A Benchmark Model for Parallel ns3

Peter D. Barnes, Jr. , James M. Brase (brase1@llnl.gov),
Thomas W. Canales (tcanales@llnl.gov), Matthew M. Damante (mdamante@llnl.gov),
Matthew A. Horsley (horsley1@llnl.gov), David R. Jefferson (drjefferson@llnl.gov),
Ron A. Soltz (soltz@llnl.gov)
Lawrence Livermore National Laboratory
7000 East Ave.
Livermore CA 94550 USA
1-925-422-3384

pdbarnes@llnl.gov

ABSTRACT

ns3 is a simulation framework for computer networks, derived from a long line of serial simulators. Recently, ns3 incorporated a parallel, distributed scheduler, which enables distributed ns3 simulation for the first time. In this paper we discuss the current implementation and some of its limitations, with an eye to exploring potential improvements. In order to gauge progress, it is essential to have a meaningful performance metric and a suitable benchmark problem. Therefore we outline how to measure the simulation critical path and use that to construct a parallel performance metric. Second, we propose a scalable benchmark model, inspired by the global Internet.

Keywords

MPI, network simulation, ns-3, distributed, performance, high performance computing.

1. INTRODUCTION

ns3 is a simulation framework for computer networks, derived from a long line of serial simulators.[4] Recently, ns3 incorporated a parallel distributed scheduler, which enables distributed ns3 simulation for the first time.[6] The parallel scheduler implementation has a number of limitations, and there are alternative implementations that might provide better performance. In order to make meaningful evaluations of alternatives, a standard benchmark model is needed. We are interested specifically in very large models ($10^{4\text{-}9}$ nodes) with a large amount of available parallelism, and want to benchmark highly distributed implementations running on up to 10^4 computing cores. After reviewing the current parallel implementation, we propose a benchmark model including topology and traffic specification. At the same time, we propose additional instrumentation for ns3 to compute the critical path, in order to measure the degree of parallelism available in any network model..

2. CURRENT IMPLEMENTATION

The current parallel scheduler is a straightforward global conservative scheduler, using MPI for communication between ns3 processes, or *ranks*, to use the MPI terminology. The implementation allows the topology to be partitioned at simulated point-to-point (P2P) channels only; CSMA and wireless links may not cross rank boundaries. At the same time, each rank must have the full topology available. (See reference [6].)

The use of remote P2P links is handled by the PointToPointHelper, which detects that a P2P link crosses ranks, attaches a PointToPointRemoteChannel and an MpiReceiver object to the PointToPointNetDevice on each side of the link. When it comes time to transmit a packet, the packet is serialized over the MPI link along with the receive time and destination node index and device index. (The destination node here is the other end of the P2P link, not the ultimate packet destination.) On the receiving side, the same data is deserialized and used to schedule the receive event for the destination node.

At initialization, each rank independently walks the topology to determine the smallest cross-rank P2P channel delay; this becomes the scheduler look-ahead, *LA*.

When a rank has no more events within the *LA* time, it enters a synchronization phase. It is guaranteed that all ranks will eventually exhaust their executable events and enter the synchronization phase. During this phase, the rank receives any outstanding messages, which will all be for events beyond the *LA* time. All ranks exchange messages containing the number of transmitted and received packets, and the time stamp of the next available event, using *MPI_ALLGATHER*. To check for transient (not yet delivered) MPI messages, each rank computes the total number of received and transmitted events. If these are unequal, indicating undelivered messages still exist, then the synchronization phase restarts. Once all messages have been received (events received equals events transmitted) each rank computes the global minimum next event time stamp (lower bound time stamp, LBTS), and adds the look-ahead, to obtain the maximum time stamp which is safe to process, called the *granted time*. The scheduler then proceeds to process any events scheduled before the end of *granted time*.

In addition to the restrictions noted above, the current implementation only transmits the receive time, the node index and device index, and any packet data. Packet and byte tags are not transmitted, nor is packet metadata, which used to interpret the packet contents correctly.

2.1 Discussion

The current implementation should perform well when the smallest P2P link delay is large compared to the typical event interval. Under this condition many events can be processed before synchronization is required. In addition, the case when there exist long time gaps, *i.e.* discontinuous busy periods, should also be handled well. By passing the timestamp of the *next*

Wns3 2012, March 23, 2012, Desanzano, Italy

available event during the LBTS computation, the simulator naturally skips over long gaps between events.

Requiring that each rank contain the global topology makes the identification of remote nodes straight-forward: each rank assigns the same index to each node, so merely passing the index between ranks is sufficient.

