BencHMAP: Benchmark-Based, Hardware and Model-Aware Partitioning for Parallel and Distributed Network Simulation *

Donghua Xu (xu@cc.gatech.edu)
Mostafa Ammar (ammar@cc.gatech.edu)
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332 USA

.

Abstract

Computer simulation of large-scale and complex networks can be resource intensive. Several tools to parallelize and distribute the simulation to a number of different machines have been developed. One of the main challenges facing users of these tools is how to partition the simulation among the computing resources available. The focus of this paper is on the development of a framework and methodology (ultimately leading to a semi-automated tool) to partition network simulation. The main distinguishing feature of our approach is that the partitioning is performed in a manner that takes into account the specific distributed computation environment available as well as the specific details of the network model. To achieve this, we derive the relations between impact factors and the simulation performance from measurements of benchmark experiments. *We then apply the derived relations to the given network topology and workload model to construct a weighted graph which we then partition using a graph partitioning tool. Experiments on a 120k-node, 100k-stream network simulation show that the full application of this approach improves the performance of partitioned simulation significantly over other partitioning heuristics.*

1. Introduction

In networking research, simulation has become an important way to study network operation and performance characteristics under various conditions. A number of network simulation tools have appeared ([1, 2, 3]) to help researchers construct and simulate complex network scenarios. Computer simulation of networks, especially large-scale and complex networks, can be resource intensive. This has often made it difficult and sometimes infeasible to run large-scale network simulations on a single workstation.

There are two general approaches to address this limitation. One is to use certain techniques (such as aggregation/compression) to reduce the resource requirement of network simulation so that a larger-scale simulation can fit into the available memory of a single workstation, such as in [4, 5].

The other approach, which is the focus of this paper, is to parallelize and distribute the simulation to a number of different machines, so that each machine can share a portion of the workload of the original simulation model (such as in [6, 7]). This approach is especially useful when large computer clusters consisting of tens or hundreds of machines are readily available for running the network simulations. One of the main problems facing the users of a parallel and distributed simulation (PADS) is how to partition the simulation among the computing resources available. The focus of this paper is on the development of a framework and methodology (ultimately leading to a semi-automated tool) to partition network simulation. The goal of a partitioning heuristic in previous work as well as ours is to 1) find a feasible partitioning (i.e., one where each component can be allocated within the memory resources to one of the available machines) and 2) to minimize the running time of simulation.

The main distinguishing feature of our approach is that the partitioning is performed in a manner that takes into account the specific distributed computation environment available as well as the specific details of the network model including both the network topology and the traffic characteristics. To achieve this, we derive the relations between impact factors and the simulation performance from measurements of *benchmark experiments.* These benchmark experiments are designed to explore the performance of parts of the specific model to be partitioned and run on the same specific computing hardware available.

There has been extensive research on general computation and data partitioning for parallel and distributed execution, such as [8, 9, 10]. This work mostly deals with compiler-level optimization for parallel pro-

* This work is supported in part by NSF under contracts number ANI-9977544 and ANI-0136936

grams, trying to automatically parallelize general computation tasks by optimizing various cost factors.

There also exists some work (such as [11, 12, 13]) specifically on simulation/emulation partitioning. The work in [11] focuses on the theoretical analysis of how factors such as load balancing and communication overhead would impact the simulation performance, with the presumption that the effect of the simulation model on the performance factors is already known. The work in [12, 13] makes the simplifying assumption that the effect is constant. This work, in general, overlooks the specifics of the actual architecture of computer cluster or other distributed computing hardware. It, therefore, ends up not making full use of the specific computing environment available.

This paper is organized as follows. In the next section we present an overview of our BencHMAP methodology. In section 3 we give a brief background on parallel and distributed simulation, and discuss conceptually how the properties of the simulation model represent factors that affect the performance. Section 4 will present more details of the partitioning procedure, particularly on how to derive the factor/performance relations based on the results of the benchmarking experiments. Section 5 gives concrete examples of factor/performance relations on our computer cluster, and Section 6 presents the performance of simulation partitioned using our BencHMAP heuristic and how this performance compares when other heuristics are used. Then Section 7 concludes the paper.

2. Overview of BencHMAP

Given a network simulation model and a computer cluster, our goal is to partition the network simulation model into a number of parts to be run on the cluster, so that the partitioned simulation runs as fast as possible.

