Tackling the Memory Balancing Problem for Large-Scale Network Simulation

Hyojeong Kim Kihong Park
Department of Computer Science
Purdue University
West Lafayette, IN 47907
{hjkim,park}@cs.purdue.edu

Abstract

A key obstacle to large-scale network simulation over PC clusters is the memory balancing problem where a memory-overloaded machine can slow down an entire simulation due to disk I/O overhead. Memory balancing is complicated by (i) the difficulty of estimating the peak memory consumption of a group of nodes during network partitioning—a consequence of per-node peak memory not being synchronized—and (ii) trade-off with CPU balancing whose cost metric depends on total—as opposed to maximum—number of messages processed over time. We investigate memory balancing for large-scale network simulation which admits solutions for memory estimation and balancing not availed to small-scale or discrete-event simulation in general. First, we advance a measurement methodology for accurate and efficient memory estimation, and we establish a trade-off between memory and CPU balancing under maximum and total cost metrics. Second, we show that joint memory-CPU balancing can overcome the performance trade-off—in general not feasible due to constraint conflicts—which stems from network simulation having a tendency to induce correlation between maximum and total cost metrics. Performance evaluation is carried out using benchmark applications with varying traffic characteristics—BGP routing, worm propagation under local and global scanning, and distributed client/server system—on a testbed of 32 Intel `x86` *machines running a measurement-enhanced DaSSF.*

1. Introduction

Large-scale network simulation is a multi-faceted problem [5, 22] spanning synchronization, network modeling, simulator design, partitioning, and resource management, to mention a few. In this paper we focus on a key obstacle to large-scale network simulation over PC clusters—the memory balancing problem—where a PC whose memory resources get overloaded can slow down an entire simulation. As with CPU balancing, an overloaded machine is a weak link whose disk I/O overhead stemming from memory management by the operating system can significantly slow down distributed simulation. Fig. 1 illustrates the impact of memory overload on BGP routing simulation running on a single Linux PC configured with 2 GB or 1 GB memory ("BGP A"). There is a factor 9.4 slow down between the two. "BGP B" compares completion time of a

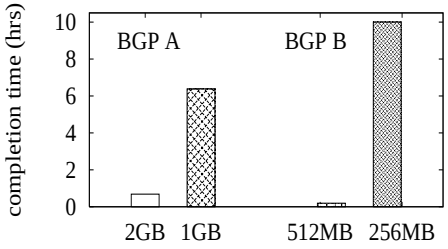


Figure 1. Completion time slow down of BGP simulation due to disk I/O overhead.

different BGP simulation instance on 512 MB and 256 MB memory configurations found in older clusters where there is a factor 52.3 difference. Slow down due to disk I/O overhead can vary significantly depending on physical memory, application memory referencing behavior, and operating system support [2, 6]. In distributed simulation, the slow down factor may get further amplified.

We tackle the memory balancing problem for large-scale network simulation by taking a memory-centric approach to network simulation partitioning. Memory-centric load balancing does not mean that we ignore CPU and communication balancing, but that we draw attention to idiosyncratic features that inject additional technical subtlety. Chief among them are (i) the difficulty of estimating the peak memory consumption of a group of nodes during network partitioning—a consequence of per-node peak mem-

ory (a simulated node's maximum memory requirement over time) not being synchronized—and (ii) trade-off with CPU balancing whose cost metric depends on total, as opposed to maximum, number of messages processed over time. Our study shows that memory balancing for large-scale network simulation admits solutions for estimation and balancing that are not availed to small-scale or discrete-event simulation in general.

The technical contributions are as follows. We advance a measurement methodology for accurate and efficient memory estimation applicable to large-scale network simulation which addresses issue (i). We establish a trade-off between memory and CPU balancing under maximum and total cost metrics whose performance gap depends on network topology and application traffic. This addresses part of issue (ii). We show that joint memory-CPU balancing can overcome individual performance trade-off which, is in general, not feasible due to constraint conflicts. We show that multi-objective optimization prowess is due to network simulation having a tendency to induce correlation between maximum and total cost metrics.

