Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

Performance of Conservative Synchronization Methods for Complex Interconnected Campus Networks in ns-3

Brian Paul Swenson
Jared S. Ivey
George F. Riley

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, USA

ABSTRACT

Distributed simulations provide a powerful framework for utilizing a greater amount of computing resources, but require that consideration be taken to ensure that the results of these simulations match results that would have been produced by a single sequential resource. Also, synchronization among the multiple nodes must also occur to ensure that events processed in the simulation maintain a timestamped order across all nodes. This work uses the popular network simulator ns-3. The ns-3 simulator is a discrete event network simulator used for educational and research-oriented purposes which provides two implementation options for distributed simulations based on the null message and granted time window algorithms. We examine the performance of both distributed schedulers available in ns-3 in an effort to gauge specific features of an overall distributed network topology that warrant the use of one synchronization method over the other.

1 INTRODUCTION

Large-scale network simulations, consisting of vast, complex network topologies and multiple competing sources of traffic, are often necessary to gain a full understanding of the impact of new internet protocols or applications. As the size of the network to be simulated grows, so do its processing and memory requirements. To avoid the hindrances of lengthy execution times or limited memory on a single computing resource, simulations may instead be distributed over multiple processing nodes. Although distributed simulations provide a powerful framework for utilizing a greater amount of computing resources, the nature of executing these simulations across multiple nodes demands that consideration be taken to ensure that the results of these distributed simulations match results that would have been produced by a single sequential resource. Additionally, synchronization among the multiple nodes must also occur in an efficient manner to ensure that events processed in the simulation maintain a timestamped order across all nodes.

Synchronization methods for distributed simulations may take one of two forms, optimistic or conservative. Optimistic synchronization allows a distributed simulation to process events without immediate concern for timestamp order but will periodically check for out-of-order events and remedy them implementing rollback and anti-messages. Conservative synchronization guarantees that events are processed in timestamp order by having each processing node within a distributed simulation communicate to other nodes the earliest possible timestamp they will receive from a remote event. This synchronization method may further be implemented in a number of ways. This paper examines two implementations: null messages (also referred to as the Chandy/Misra/Bryant algorithm) and global consensus based on an agreed-upon granted time window.

This work uses the popular network simulator ns-3. The ns-3 simulator is a discrete event network simulator used for a variety of educational and research-oriented purposes within the computer network communications realm. It provides an open-source, modular framework for creating simulated network topologies using C++ and/or Python and installing stock or user-created applications on nodes of these topologies in order to simulate transmission and reception of data within a network. From these simulations, users of ns-3 can glean simulated performance metrics from the collected data in order to better understand and gauge the performance of existing and proposed real-world networks.

Currently, ns-3 provides implementations for distributed simulations based on the aforementioned conservative synchronization methods. In some cases, the choice of which synchronization method to consider for a distributed network simulation falls primarily to the preference of the user. While previous studies have compared conservative synchronization algorithms in other network simulators, such as pdns (Park, Fujimoto, and Perumalla 2004), the recent addition of the null message algorithm to ns-3 requires a thorough performance analysis of the options available to a user of ns-3 to aid selection of a particular synchronization method. This paper examines the performance of both distributed schedulers available in ns-3 in an effort to gauge specific features of an overall distributed network topology that warrant the use of one synchronization method over the other.

The remainder of the paper is organized as follows. In Section 2 background is provided on parallel discrete event simulation as well as time synchronization algorithms. In Section 3 the topology used for the simulation is described as well parameters used for the experiments. In Section 4 the results are presented. Finally in Section 5 future work is discussed and the work is concluded.

2 BACKGROUND INFORMATION

In this section we provide background information on parallel discrete event simulation and time synchronization algorithms as well as the two conservative time synchronization algorithms provided by ns-3.