Compared to some other computation partitioning tasks, network simulation partitioning has the obvious advantage that a network topology can be naturally mapped into a graph, where each network node is represented by a vertex, and each network link is represented by an edge in the graph. Therefore we can reduce the network simulation partitioning task into a graph partitioning task, and employ existing graph partitioning tools which utilize well-researched graph partitioning algorithms that can efficiently produce high quality partitioning.

Our approach, then, is to reduce the simulation partitioning problem into the well-known *weighted graph partitioning* problem [14]. The weights on the graph are determined through a heuristic procedure that involves benchmarking of simulation tasks derived from

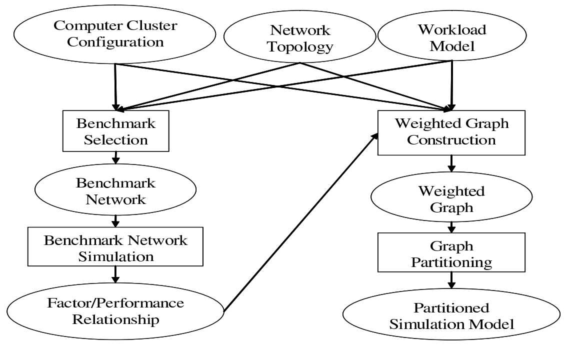

Figure 1. Procedure of Benchmark-Based, Hardware and Model Aware Partitioning

the particular model to be partitioned on the computing environment (*computer cluster*) we desire to partition over.

It is well known that there are three factors, *load balancing, lookahead and communication overhead*, that can affect the parallel simulation performance [15]. We discuss these three factors in detail in the next section. The main challenge here is how to obtain a good trade-off among these three factors while taking into account the given simulation model and the computer cluster hardware configuration in order to make the best use of the given resources. This is difficult because there is no known analytical model to predict the effect of the simulation model and hardware configuration on these three factors in a particular simulation. In our BencHMAP methodology we employ a benchmark-based scheme to empirically determine how the performance of a given simulation model running on a specific hardware configuration would be affected by these three factors.

The overall partitioning procedure is shown in Figure 1. With the given computer cluster configuration and a simulation model (including network topology and workload or traffic model), first we need to construct and run a set of benchmark experiments to derive the factor/performance relations. We then apply the derived relations to the given network topology and workload model to construct a weighted graph. The graph is then input into a graph partitioning tool to produce a partitioned simulation model.

3. Factors Affecting Parallel Simulation Performance

3.1. Load Balancing

Partitioning a simulation results in a division of the resource requirement (including computation, I/O, and memory requirement) of the simulation into a number of parts. Each part is then allocated to a computing resource (processor and memory). Observe that a partitioned simulation is complete only when the last partition completes processing its last event. Ignoring communication and synchronization overhead, minimizing this time means that simulation should be partitioned such that the total event execution time (= number of events to be executed / rate of event processing at a machine) for each machine should be made equal. Note that this total event execution time is a complex function of CPU speed, memory speed, memory sharing architecture and other hardware and operating system factors. It is in general not possible to determine this number *a priori* for any hardware.

3.2. Lookahead

In the field of Parallel And Distributed Simulation (PADS) research, the term *lookahead* means the simulation time window in which a system can process its events without the fear that it will receive past events from other systems. A larger lookahead value has two benefits:

1. More parallelism and more speedup: This is because with large lookahead, each system can independently process more events before each step of synchronization. Hence large lookahead can reduce the total idle time each system has to spend waiting for other systems to catch up with it after this system finishes processing its current safe events before other systems.
2. Less synchronization overhead: When there are a large number of systems participating in the PADS, the cost of each step of synchronization (such as LBTS computation [15]) sometimes cannot be neglected. Since the number of LBTS computations is roughly the total simulation time over the lookahead value, larger lookahead value means less number of times LBTS is computed, hence less total synchronization overhead.

When partitioning packet-level network simulation, we can either split the simulation between protocol layers (*split-stack*) [16], or split at the links between network nodes (*split-link*). The split-stack approach is good for composing simulations from different network simulators that excel at different protocol layers. However, it does not bring much parallelism benefit because it provides zero or near-zero lookahead, since network simulation models usually assume a packet takes no time or very little time to travel between protocol layers. The split-link approach, on the other hand, has the potential of providing much better parallelism, since in this case lookahead between two partitions takes the value of the lowest link propagation delay among the links that are split to provide the partition. We focus on the split-link approach in this paper.

However, as we will see in later sections, there is a threshold of lookahead value beyond which larger lookahead no longer brings obvious advantage. This is because this threshold of lookahead value already provides a large enough window, and keeps all systems executing partitions busy all the time without exhausting safe events before the next LBTS computation. In this case, there is no more room to improve on aspect 1) discussed above, and the effect of aspect 2) can be neglected when the frequency at which LBTS is computed is small enough.

