: A Conservative Parallel Simulation Engine for TeD*, in *12th Workshop on Parallel and Distributed Simulation.* 1998. p. 180-187.
15. Cowie, J.H., D.M. Nicol, and A.T. Ogielski, *Modeling the Global Internet.* Computing in Science and Engineering, 1999.
16. Liu, J. and D.M. Nicol, *DaSSF 3.0 User's Manual*. 2001.
17. Unger, B., *The Telecom Framework: a Simulation Environment for Telecommunications*, in *Proceedings of the 1993 Winter Simulation Conference*. 1993.
18. Pham, C.D., H. Brunst, and S. Fdida, *Conservative Simulation of Load-Balanced Routing in a Large ATM Network Model*, in *Proceedings of the 12th Workshop on Parallel and Distributed Simulation.* 1998. p. 142-149.
19. Miller, D.C. and J.A. Thorpe, *SIMNET: The Advent of Simulator Networking.* Proceedings of the IEEE, 1995. **83**(8): p. 1114-1123.
20. IEEE Std 1278.1-1995, *IEEE Standard for Distributed Interactive Simulation -- Application Protocols.* 1995, New York, NY: Institute of Electrical and Electronics Engineers, Inc.
21. IEEE Std 1278.2-1995, *IEEE Standard for Distributed Interactive Simulation -- Communication Services and Profiles.* 1995, New York, NY: Institute of Electrical and Electronics Engineers Inc.
22. Kuhl, F., R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture for Simulation.* 1999: Prentice Hall.
23. IEEE Std 1516.3-2000, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) -- Interface Specification*. 2000, New York, NY: Institute of Electrical and Electronics Engineers, Inc.

24. Nicol, D. and P. Heidelberger, *Parallel Execution for Serial Simulators.* ACM Transactions on Modeling and Computer Simulation, 1996. **6**(3): p. 210-242.
25. Perumalla, K., et al., *Experiences Applying Parallel and Interoperable Network Simulation Techniques in On-Line Simulations of Military Networks*, in *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*. 2002. p. 97-104.
26. Perumalla, K.S., et al., *Scalable RTI-Based Parallel Simulation of Networks*, in *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*. 2003. p. 97-104.
27. Fujimoto, R.M., T. McLean, and K.S. Perumalla. *Design of High Performance RTI Software*. in *the 4th Workshop on Distributed Simulation and Real-Time Applications*. 2000. San Francisco, CA.
28. Fujimoto, R.M., *Parallel and Distributed Simulation Systems.* Wiley Series on Parallel and Distributed Computing, ed. A.Y. Zomaya. 2000, New York: Wiley-Interscience. 320.
29. Mattern, F., *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation.* Journal of Parallel and Distributed Computing, 1993. **18**(4): p. 423-434.
30. Riley, G., R.M. Fujimoto, and M. Ammar, *A Generic Framework for Parallelization of Network Simulations*, in *Proceedings of the Seventh International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 1999. p. 128-135.
31. Nicol, D.M., *The Cost of Conservative Synchronization in Parallel Discrete Event Simulations.* Journal of the ACM, 1993. **40**(2): p. 304-333.
32. Chandy, K.M. and J. Misra, *Asynchronous Distributed Simulation via a Sequence of Parallel Computations.* Communications of the ACM, 1981. **24**(4): p. 198-205.
33. Lubachevsky, B.D., *Efficient Distributed Event-Drive Simulations of Multiple-Loop Networks.* Communications of the ACM, 1989. **32**(1): p. 111-123.
34. Chandy, K.M. and J. Misra, *Distributed Simulation: A Case Study in Design and Verification of Distributed Programs.* IEEE Transactions on Software Engineering, 1979. **5**(5): p. 440-452.
35. Bryant, R.E., *Simulation of Packet Communications Architecture Computer Systems*. 1977, Massachusetts Institute of Technology. MIT-LCS-TR-188.
36. Bagrodia, R. and M. Takai, *Performance evaluation of conservative algorithms in parallel simulation languages.* IEEE Transactions on Parallel and Distributed Systems, 2000. **11**(4): p. 395-411.
37. Su, W.-K. and C.L. Seitz. *Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm.* in *the SCS Multiconference on Distributed Simulation '89*. 1989. Miami, FL.
38. Porras, J., et al. *Improving the Performance of the Chandy-Misra Parallel Simulation Algorithm in a Distributed Workstation Environment*. in *Summer Computer Simulation Conference*. 1997. Arlington, VA.
39. Cai, W. and S.J. Turner, *An Algorithm for Reducing Null-Messages of the CMB Approach in Parallel Discrete Event Simulation*. 1995, University of Exeter.
40. Xiao, Z., et al. *Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation.* in *the 13th Workshop on Parallel and Distributed Simulation*. 1999. Atlanta, GA.
41. Song, H.Y., R.A. Meyer, and R. Bagrodia. *An Empricial Study of Conservative Scheduling*. in *the 14th Workshop on Parallel and Distributed Simulation*. 2000. Bologna, Italy.
42. Bailey, M.L. and M.A. Pagels. *Measuring the Overhead in Conservative Parallel Simulations of Multicomputer Programs*. in *the 23rd Winter Simulation Conference*. 1991. Phoenix, AZ.
43. Nicol, D.M. *Scalability, Locality, Partitioning and Synchronization in PDES*. in *the 12th Workshop on Parallel and Distributed Simulation*. 1998. Banff, Alberta, Canada.
44. Nicol, D.M. and J. Liu, *Composite Synchronization in Parallel Discrete-Event Simulation.* IEEE Transactions on Parallel and Distributed Systems, 2002. **13**(5): p. 433-446.
45. Nicol, D.M. *Analysis of Synchronization in Massively Parallel Discrete-Event Simulation*. in *the 2nd ACM SIGPLAN*. 1990. Seattle, WA.
46. Nicol, D.M., C. Michael, and P. Inouye. *Efficient Aggregation of Multiple LP's in Distributed Memory Parallel Simulations*. in *the 21st Winter Simulation Conference*. 1989. Washington, D.C.
47. Lim, C.-C., et al. *Performance Prediction Tools for Parallel Discrete-Event Simulation.* in *the 13th Workshop on Parallel and Distributed Simulation*. 1999. Atlanta, GA.
48. Naroska, E. and U. Schwiegelshohn, *Conservative Parallel Simulation of a Large Number of Processes.* Simulation, 1999. **72**(3): p. 150-162.
49. Lemeire, J. and E. Dirkx. *Performance Factors in Parallel Discrete Event Simulation*. in *the 15th European Simulation Multiconference*. 2001. Prague.

50. Jha, V. and R. Bagrodia. *A Performance Evaluation Methodology for Parallel Simulation Protocols*. in *the 10th Workshop on Parallel and Distributed Simulation*. 1996. Philadelphia, PA.
51. *NMS Baseline Model. http://www.cs.dartmouth.edu/~nicol/NMS/baseline/*.
52. Floyd, S. and V. Paxson, *Difficulties in Simulating the Internet.* IEEE/ACM Transactions on Networking, 2001. **9**(4): p. 392-403.