2.1 Parallel Discrete Event Simulation

As of version 3.8, ns-3 provides a framework for performing distributed simulations (Pelkey and Riley 2011). It follows the federated (Riley, Ammar, Fujimoto, Park, Perumalla, and Xu 2004) approach to distributed network simulation by breaking the network topology into groups of nodes and placing each group into a logical process (LP)(Fujimoto 2000). Within an ns-3 distributed simulation, network nodes are created with a rank value signifying on which LP they will reside. An entire network topology will be created on each LP in a distributed simulation; however, at the application level, only those network nodes possessing the rank of a particular LP will be processing events on that LP. Each LP has its own local simulation time and its own event list which will only contain the events that are to be processed on that particular LP. An LP will communicate an event impacting another LP to that particular LP through the transmission of a time-stamped message. It is mandatory that a distributed simulation produce the same results as a corresponding sequential simulation; therefore, causality must be enforced among the LPs. Specifically, an LP must be prevented from advancing too far ahead of other LPs such that it receives an incoming event from another LP that has a timestamp less than the current simulation time of the receiving LP. If an LP attempts to process an event with a timestamp that is less than its current simulation time, a causality error will occurr. To enforce the property of causality among LPs, distributed simulators make use of synchronization algorithms. These algorithms can be classified into two groups: optimistic and conservative.

Optimistic synchronization allows LPs to progress independently of one another for short periods of time, which can result in possible causality violations. If an LP processes an event out of order, i.e. it later receives an event with an earlier timestamp from another LP, the LP must undo, or rollback, its local state to its state prior to processing the out of order event. The newly received event with the earlier timestamp can then be processed, and the LP will resume processing events in its event list, possibly reprocessing events that were rolled back if they are still valid given the update to the state of the LP by the new message.

The Time Warp mechanism (Jefferson 1985) is the most commonly implemented optimistic time synchronization algorithm. In the original proposal, the LP saves a copy of its entire state prior to the processing of each event. Then if the LP needs to rollback, it discards its local state and reloads the saved state prior to the processing of the out-of-order event. In order to minimize the amount of state storage for each LP, a global virtual time (GVT) is maintained within the system. The GVT is a timestamp which contains the earliest timestamp of all of the unprocessed events in the system. Since no event with a timestamp earlier than the GVT will ever appear in the event list of a particular LP to cause a rollback, any state saved prior to the GVT can be removed. This procedure of reclaiming memory is referred to as fossil collection (Fujimoto 2000).

More recently, simulations using Time Warp have made use of reverse computation (Perumalla 2014). In reverse computation, the LP performs the inverse of events that have been processed out of order, effectively allowing the LP to run backwards in time to the point where it has processed an out-of-order event. However, for this process to succeed, reverse computation code must be written for each event type. Attempts have been made to create a compiler that automatically generates the reverse code for events (Vulov, Hou, Vuduc, Fujimoto, Quinlan, and Jefferson 2011); however, frequently it must be done manually.

The distributed ns-3 framework currently only supports conservative time synchronization algorithms. With conservative time synchronization algorithms, each LP has to determine when an event is safe to process. An event is safe only when it can be guaranteed that the LP will not receive events from other LPs with timestamps less than the event being considered. In this way, the LP guarantees that all of its events are processed in order. To determine if an event is safe, the LP finds the lowest bound timestamp (LBTS) on all possible events that it may receive in the future. How this bound is determined is dependent on the particular algorithm used. As of ns-3 version 3.19, the simulator supports two conservative time synchronization algorithms, the Granted Time Window algorithm and the Null Message algorithm.

2.2 The Granted Time Window Algorithm

The Granted Time Window algorithm in ns-3 uses the Distributed Snapshot algorithm proposed by Mattern (Mattern 1993). This algorithm provides a global synchronization method to obtain minimum timestamp values by examining a consistent cut or snapshot of the system. The basic idea of this approach is to globally examine all of the event queues in the system and find the event with the smallest timestamp. However, the process is complicated by the presence of transient messages within the system. Transient messages are messages that have been sent by one LP but have not yet been processed by the receiving LP. It is possible that the event generated by a transient message will contain a timestamp that is less than any of the events already contained in the event list of the LP. Therefore, all transient messages must be handled before the LBTS calculation can take place. Using the LBTS values, an LP can determine a safe, or granted time, window within which the LP may process events in its event list and be guaranteed not to receive an event with an earlier timestamp from another LP.