Therefore, our second simulation partitioning objective is to find the partitioning result whose lookahead (i.e. smallest split link propagation delay) is larger than the lookahead of other partitioning's, while treating all partitioning results with a lookahead larger than the threshold as optimal in the sense of lookahead optimization.

3.3. Communication Overhead

Other than synchronization, another kind of interaction between different systems in PADS is the remote scheduling of certain events. In particular, in packet-level network simulation, when a network node delivers a packet to a next hop through a link, it needs to schedule a receiving event at the next hop. But if the next hop is simulated by another system, i.e., the link is a split link, then information about this packet has to be actually transmitted from the current system to the next hop's system, and the receiving event at the next hop has to be remotely scheduled. The cost associated with this transmission sometimes cannot be neglected when different systems are connected through physical network links (e.g., an Ethernet) and not through shared memory, and there is a large number of such transmissions going on in the simulation. We call the packets transmitted between systems the *cross traffic.*

To reduce the total communication overhead, we want to find a partitioning that can reduce cross traffic as much as possible. This means we need to estimate how much traffic each link would have to carry during the simulation, then try to split at the links with the least traffic.

4. The BencHMAP Methodology for Network Simulation Partitioning

As shown in the flowchart in Figure 1, the overall framework of network simulation partitioning can be roughly divided into three segments: 1) the input of computer cluster configuration, network topology and workload model, 2) the determination of factor/performance relations through benchmark experiments, and 3) partitioning the given network simulation model with the derived relations. In this section we will discuss these three segments in more details, with the three performance factors (load-balancing, lookahead, and communication overhead) in mind.

4.1. Input

Network topology: Our ultimate goal is to partition this input network topology into a number of sub-topologies, each sub-topology containing a number of nodes and links of the original topology, with a number of *split links* representing the links split by the partitioning process between sub-topologies.

Current large network simulation models usually model the network topology by assuming a number of end-nodes and a number of routers, each end-node connected to only one router to represent the last hop link from the network to the end host, and the routers are interconnected to represent the backbone network. If the simulation is trying to approximate a realistic Internet scenario, the links closer to the core (i.e., farther away from the end hosts) would tend to carry a larger volume of aggregate traffic. Therefore, if we only consider the cross-traffic factor in the partitioning process, the links to split would mostly be the last hop links. But adding in the lookahead and load-balancing factors would split the topology at links closer to the core, producing better overall performance. We resolve this tradeoff through the benchmarking process.

Workload Model: The workload model represents the traffic flows being simulated. The end hosts run certain applications and send packets to each other through the routers in between. A rough indicator of the amount of processing required in a simulation is the number packet-hops [1] that the workload flows generate during the simulation [17]. This is because each packet-hop typically results in two events, one for a packet's transmission and one for its reception. To provide load balancing in the partitioning it is desirable to balance the packet-hops resulting from the workload in each partition.

It is usually difficult to precisely predict packet-hops before actually running the simulation, because some packets could get dropped at congested links, and it is hard to predict when and where congestion would occur beforehand. However, if congestion is not severe or it does not last long, one may be able to estimate the number of packet-hops processed in the simulation as a whole.

1 A *packet-hop* is one packet traversing a single hop.

Computer Cluster Configuration: The typical platform for running PADS is a computer cluster that consists of a number of stand-alone machines interconnected through a high-speed LAN; each machine has a number of CPUs sharing a certain amount of memory. In this paper we only consider homogeneous computer cluster, where every machine in the cluster has the same number of the same type of CPU, and the same amount of memory. Such clusters are commonly used for such simulations. Dealing with heterogeneous cluster is left for future study.

Since the primary goal of the partitioning task is to make the PADS run as fast as possible on this platform, we need to take a more careful look at the characteristics of this platform. When each CPU runs a part of the simulation, the interaction [2] cost between two parts can vary greatly depending on whether the two CPUs running the two parts belong to the same machine or different machines. If the two CPUs belong to the same machine, then the interaction is done through fast shared memory communication; but if the CPUs are on different machines, then the interaction would have to go through the much slower physical network links. Therefore the impact of interaction is very different between these two cases. We should take this discrepancy into account when devising the partitioning approach.