This requirement is somewhat of a limitation for really big networks. Beyond simplifying the implementation, this requirement is driven by the existing routing algorithms. All current `ns3` automatic routing algorithms ultimately require access to the global topology. Obviously, global (GOD) routing does, since it pre-computes all possible routes. Nix-vector routing computes routes as needed, but uses the global topology to do so. The total memory requirement is reduced somewhat by only instantiating applications on local nodes, but the need to carry the global topology, and global routing tables (for GOD routing) at each rank will limit the ultimate scaling. Removing this bottleneck, however, will require implementation of a distributed shortest path algorithm, for example Δ-Stepping.[6]

3. DEVELOPMENT DIRECTIONS

There are a number of straightforward ways to extend the current implementation, primarily by adding support for packet and byte tags, and packet metadata. In fact, the issue of packet tags has already been raised in online discussions.[9]

More involved is to remove the requirement that every rank instantiate the global topology. A much more scalable solution would only instantiate the rank-local nodes, and perhaps ghost nodes for the endpoints of any cross-rank links emanating from the current rank.[6] This will require implementation of a distributed routing algorithm, as noted above. At the same time, there will have to be a method to identify the remote receiving node without knowing the remote rank topology, in order to create and schedule the receive event correctly. In the current implementation non-blocking receives are scheduled for all other ranks. When receiving messages, the source rank of the message is not used to determine which P2P link is involved. Rather, the destination node (global) index is transmitted as part of the MPI message. Instead of identifying the receiving node by global index, the MPI interface should track *which* rank sent the message, and with that one rank build a common picture of the shared P2P links and end nodes.

Putting aside the high-level issues in routing and distributed topology, and the medium level issues of efficiently serializing and transmitting packets and metadata between ranks, we should also look at the low-level implementation of the distributed scheduling algorithm. For example, the use of a global minimum *LA* and *MPI_ALLGATHER* for synchronization should become bottlenecks for very large models. As an alternative, we propose to study a locally synchronized scheduler. This would be built around MPI links only to neighboring ranks, that is, ranks that share P2P links with the current rank. We determine the rank-local *LA* as the minimum channel delay for the P2P links to our neighbors, not across the global topology. For each neighbor rank, we store the timestamp of the last received packet; this tells us that neighbor has advanced to at least the simulation time the packet transmission began. If the cross-rank P2P links are generally busy, we can update our knowledge of our neighbors' simulation time without having to block and exchange LBTS messages. To ensure that our neighbors know our time regularly, we pre-schedule a *NULL* message for some fraction, *f*, *e.g.* *f* = 80%, of the link delay for each off-rank P2P link. When we send a packet on the link, we reschedule the pending *NULL* message for *f LA* time in the future. As a further variation, instead of using a rank-local *LA* equal to the minimum off-rank P2P delay, we could use a link-local *LA* equal to the specific link delay.

4. CRITICAL PATH

In order to evaluate alternative implementations, we need accurate instrumentation to measure changes in performance. To this end we propose to add critical path tracking to `ns3`, using the method of Srinivasan and Reynolds.[8] Each node will track the number of packet events (sends and receives) it has processed, which is the current *path length* for that node. When sending a packet, the node will increment its own path length and add that as a tag to the packet. When receiving a packet, the receiving node will update its path length with the larger of its own path length or the sending node path length, then add one (for receipt of the packet). We see that the node path length is the total number of packet send and receive events that must occur earlier in execution of the model in order for this packet to arrive at this node. At the end of the simulation, the largest node path length is the *critical path* for the model.

A perfect parallel simulator would only need as much time as a serial simulator executing just the critical path. The ratio of total packet events to the critical path length is the *available parallelism* in the model. In running a parallel simulation the *speedup* is the serial wall clock time divided by the parallel wall clock time executing the same model. Performance of the parallel implementation is measured as the speedup divided by the available parallelism:

$$\pi = \frac{T_{Serial}}{T_{Parallel}} \times \frac{L_{CP}}{N_{Events}} \tag{1}$$

where π is the parallel efficiency, T_{Serial} and $T_{Parallel}$ are the serial and parallel wall clock time, respectively, L_{CP} is the length of the critical path, and N_{Events} is the total number of packet send/receive events in the simulation. Equivalently, serial running time divided by total packet send and receive events is the average execution time per event. This, multiplied by the critical path length, is the minimum expected parallel running time. Finally, the minimum expected parallel running time divided by actual parallel running time is the parallel performance or efficiency. We should expect this to be in the range (0,1].

5. BENCHMARK MODEL

We are interested specifically in very large models with a large amount of available parallelism. Since the current automatic routing algorithms are not distributed, we do not want routing protocols and implementations to become limiting factors. To that end, we draw inspiration from a toy model proposed by Riley and Ammar (R&A).[7] The essential features of the R&A model are these:

- The node count, 10^8, is dominated 100:1 by end hosts, not routers.
- End hosts originate all traffic.
- The router-router graph determines the average diameter of the topology, $\bar{d} \cong 10$.
- Consequently the number of simulation events is dominated by router-router packet transmissions, not packet events at end hosts.

R&A leave the router-router network unspecified, except to say that all routers have degree 4, suggesting a Erdös-Rényi $G(n,M)$-

type random graph, consisting of n nodes each with M links to random other nodes.

The key observation is that the model is dominated by packets transiting between routers, with the routing unspecified. The host nodes exist only as sources and sinks for the packets.