The ns-3 granted time window algorithm and its accompanying MPI interface enlist a system for LPs to determine a granted time window for which they may safely process events in simulation timestamp order by achieving a global consensus among all LPs. During initialization the simulation begins by calculating the lookahead and initial granted time based on remote channel delays in the specified network topology. Distributed simulations in ns-3 currently only support remote point-to-point links between LPs, and as such, lookahead values are only derived from channels configured in this way. At the start of simulation, each LP examines the nodes within it to determine the shortest propagation delay between its nodes and nodes in any adjacent LP. This value becomes the lookahead for that particular LP. To maintain adequate performance, LPs that do not possess inter-task links are given lookahead and initial granted time values equivalent to the maximum lookahead from all LPs that do possess inter-task links in order to allow all tasks to advance in simulation time at similar rates. Once simulation execution begins, an LP will process the events in its event list until it reaches the end of its granted time window. When an LP has no more events within its granted time window, the synchronization phase will begin. The LBTS values will be gathered from all LPs using an MPI_Allgather call, effectively blocking individual LPs until all LPs reach this point in execution of the code. A check that the total count of completed received and completed transmitted messages are equal is executed to ensure that transient messages are not awaiting delivery by MPI. If transient messages exist in the system, each LP enters a phase where it stops sending traffic and completes all of its reads. Subsequently, another call to MPI_AllGather is made to verify that all transient messages are out of the system and gather the LBTS for the system. The minimum of these values will be summed with the lookahead to determine the current granted simulation time within which an LP may process its events. Multiple calls to MPI_AllGather are sometimes necessary to ensure that the system can obtain an accurate snapshot and to obtain the correct LBTS values. Following this synchronization process, if the next event time lies within the granted time window, the next event will be processed, and the simulation will continue.

2.3 The Null Message Algorithm

For ns-3.19, an additional distribution implementation was incorporated which provides a null message synchronization option following the Chandy/Misra/Bryant (CMB) algorithm (Chandy and Misra 1979) (Bryant 1977). Unlike the granted time window algorithm, which is considered a synchronous algorithm due to its use of global communication, the CMB algorithm is asynchronous, only requiring peer-to-peer communication between neighboring LPs to preserve global causality.

Similarly to the granted time window implementation, an LP participating in the null message synchronization simulation begins by scanning the nodes within its topology to find external links to remote LPs. It groups all of the links to remote LPs into bundles according to which LP it connects. It then determines the minimum propagation delay value for each bundle. This value becomes the lookahead between the two LPs. Prior to the start of simulation, the LP queues a null message to the remote LP on each bundle with a timestamp of the lookahead value. The implementation also schedules a future null message to be sent to the remote LP when the sender's local simulation time reaches the minimum propagation delay to the remote LP. When this null message is sent, assuming no transfer of packets between the two LPs has occurred, the sender will continue to send null messages to the remote LP every lookahead period. When a packet is transferred between the two LPs, the next pending null message to be sent is canceled. Following the transfer of the packet, a new null message is scheduled to be sent to the remote LP after the next lookahead period.

An important note regarding the ns-3 implementation of the null message algorithm relates to its handling of stopping the simulation. In ns-3.19, the simulation will only stop under the null message synchronization for a specified simulation stop time. Otherwise, the simulation will continue to run, simply passing null messages between LPs, until the simulator reaches the ns-3 definition of infinity. This drawback can be a hindrance when using the ns-3 null message synchronization because a user must either specify a stop time or allow the simulation to reach infinity, a task which can be significantly time-consuming. This issue is not present in the ns-3 implementation of the granted time window algorithm as LPs will reach a consensus that the simulation is globally complete when all LPs have empty event queues.

3 EXPERIMENTS

The performance study demonstrated in this paper examines the runtime and memory usage of multiple instances of the well known DARPA campus network topology (Nicol D. M. 2002) shown in Figure 1. These are all connected in a ring topology as shown in Figure 2. Simulations are implemented using a modified version of nms-p2p-nix-distributed.cc from ns-3.19. A single campus network consists of 3 networks connected to a central network Net 0, which serves as the entry and exit point of data travelling between campus networks. Connecting each campus network at Router 0 in Net 0 involves a 2Gbps point-to-point link with a 200ms propagation delay. Within each campus network, routing nodes are connected to one another via 1Gbps point-to-point links with 5ms delays. The components labeled LAN in Figure 1 are actually 42-node local area networks (LAN) with each node individually connected to its respective router via 100Mbps point-to-point links with 1ms delays. In total, each campus network is composed of 538 nodes.