4.2. Determining Factor/Performance Relations by Benchmark Experiments

Benchmark Selection → Benchmark Network: Given a computer cluster configuration and a class of large network simulation models, ideally we should be able to abstract the essential features from the detailed models, and construct a set of much smaller benchmark simulation models. The relations between the performance factors and the performance that we derive from these benchmark experiments should be applicable to the original simulation model with similar effectiveness.

For example, a basic set of benchmark networks can consist of two symmetric groups of nodes and links, each group simulated by a partition, and the two partitions are connected through a split link. Then we could vary the link delay of the split link to measure the effect of lookahead, vary the cross-traffic ratio to measure the effect of communication overhead, and vary the traffic on each side to measure the effect of load-balancing between two partitions on the overall performance. Section 5 will give a concrete example of this kind of benchmarking.

2 As described before, the interaction includes synchronization and packet transmission.

Benchmark Network Simulation → Factor/Performance Relations: After running the set of benchmark experiments, we should have a set of measurements of overall simulation performance vs. varying lookahead, cross-traffic ratio and load-balancing. We can then plot these measurements to determine the formulas that approximate these relations, and later use these formulas to estimate the effect of the factors on the overall performance of actual simulations.

Specifically, in this step our goal is to determine the following functions and relative weights, which can be combined together later to calculate the edge weights when partitioning actual simulation models.

- $F_{LA}(Delay)$: This function approximates how the variation of lookahead value (i.e., split link delay) would affect the simulation performance. To determine this function, we can run on the given computer cluster a set of benchmark partitioned simulations that keep other factors constant but vary the split link delay value, and measure the performance of these simulations. Then we can plot the performance against lookahead value to extrapolate a specific formula of $F_{LA}(Delay)$.
- $F_{CT}(CrossTraffic)$: This function approximates how the variation of the cross traffic amount over a split link would affect the simulation performance. To determine this function, we can run on the given computer cluster a set of benchmark partitioned simulations that keep other factors constant but vary the cross traffic volume, and measure the performance of these simulations. Then we can plot the performance against cross traffic to extrapolate the formula of $F_{CT}(CrossTraffic)$.
- W_{LA} and W_{CT}: These are two relative weights representing the comparison of the effects of lookahead and cross traffic on the simulation performance. Getting the right weights would result in good trade-off between these two factors when partitioning the actual simulation. To determine the two weights, we can run on the given computer cluster a set of benchmark partitioned simulations that vary both the lookahead and cross traffic volume, and measure their performance. Then we can plot the performance against cross traffic and link delay in a 3-D plot to deduce the relative weights between cross traffic and link delay.

4.3. Partitioning the Network Simulation Model

Weighted Graph Construction → Weighted Graph: The weighted graph consists of both weighted edges and weighted vertices. In the simulation the network nodes and links can be naturally mapped to graph vertices and edges, therefore we only need to determine the weights of the vertices and the edges.

- *Edge Weight:* Since the weight of an edge should be determined by combining the lookahead factor and cross-traffic factor, after the functions $F_{LA}(Delay)$ and $F_{CT}(CrossTraffic)$ and the weights W_{LA} and W_{CT} have been determined as described above, the weight of an edge can be calculated from:

$$W_{total} = F_{LA} * W_{LA} + F_{CT} * W_{CT}$$

- *Vertex Weight:* For the load balancing objective described in Section 3.1, vertex weights can be converted from the load associated with each node. A rough indicator of the load is the number of packets a node sends/receives in the simulation. One way to estimate beforehand the total number of packets each node sends/receives in the simulation is, traverse the path of every traffic stream from one end node to the other end node, and add the estimated number of packets in this traffic stream to a variable associated with each node on the path. However this is just rough estimation, because the actual number of packets transferred in a stream can be different in the actual simulation.

Graph Partitioning → Partitioned Simulation Model: After constructing the weighted graph, we can use an existing graph partitioning tool to partition the graph into a number of parts, and create the partitioned simulation model to be run in parallel by different machines. The graph partitioning tool we use in this work is the METIS package developed by UMN [18]. METIS is by far the fastest graph Partitioning tool we know of. This tool takes a graph with vertex weights and edge weights as input, tries to partition the graph with balanced vertex weights and minimized edge-cut weights, then outputs which part each vertex belongs to.