The remainder of the paper is organized as follows. In the next section, we summarize related works. In Section 3 we describe the performance evaluation framework. In Section 4 we evaluate memory and CPU balancing trade-off. Section 5 provides analysis of balancing performance and memory estimation. In Section 6 we study joint memory-CPU load balancing. We conclude with a discussion of our results.

2. Related Work

Network simulation partitioning has been studied in a number of papers [12, 27, 29] perhaps the most relevant to our work being BencHMAP [27]. BencHMAP is a general framework for network simulation partitioning whose scope included memory balancing. The main concern was managing CPU, communication, and synchronization cost (i.e., lookahead), with memory balancing receiving tangential treatment as part of total message balancing which was given as node weight input during benchmarking. The latter roughly corresponds to total cost metric in our study. In [29] topological partitioning is studied for scaling network emulation where component partitioning is driven by estimated communication cost. In [12] focus is placed on communication cost with respect to cross traffic and parallel speed-up in PDNS for partitioning simple topologies.

In [3, 8, 28] reducing memory requirement is studied from an application based perspective, and in [16, 18] simulator memory requirements are considered from the systems side. In [18] subtleties underlying comparative evaluation of simulators, including memory footprint, are highlighted where workload configuration, as a function of network size, is shown to significantly influence outcome. Several works follow a benchmark driven partitioning approach to scalable network partitioning [21, 27]. Paucity of memory-centric load balancing also holds in the parallel distributed computing community where focus has been on computation and communication balancing [9, 11]. In recent work [17], CPU-memory balancing has been studied using an adaptive application-driven approach aimed at scientific applications. In [26, 30], CPU-memory balancing is evaluated from a dynamic load balancing perspective with job assignment and migration incorporating CPU and memory balancing constraints.

3. Performance Evaluation Framework

3.1. Distributed Simulation Environment

We use a modified version of DaSSFNet as our distributed network simulation environment. DaSSF (Dartmouth SSF) [24] is a realization of SSF [23] written in C++ that is well suited for distributed simulation over PC clusters. DaSSFNet is an extension of DaSSF that implements a network stack and user API over its simulation kernel. The main modification we have added is a measurement subsystem that keeps track of dynamic simulation events spanning memory, computation, and communication both inside the simulation kernel and in the protocol stack outside the kernel. Distributed synchronization across network partitions is effected through a barrier mechanism that occurs at fixed time granulity (i.e., epoch)—the the minimum link latency of inter-partition links—which assures causal ordering of distributed simulation events [15, 19]. To focus on memory and CPU balancing, we set link latency to a uniform value which limits dependence of synchronization cost with respect to lookahead on network partitioning. Distributed coordination is implemented over MPI.

3.2. Measurement Subsystem

3.2.1. Message-centric Monitoring Evaluating the efficacy of memory, CPU, and communication balancing requires accurate and efficient monitoring of dynamic network simulation events. Network simulation events are dominated by message related objects that are generated, processed, stored, forwarded, and deleted. We track all message related events in DaSSFNet both inside the simulation kernel and the network stack outside the kernel. The protocol stack in DaSSFNet is comprised of IP/NIC, TCP, UDP, and BGP which are accessed via a socket API-like interface. A message related event m is represented by a data structure which has a size attribute $s(m)$ that is dependent on the event type and whether it is copied or pointer based. m has a start time $t_s(m)$ at which it is generated