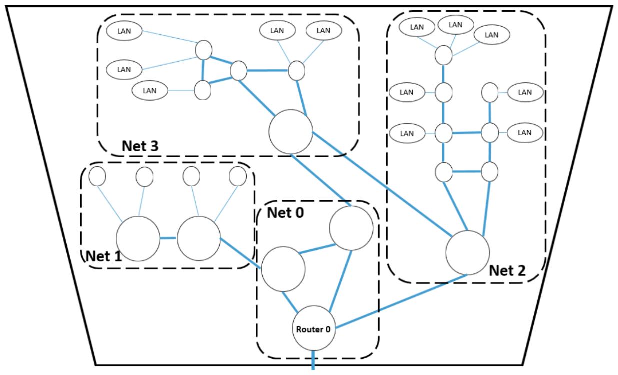

Figure 1: The DARPA campus network topology is composed of three networks which connect to Net 0, the entry and exit point for the campus network. Net 1 consists of 2 routers and four ”server” nodes while Nets 2 and 3 connect a number of routers to multiple LAN networks consisting of 42 nodes each. Routers and server nodes are connected via 1Gbps point-to-point links with 5ms delays. LAN nodes connect to their associated routers with 100Mbps point-to-point links with 1ms delays. Data flowing into and out of the campus network travels a 2Gbps point-to-point channel with a 200 ms delay.

For these experiments, a modified version of the nms-p2p-nix-distributed.cc simulation script distributed with the ns-3 release has been used. The first modification implemented enabled the addition of multiple chords between members of the ring topology. A parameter named nConCon signifies the number of ”consecutive connections” that each campus network will employ to connect to multiple campus networks. In this way, nConCon equal to 1 represents the original topology and can be used as a basis for comparing results produced by increasing the nConCon value. An example showing multiple consecutive connections can be seen in Figure 2.

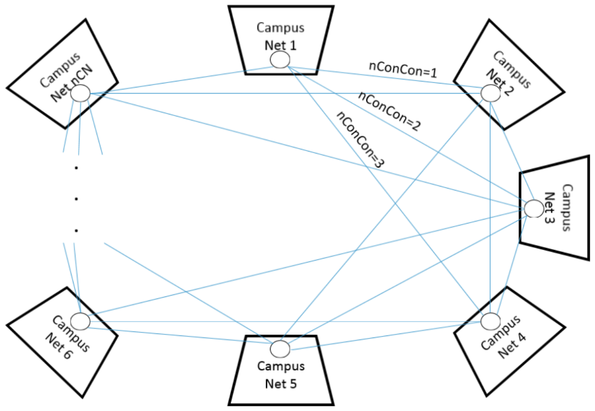

Figure 2: Multiple campus networks are connected in a ring topology with nConCon being the number of consecutive connections from one campus network to another campus network. For example, when nConCon=3, campus network 1 will connect to campus networks 2, 3, and 4, campus network 2 will connect to campus networks 3, 4, and 5, etc.

Each of the 42 nodes in each LAN in Nets 2 and 3 is given a UDP packet sink. For each packet sink added, an on-off application is installed on a randomly selected server node in Net 1. Each server node sends data to one of the sinks in the next campus on the ring. These applications oscillate between configurable periods of on time, during which data is sent, and off time, during which no data transmission occurs. Furthermore, for each additional consecutive connection for an LP, an additional on-off application is created on a server node in Net 1 and configured to send data to the campus network on the other end of the connection. The duty cycle is configured such that only one on-off application from each campus network is transmitting data out of that campus network at a time with each application being granted an equal amount of time to transmit. This setup ensures that the traffic in and out of each campus network effectively remains the same while externally presenting the distributed simulation implementations and the distributed systems on which they execute with new choices and challenges concerning their synchronization.

One other parameter is considered in this experiment which enables control of how much data is being sent remotely for each LP. When creating each on-off application, a uniformly random number is generated. If the random number is less than the specified remote percentage, the application will send data to a remote campus network. If the random number is greater than the remote percentage, the application will send data to one of its local LANs.