To address the communication overhead difference between communicating across shared memory and communicating across a network link discussed before, we devise a simple *two-level partitioning* approach in which we use the graph partitioning tool twice. Suppose there are n machines each with m CPU's, we first partition the whole simulation at the cluster level into n parts, with a large weight associated with each link; then partition each part further into m sub-parts with a small weight assigned to each link. When starting the simulation, we start the $n * m$ sub-parts at the same

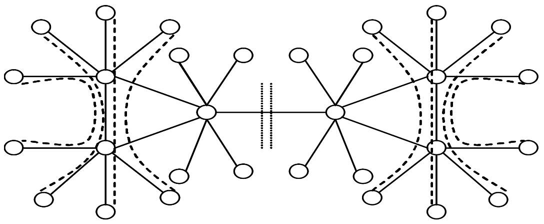

Figure 2. Benchmark simulation with 0 cross traffic stream

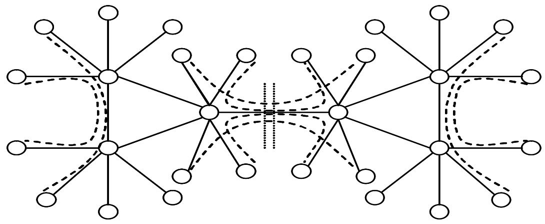

Figure 3. Benchmark simulation with 4 cross traffic streams

time on the n machines, with the rule that sub-parts of the same part must be started on the same machine, then the operating system would automatically assign each sub-part to a CPU of the machine.

The first level of partitioning tries to reduce interaction between the n parts as much as possible, since the interaction across physical network link is comparatively much more expensive. The second level of partitioning on the other hand focuses much more on load balancing; because the interaction between CPU's through shared memory is fast.

One point to note is that, in the second level partitioning, though the weight associated with link traffic should be reduced significantly, the weight associated with link delay should not be reduced. This is because when all the $n*m$ sub-parts are running, for each sub-part the simulation engine always picks the smallest split link delay between this sub part and the other $n*m-1$ sub-parts as the lookahead. Therefore, partitioning at the second level should also result in split links with large enough delays to avoid dragging down the overall lookahead in the whole simulation.

5. A Benchmarking Case Study

This section describes a set of benchmarking experiments that we ran in order to obtain appropriate graph weights. The computer cluster used in this work consists of 8 machines interconnected through a Gigabit LAN, each machine having two 3GHz P4 CPUs and 2GB RAM. The simulator we use is the Parallel and Distributed ns (PDNS) [6] developed within our group. PDNS is a parallel and distributed version of the ns2 simulator, relying on the Libsynk/RTI library [19] as the underlying synchronization and communication infrastructure between different simulator instances.

The following subsections will describe simple examples of the benchmark simulations, how we design them, run them and use the measurement to derive the factor/performance relations and weights.

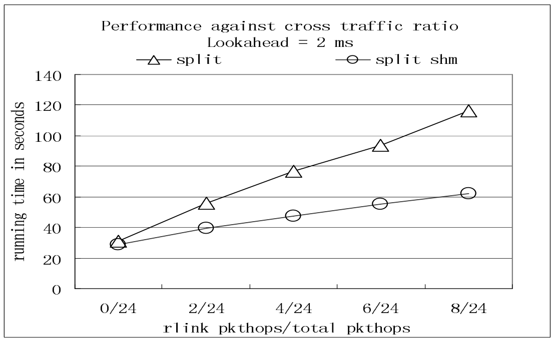

Figure 4. Varying cross traffic ratio

5.1. Communication Overhead: Varying Ratio of Cross Traffic

To determine the effect of one performance factor in the partitioned simulation, we need a set of benchmark simulations that vary the value of this factor while keeping other factors unchanged. As shown in Figures 2 and 3, the simple benchmark simulation model is designed so that each packet traverses exactly three links (3 hops), because the outer nodes send packets to vertical counterparts, and internal nodes send packets to horizontal counterparts. We split the link in the middle to partition this topology into two parts, each part being simulated by one PDNS instance.

In this set of benchmark simulations we maintain the total number of traffic streams at eight, and since each traffic stream traverses 3 hops, the total is 24 hops. The load between the two partitions remains evenly divided. We vary the ratio of (cross traffic packet-hops/total traffic Packet-hops) by increasing the number of horizontal traffic streams while decreasing vertical traffic streams by the same number, hence keeping the total packet-hops unchanged.