Therefore, we propose the following benchmark model:

1. n router nodes; n sets the problem size. There are no host nodes.
2. Routers are connected in a random Watts-Strogatz graph, with average degree k.
3. Routers send packets to their nearest neighbors. There is no routing; no packets are retransmitted from a receiving interface to a sending interface on the same router.
4. Link utilization is either fixed at $u = 10\%$,[5] or given by a (unspecified) distribution, or varied to control the number of events per LA time.
5. Link bandwidth is either fixed at $B = 400$ Mbps,[scaled from 7] or given by an (unspecified) distribution.
6. Link delays are either fixed at $\delta = 20$ ms,[9] or given by a (unspecified) distribution.
7. Packets are all $S = 5$ kbit (625 byte) in size.
8. Inter-packet intervals are exponentially distributed (yielding Poisson count behaviour).

Rather than $G(n,M)$ we choose to use a random Watts-Strogatz graph with $k \geq 4$ so that we almost surely obtain a single connected graph, with minimum router degree ≥ 2. The choice of n and k determine the average diameter from the relations

$$n = k^{\bar{d}} \quad \bar{d} = d - \frac{1}{k-1} \approx d \tag{2}$$

For $n = 10^6$, roughly the number of routers in the Internet,[2] and $k = 4$ we obtain average diameter $\bar{d} = 10$, which is not too far from the observed $\bar{d} = 16$.[3]

The choice of link bandwidth, delay, and packet size are motivated by average values seen in the Internet. The quantity of interest for benchmarking the simulator is the number of packet receive events per LA time. Given a packet size S in bits, and link bandwidth B, the time spent transmitting a packet is

$$t_{TX} = \frac{S}{B} \tag{3}$$

The utilization, u, is this time multiplied by the rate of packet transmission, R. The LA time will be comparable to the link delay, δ, and the number of events within the LA window is

$$\begin{aligned} r = \delta R &= \delta \frac{u}{t_{TX}} \\ &= u \frac{\delta B}{S} \end{aligned} \tag{4}$$

This is the delay-bandwidth product divided by the packet size, times the utilization. If $r \gg 1$, then packets are regularly transmitted on the link and *NULL* messages will never be needed. If $r < 1$, then quiescent periods longer than the LA should occur. By varying the utilization we can control which regime the model tests. Typical values for the Internet are $S \approx 5$ kbit; $B \approx 400$ Mbps; $\delta \approx 20$ ms, as referenced above. For $u \approx 10\%$ this gives $r \approx 160$ events per LA window.

The choice of exponential inter-packet intervals is arbitrary; a Pareto distribution with mean given by Eq (3) would be interesting.

For a model running for total simulated time T, we expect the critical path to be of order $TuBk/S$, and available parallelism of n.

Partitioning random Watts-Strogatz graphs is straightforward, with *METIS*, for example.[2] For large graphs ($n > 10^3$) partitioning with 1% of the nodes per rank results in rank degree $\sim n/10$, *i.e*, ranks communicate with 10% of all other ranks.[1] This partitioning avoids all-to-all communication. On the other hand, the partitioning results in caterpillar graphs on each MPI rank, with half the links pointing to other ranks. Therefore half of all packets will be transmitted between ranks over MPI, providing a good test of the parallel implementation.

6. EVALUATION

To be clear, we have not studies this benchmark model with the current parallel implementation, never mind modifications. Our next steps are to implement the model, and the critical path algorithm, in order to identify which of the improvements discussed will likely have the greatest impact, and should be tackled first.

7. CONCLUSION

In this paper we have outlined possible development directions for the parallel, distributed scheduler in `ns3`. To enable meaningful performance measurements, we propose to augment `ns3` with a critical path algorithm. We also propose a benchmark model inspired by the global Internet. This model is trivially scalable with a high degree of available parallelism, and requires no routing. The combination of critical path measurement and a highly parallel benchmark model will enable disciplined improvement of the parallel `ns3` implementation.

8. REFERENCES

[1] Barnes, P. D., Jr. *Partitioning Random Watts-Strogatz Graphs*. LLNL, Livermore, CA, 2011.

[2] CAIDA *Internet topology at router- and AS-levels, and the dual router+AS Internet topology generator*. 2011.

[3] Fei, A., et al. Measurements on delay and hop-count of the Internet. In *Proceedings of the EEE GLOBECOM'98 - Internet Mini-Conference* (1998).

[4] ns3 Collaboration *The ns-3 network simulator*. Washington, 2011.

[5] Odlyzko, A. M. The Internet and other networks: utilization rates and their implications. *Information Economics and Policy*, 12, 4 2000), 341-365.

[6] Pelkey, J. and Riley, G. F. Distributed simulation with MPI in ns-3. In *Proceedings of the Workshop on ns-3* (Barcelona, Spain, March 25, 2011, 2011). ACM Digital Library.

[7] Riley, G. F. and Ammar, M. H. *Simulating Large Networks – How Big is Big Enough*. Georgia Institute of Technology, Atlanta, GA, 2002.

[8] Srinivasan, S. and Reynolds, P. F., Jr. *On Critical Path Analysis of Parallel Discrete Event Simulations*. TR-93-29, 1993.

[9] Zhang, B., et al. Measurement-based analysis, modeling, and synthesis of the internet delay space. *IEEE/ACM Trans. Netw.*, 18, 1 2010), 229-242.