For routing traffic between the nodes ns-3's nix-vector routing protocol was used. nix-vector(Riley, Ammar, and Zegura 2001) routing does an on demand BFS to find the shortest path from source to sink. It then stores the steering information to guide the packet along that path within the packet itself. The node that generates the nix-vector also caches it so future transfers are done without the need for another BFS.

4 RESULTS

All of the experiments performed for this work were initially run on a machine with dual hex-core Intel Xeon E5-2620s running at 2.0 GHz. The machine was configured with 64 GBs of RAM. The experiments were all run using ns-3.19 with optimized build settings.

For the first experiment, a topology of 24 campus networks was created, for a total of 12,912 ns-3 nodes. Each on-off application was configured to send .5 MB to its chosen packet sink. For this experiment all on-off applications were configured to send data to their local LAN. In this configuration the simulation is embarrassingly parallel as there is no cross LP traffic. By performing this experiment, the cost of synchronization among the LPs can be measured. The consecutive connection parameter was varied from 1 to 23. The corresponding run time for each configuration is shown in Figure 3. As expected for both synchronization types, as the number of consecutive connections is increased, the run time is increased due to the added number of on-off applications and packet traffic. However, the results show that the granted time window application scales better as the connectivity of the topology increases. This trend is expected because the number of null messages that the CMB algorithm needs to send increases with the connectivity of the graph. For the granted time window algorithm, the number of nodes participating in the MPI_Allgather is the same for each experiment. It should be pointed out that this type of situation is ideal for the granted time window algorithm. Due to the lack of transient messages, during synchronization it only needs to perform one MPI_Allgather, a rather expensive operation, in order to obtain an accurate snapshot for LBTS calculation.

For the second experiment, a topology of 8 campus networks was created, for a total of 4,304 ns-3 nodes. The number of consecutive connections was set to one making a standard ring topology. Each on-off application was configured to send a maximum of 50 kilobytes. In this experiment, the percentage of remote traffic each LP sends was varied. For the results presented here, each configuration was run 10 times, and the average result was recorded. The results are shown in Figure 4. As shown in the graph, as the percentage of remote data increases, the performance of the granted time window decreases. Part of this decrease is likely due to the increase in frequency of transient messages during the synchronization phase causing the need for a second MPI_Allgather. The run times for CMB simulations do increase, as expected since the number of packets that need to be serialized and sent is increasing; however, the increase is much less noticeable. One benefit for the CMB algorithm in this experiment is that as the number of packets being sent to remote LPs increases, the number of null messages it needs to send to those LPs decreases.

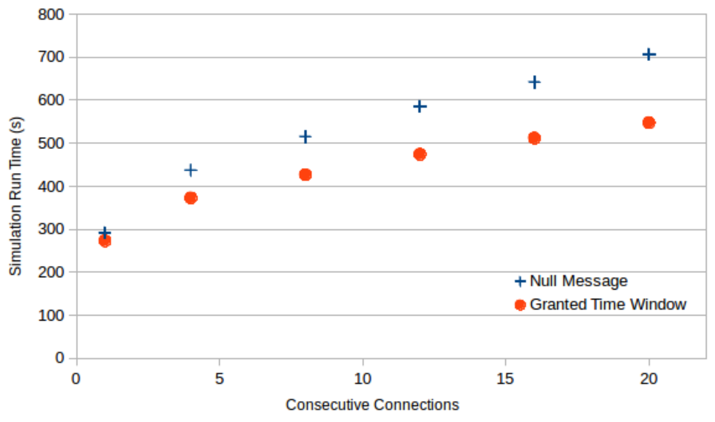

Figure 3: Simulation run time in seconds as a function of the number of consecutive connections. The total number of campus networks is 24 with each campus network residing on its own LP.

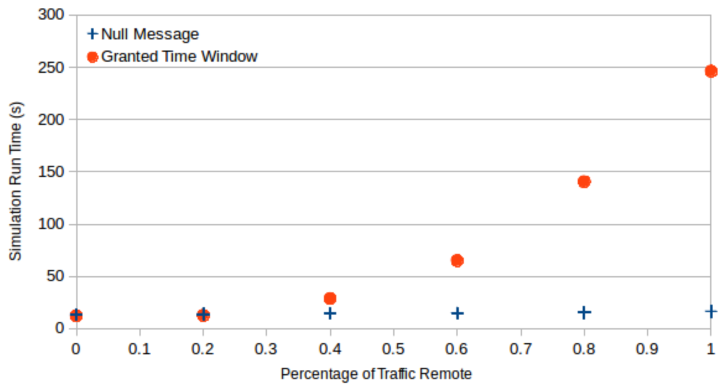

Figure 4: Simulation run time in seconds as a function of the percentage of remote traffic. The total number of campus networks is 8 with each campus network residing on its own LP and only one consecutive connection between campus networks (nConCon=1 from Figure 2).