Figure 2 shows 0 cross traffic streams, and since the total number of hops is 24 for all streams, the cross traffic ratio in this case is 0/24. Figure 3 shows 4 cross traffic streams, i.e., 4 cross traffic hops, and since the total traffic is still 8 streams, i.e., 24 hops, the cross traf-

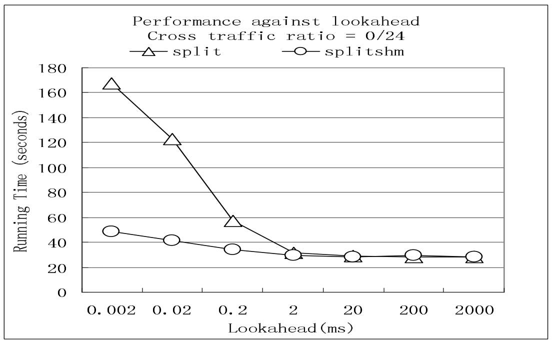

Figure 5. Varying split link delay

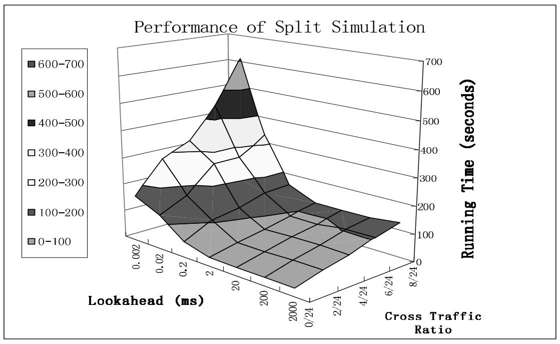

Figure 6. Varying both lookahead and cross traffic ratio for split simulation on two machines

fic ratio in this case is 4/24.

Varying the cross traffic ratio from 0/24 to 8/24, we get the performance plot shown in Figure 4. The two lines represent respectively the running time of the split simulation running on two machines connected via LAN (SPLIT), and the split simulation running on two CPUs on the same machine communicating via shared memory (SPLIT-SHM).

We can see that as the cross traffic ratio increases, the running time of both SPLIT and SPLIT-SHM increase linearly. Starting from about the same point, SPLIT increases faster than SPLIT-SHM, as expected, because the packet transmission over the LAN is much slower than over shared memory.

Therefore we can obtain F_{CT} of a link from the traffic amount on this link divided by the total traffic amount:

$$F_{CT}(CrossTraffic) = \frac{CrossTraffic}{TotalTraffic} * 24 + C$$

5.2. Lookahead: Varying Split Link Delay

To measure the effect of lookahead on the performance of split simulations, we use the same simulation scenario in Figure 2, i.e., no cross traffic between two partitions. Then we vary the propagation delay of the split link to create different lookahead values between the two partitions. The result can be seen in Figure 5. Notice that when lookahead is smaller than 2ms, both SPLIT and SPLIT-SHM running times drop logarithmically as lookahead increases. But when lookahead is greater than 2ms, the lines become flat, which means that lookahead larger than 2ms does not improve the performance. This is because, as we explained in section 3.2, lookahead values beyond the threshold result in few synchronizations anyway.

Therefore we can obtain F_{LA} of a link by:

$$F_{LA}(Delay) = \begin{cases} \log(LA_{th}/Delay) + C, & Delay < LA_{th} \\ C, & Delay \geq LA_{th} \end{cases}$$

Where $Delay$ is the actual propagation delay of this link, LA_{th} is the threshold value, and in this case $LA_{th} = 2ms$. Here the log calculation is to convert the logarithmic decrease to linear decrease for the convenience of linear weighted sum calculation of W_{total}.

5.3. Varying Both Lookahead and Cross Traffic

Now that we know how lookahead and cross traffic affect the performance individually, the remaining problem is to determine the relative weights between these two factors. Using the same topology in Figure 2, we vary both the lookahead and the cross traffic ratio for SPLIT, and got the performance measurements in Figure 6

From these measurements, we notice that in these cases, the rough trend of the surface is that when it climbs along both axes, moving 2 steps along the axis of cross traffic ratio is about equivalent to moving 1 step along the axis of lookahead. Therefore the weights are roughly:

$$W_{CT} \approx 1, \;\; W_{LA} \approx 2$$

6. Partitioning Experimental Results

In this section we use the factor/performance relations that we obtained from the benchmark simulation above to compute a weighted graph derived from a large-scale network model. For the network topology we use MILNET, an Internet topology developed in [20]. We convert the MILNET topology from the original XML format to ns2 format, and create random traffic streams for the test. More specifically, the topology we test in this research consists of 123,536 nodes and 132,341 links. Among the nodes, 5,775 are routers and 117,761 are end hosts. We create 100,000 FTP streams, for each stream selecting two random end hosts to be the source and the sink. The simulation stops at the

40th second of simulation time. This scale of simulation is not achievable with ns2 on a single workstation. We use 4 machines (each with 2 CPUs) from the cluster described in the previous section. Our goal is to partition the simulation into 8 parts, and run the 8 parts in pdns in parallel.

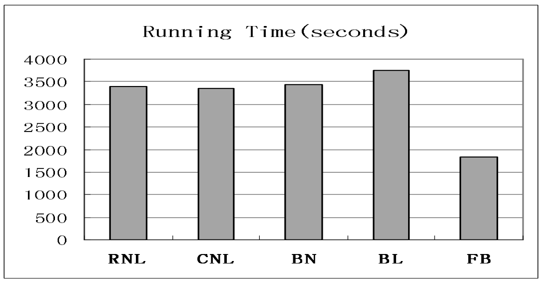

Figure 7. Simulations' performance

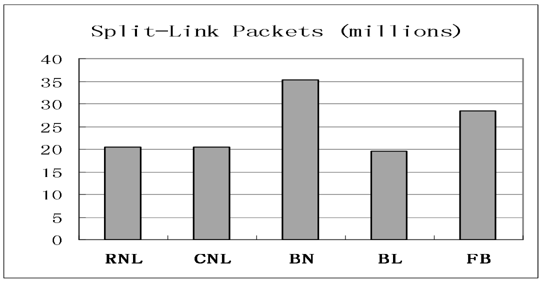

Figure 8. Total packets transmitted on split-links

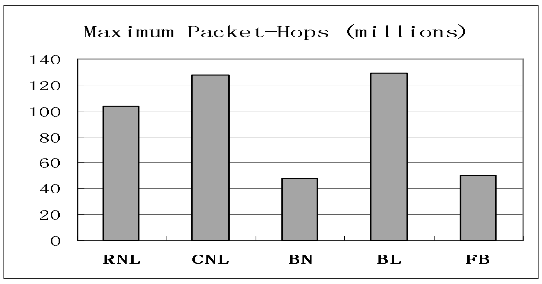

Figure 9. Maximum partition packet hops among 8 partitions

As mentioned before, we partition the simulation in two steps. The first step is at the cluster level where we partition the whole simulation into 4 parts, each part to be run on a machine in the cluster, and the 4 parts communicate through the LAN. The second step is at the machine level where we further partition each of the 4 parts into 2 sub-parts, so that each sub-part can be run on a different CPU, and these 2 sub-parts communicate through shared memory. A partitioned simulation is then run with pdns on 4 machines in the cluster to measure the performance resulting from a partitioning strategy.

Here we compare the BencHMAP with several other partitioning strategies:

- *RNL*: We assign a random weight to each node and each link before using METIS to partition the simulation. With this strategy the result of partitioning is 8 random partitions, each partition with a random size.
- *CNL*: We assign a constant node weight to each node, and a constant link weight to each link. With this strategy the partitioning results in 8 partitions with about the same number of nodes, since METIS tries to balance the load between different partitions.
- *BencHMAP-Link(BL)*: We assign a constant weight to each node, and calculate link weights with the traffic and link delay information using the benchmark results from the previous section. This strategy tests how link weight alone affects the partitioned simulation performance.
- *BencHMAP-Node(BN)*: We assign a constant weight to each link, and calculate node weight with the work load information. This strategy tests how node weight alone affects the partitioned simulation performance.
- *Full-BencHMAP(FB)*: This is the full application of our methodology in which both node weights and link weights are calculated from the estimated work load of each node as well as the benchmark results.

Figure 7 shows the performance of the partitioned simulation of these partitioning strategies. Figure 8 shows the total number of packets transmitted on the split links in the simulation in each case. Since the split-link packets transmitted over LAN constitute the main interaction cost in the parallel simulation, more split link packets would result in slower simulation. Figure 9 shows the maximum number of processed packet-hops among the 8 partitions in each case. This is an indicator of the load-balance factor, and since the performance is to a large degree determined by the slowest partition, i.e., the partition that processes the most packet-hops, larger maximum packet-hops would result in slower simulation.

As we can see in the figures, *RNL* and *CNL* run slow mainly because load is imbalanced: the maximum packet-hops is too high for these two cases. *BN* balances the load well and results in a low maximum packet-hops, but since it does not take into account link weights, too many split-link packets makes the simulation slow as well. *BL* considers link weight, therefore reducing split-link packets, but ignoring load balancing factor results in a high maximum packet-hops, slowing down the whole simulation. This shows how important it is to get a good trade-off between the different factors for a good performance. *FB* takes all factors into account, producing a moderate number of split-link packets and better load-balancing, resulting in the fastest simulation.