Following the examination of the results of the second experiment, the significant increases in run time for the granted time window experiments were deemed unexpected; therefore, the experiment was repeated using a dedicated computing cluster rather than the original computing resource that employed shared memory. This computing cluster, managed by the Georgia Tech Partnership for an Advanced Computing Environment (PACE), provides dedicated cores as individual LPs. These cores are gathered for MPI job execution from a collective of six-core 2.4 GHz AMD Opteron 8431 processors. Ten time trials for each percentage of remote traffic were executed for both the null message and granted time window algorithms. As shown in Figure 5, a different picture is revealed as compared to that shown in Figure 4. The run times for the granted time window simulations were greatly improved.

The main difference between the two experiments is how MPI is utilized. For a many-core machine, as was used in the experiment depicted in Figure 4, MPI uses shared memory to pass data between the LPs. For the second cluster experiment, data sent between the LPs is passed across the network. The data obtained shows that the granted time window algorithm is performing poorly only when MPI is using

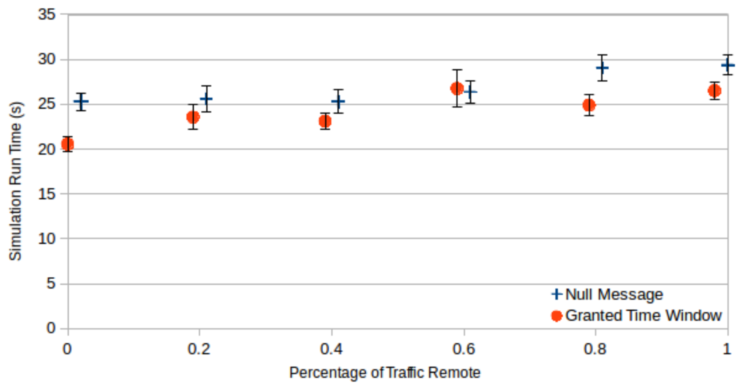

Figure 5: Simulation run time in seconds as a function of the percentage of remote traffic. This experiment utilized a cluster of computing resources rather than a shared memory resource. The total number of campus networks is 8 with each campus network residing on its own LP and only one consecutive connection between campus networks (nConCon=1 from Figure 2). Vertical bars represent the standard error of the sample mean.

shared memory. The shared memory experiment was repeated on another machine to verify that faulty operation was not inhibiting the first machine; similar results were obtained. All of these experiments were performed using OpenMPI 1.4.3.

In ns-3, the granted time window and null message algorithms check for received MPI messages at different frequencies. The null message algorithm checks for received messages every time it processes an event. The granted time window algorithm only checks for incoming messages once it determines it has no more safe events to process. It was hypothesized that the longer run-times on the shared memory machine were due to excessive buildup of unprocessed MPI messages. For the next experiment, the granted time window algorithm was changed to check for new messages after processing each event. The same experiment as the last two was performed on the many-core shared memory machine. The time trials were again repeated 10 times across the percentages of remote traffic to better gauge the confidence of the results. The results are shown in Figure 6.

As shown by Figure 6, the results of the granted time window algorithm on a many-core shared memory computer were greatly improved by increasing the frequency with which each LP checks for received messages. These results support the hypothesis that the delay was due to excessive buildup of MPI messages. As Figure 6 shows, after the modification, the performance for both synchronization algorithms is effectively identical for this configuration. The granted time window algorithm experienced a 15.6x speedup compared to the original run-times measured on the simulations with exclusively remote traffic.