7. Concluding Remarks

In this paper we propose BencHMAP: Benchmark-Based Hardware and Model Aware Partitioning, a methodology to partition large scale network simulations to best exploit the hardware resources and create the best simulation performance. We also discuss the factors that affect the performance of parallel and distributed network simulation, and how we can get a good trade-off between the factors.

We give examples of how to use simple and fast benchmark simulations to obtain the formulas that approximate the relations between the factors and the performance of parallel simulation. These formulas are then applied to the partitioning of a large scale network simulation with 120k nodes 100k traffic streams. Comparing various partitioning techniques, we show that the full application of BencHMAP results in the best trade-off between various factors, therefore producing the best performance for our parallel network simulation.

We are in the process of developing a semi-automated network simulation partitioning tool based on BencHMAP for public release.

Acknowledgments

We would like to thank George F. Riley, Richard M. Fujimoto and Steve L. Ferenci for the initial discussions on this research and for various help they provided in the process.

References

[1] S. McCanne and S. Floyd, "The LBNL network simulator," Software on-line: http://www.isi.edu/nsnam, 1997, Lawrence Berkeley Laboratory.

[2] S. Bertolotti and L. Dunand, "Opnet 2.4: an environment for communication network modeling and simulation," in *Proceedings of the European Simulation Symposium*, October 1993.

[3] X Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks," in *Proceedings of the 12th Workshop on Parallel and Distributed Simlations*, May 1998.

[4] Donghua Xu, George F. Riley, Mostafa H. Ammar, and Richard M. Fujimoto, "Enabling large-scale multicast simulation by reducing memory requirements," in *Proceedings of Workshop on Parallel and Distributed Simulation*, 2003.

[5] G. F. Riley, E. W. Zegura, and M. H. Ammar, "Efficient routing using nix-vectors (long version)," May 2001.

[6] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, "Parallel/Distributed ns," Software on-line: www.cc.gatech.edu/computing/compass/pdns/ index.html, 2000, Georgia Institute of Technology.

[7] J. H. Cowie, D. M. Nicol, and A. T. Ogielski, "Modeling the global internet," *Computing in Science and Engineering*, January 1999.

[8] Hesham El-Rewini and Ted G. Lewis, *Distributed and Parallel Computing*, Manning Publications Co., 1998.

[9] Ian Foster, *Designing and Building Parallel Programs*, Addison-Wesley Publishing Company, 1995.

[10] Sudarsan Tandri and Tarek S. Abdelrahman, "Computation and data partitioning on scalable shared memory multiprocessors," in *International Conference on Parallel and Distributed Processing Techniques and Applications*, 1995.

[11] David Nicol, "Scalability, locality, partitioning, and synchronization in pdes," in *Proceedings of the Parallel and Distributed Simulation Conference*, 1998.

[12] Xin Liu and Andrew A. Chien, "Traffic-based load balance for scalable network emulation," in *Proceedings of the ACM Conference on High Performance Computing and Networking, SC2003*, 2003.

[13] Ken Yocum, Ethan Eade, Julius Degesys David Becker, Jeff Chase, and Amin Vahdat, "Toward scaling network emulation using topology partitioning," in *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 92.

[14] K. Schloegel, G. Karypis, and V. Kumar, "Graph partitioning for high performance scientific simulations," 2000.

[15] R. Fujimoto, *Parallel and Distributed Simulation Systems*, Wiley Interscience, 1999.

[16] Donghua Xu, George F. Riley, Mostafa H. Ammar, and Richard M. Fujimoto, "Split protocol stack network simulations using the dynamic simulation backplane," in *Proceedings of the Ninth International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2001.

[17] Richard Fujimoto, Kalyan Perumalla, Alfred Park, Hao Wu, Mostafa Ammar, and George Riley, "Large-scale network simulation – how big? how fast?," in *Proceedings of International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems*, 2003.

[18] George Karypis, Rajat Aggarwal, Kirk Schloegel, Vipin Kumar, and Shashi Shekhar, "Metis: Family of multilevel partitioning algorithms," Software online: http://www-users.cs.umn.edu/ karypis/metis/, 2003.

[19] Kalyan Perumalla, "libsynk/rti," Software online: http://www.cc.gatech.edu/computing/pads/kalyan/ libsynk.htm, 2004.

[20] Michael Liljenstam, Jason Liu, and David M. Nicol, "Development of an internet backbone topology for large-scale network simulations," in *Proceedings of the 2003 Winter Simulation Conference (WSC'03)*, 2003.