The next experiment performed examined whether increasing the frequency with which the granted time window algorithm checked for incoming messages had any negative consequences on a larger network topology run on a computer cluster. In this experiment 32 LPs were created for a total of 17,216 nodes. Each on-off application was configured to send 50 kilobytes to its selected target. All targets were in remote LPs. The number of consecutive connections was varied from 5 to 30. Again each configuration was run 10 times. The results are shown in Figure 7. The results indicate for the configurations that were tested, no performance penalty was incurred by increasing the polling frequency for new messages. In fact for all of the connection configurations, a small performance increase in the modified granted time window implementation was observed compared to the original implementation.

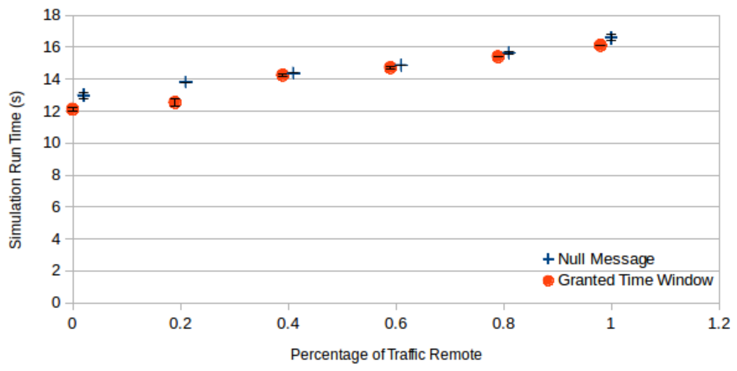

Figure 6: Simulation run time in seconds as a function of the percentage of remote traffic. This experiment utilized a shared memory resource but implemented a fix that checked for received MPI messages more frequently. The total number of campus networks is 8 with each campus network residing on its own LP and only one consecutive connection between campus networks (nConCon=1 from Figure 2). Vertical bars represent the standard error of the sample mean.

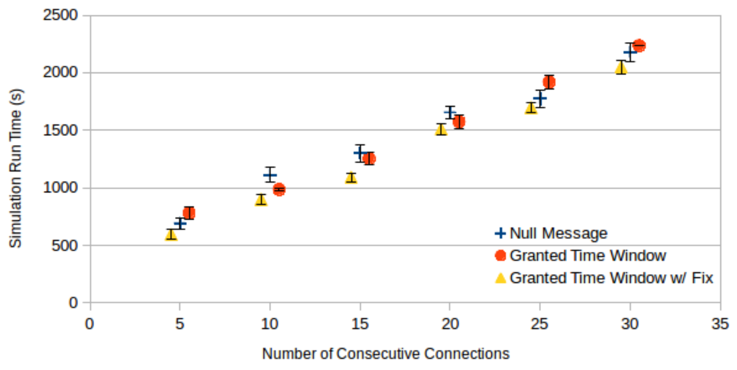

Figure 7: Simulation run time in seconds as a function of the number of consecutive connections. This experiment utilized a cluster of computing resources rather than a shared memory resource. The total number of campus networks is 32 with each campus network residing on its own LP. Vertical bars represent the standard error of the sample mean.

5 CONCLUSION AND FUTURE WORK

In this work we examined and compared the two conservative time synchronization algorithms available in ns-3.19. We performed experiments varying network connectivity and remote traffic percentage. The most interesting result was how poorly the granted time algorithm performed on the many-core machine using OpenMPI 1.4.3. Given these results, ns-3 users using this type of machine configuration would be best served by using the new null message algorithm or modifying the granted time window algorithm as we did.

The null message algorithm possesses drawbacks as well. As mentioned earlier, the algorithm is dependent on the simulation user to set an appropriate stop time. This requirement does not exist for the granted time window algorithm since it automatically exits once the simulation ends. In simulations with a fixed amount of data needing to be sent, overestimating the stop time could result in the simulator spending an excessive amount of time just passing null messages and not doing any actual simulation. Forgetting to set a stop time will result in a simulation that will run until the time variable overflows.

In the future we plan to continue testing the two algorithms with a variety of different topologies. Specifically, we plan to utilize the ns-3 interface to the BRITE topology generator (Medina, Lakhina, Matta, and Byers 2001)(Swenson and Riley 2013) which will allow us to generate highly customizable, large scale network topologies.

REFERENCES

Bryant, R. E. 1977. “Simulation of Packet Communication Architecture Computer Systems”. Technical report, Cambridge, MA, USA.

Chandy, K., and J. Misra. 1979, Sept. “Distributed Simulation: A Case Study in Design and Verification of Distributed Programs”. *Software Engineering, IEEE Transactions on* SE-5 (5): 440–452.

Fujimoto, R. M. 2000. *Parallel and Distributed Simulation Systems*. Wiley.

Jefferson, D. R. 1985. “Virtual time”. *ACM Transactions on Programming Languages and Systems* 7:404–425.

Mattern, F. 1993, August. “Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation”. *J. Parallel Distrib. Comput.* 18 (4): 423–434.

Medina, A., A. Lakhina, I. Matta, and J. Byers. 2001. “BRITE: an approach to universal topology generation”. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, 346–353.

Nicol D. M. 2002. “Standard baseline NMS challenge topology”. Accessed April 7, 2014. http://ssfnet.org/Exchange/gallery/baseline/.

Park, A., R. Fujimoto, and K. Perumalla. 2004, May. “Conservative synchronization of large-scale network simulations”. In *Parallel and Distributed Simulation, 2004. PADS 2004. 18th Workshop on*, 153–161.

Pelkey, J., and G. Riley. 2011. “Distributed Simulation with MPI in Ns-3”. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools ’11, 410–414. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Perumalla, K. S. 2014. *Introduction to Reversible Computing*. CRC Press.

Riley, G., M. Ammar, and E. Zegura. 2001. “Efficient routing using NIx-Vectors”. In *High Performance Switching and Routing, 2001 IEEE Workshop on*, 390–395.

Riley, G. F., M. H. Ammar, R. M. Fujimoto, A. Park, K. Perumalla, and D. Xu. 2004, April. “A Federated Approach to Distributed Network Simulation”. *ACM Trans. Model. Comput. Simul.* 14 (2): 116–148.

Swenson, B. P., and G. F. Riley. 2013. “Simulating Large Topologies in Ns-3 Using BRITE and CUDA Driven Global Routing”. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools ’13, 159–166. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Vulov, G., C. Hou, R. Vuduc, R. Fujimoto, D. Quinlan, and D. Jefferson. 2011, Dec. "The Backstroke framework for source level reverse computation applied to parallel discrete event simulation". In *Simulation Conference (WSC), Proceedings of the 2011 Winter*, 2960–2974.

AUTHOR BIOGRAPHIES

BRIAN PAUL SWENSON is currently working on a Ph.D. in Electrical and Computing Engineering at the Georgia Institute of Technology. He received his B.S. in Computer Engineering and Computer Science from the University of Wisconsin-Madison and received an M.S. in Electrical and Computing Engineering from the Georgia Institute of Technology in 2013. His research interests include distributed discrete event simulation, high performance computing and computer networks. His email address is bpswenson@gatech.edu.

JARED S. IVEY is currently pursuing a Ph.D. in Electrical and Computer Engineering from the Georgia Institute of Technology, focusing primarily on large scale distributed systems and simulations. He received his B.S. in Biomedical Engineering from Georgia Tech in May 2009 and his M.S.E. in Software Engineering from Mercer University in May 2012. He is currently employed as an electronics engineer at Robins AFB, GA. His email address is j.ivey@gatech.edu.

GEORGE F. RILEY received his Ph.D. from the Georgia Tech College of Computing in August 2001, and joined the faculty of ECE at that time. Mr. Riley received a MSCS from Florida Tech in 1996, and a BSEE from University of Alabama in 1972. Prior to enrolling at Tech in 1996, Mr. Riley was president and CEO of Infoware, Inc. of Cocoa Beach Florida. From 1987 to 1996 Infoware provided software and system design services to the United States Air Force at Patrick Air Force Base, Florida. During that time, Infoware designed, implemented, and deployed numerous systems in support of the missile launch activities at Cape Canaveral Air Force Station, including a communications front-end processor for real-time data gathering and a real-time distributed flight safety display system. Concurrently, from 1984 to 2000, Mr. Riley was also vice-president and co-owner of CAM Systems Inc. of Atlanta Georgia. CAM systems developed, under Mr. Riley's direction, a suite of PC based software tools for residential property management. His email address is riley@ece.gatech.edu.