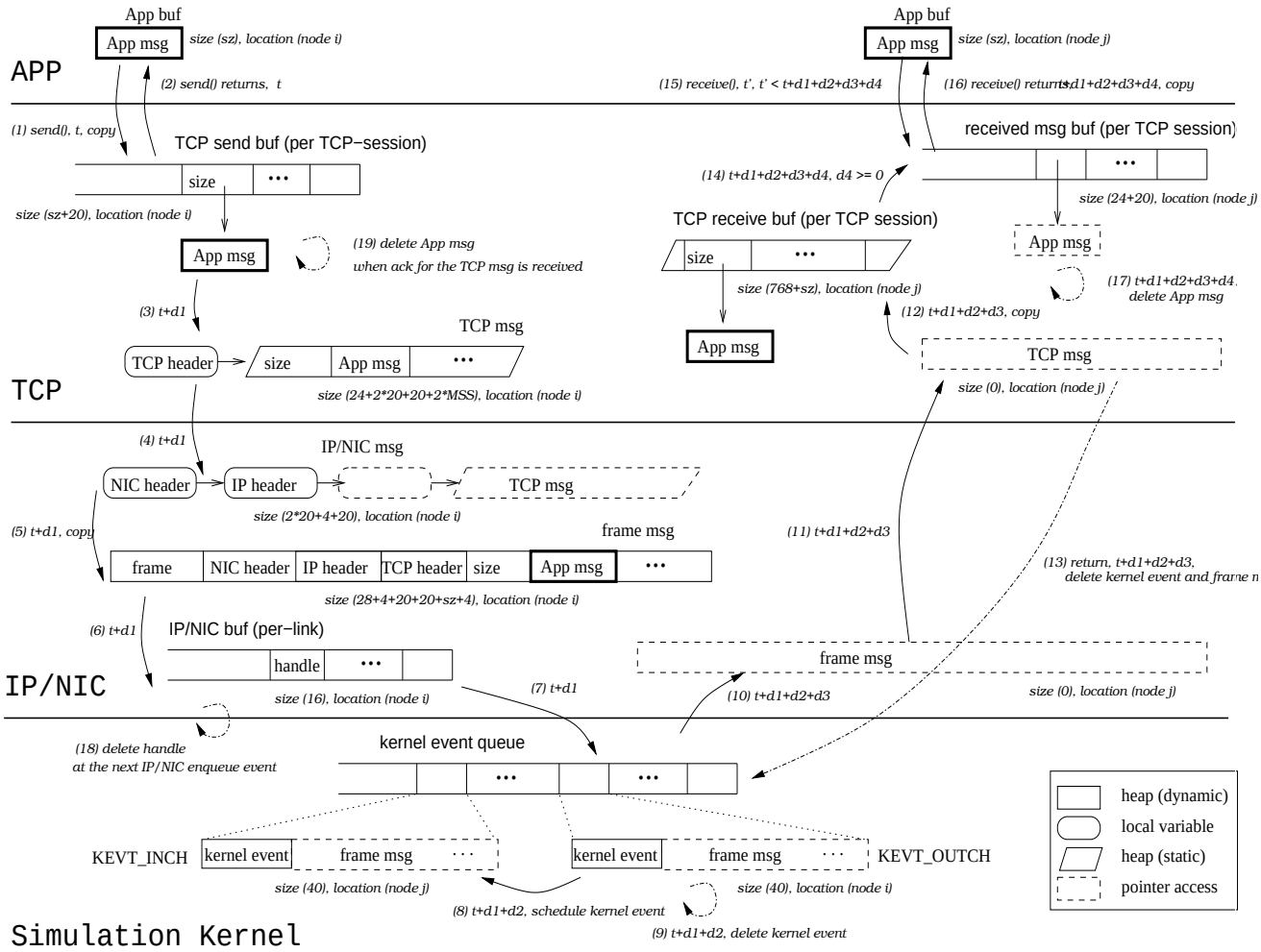


Figure 2. TCP based message event evolution and footprint spanning application layer, DaSSFNet protocol stack, and DaSSF simulation kernel.

and a finish time $t_f(m)$ at which it is deleted. Time stamps are recorded with respect to both simulation time and wall clock time. An important issue for memory consumption monitoring is accurate accounting of dynamic memory allocation and deallocation which is system library and OS dependent. We discuss memory leakage and garbage collection issues in Section 5.2.

3.2.2. Message Event Evolution and Footprint We describe event monitoring for TCP based messaging which is a representative example. Per-node memory, computation, and communication cost accounting is performed by aggregating—maximum over time for memory and total over time for computation and communication—message related events. Fig. 2 shows TCP based message event evolution spanning application, DaSSFNet protocol stack, and DaSSF simulation kernel. The starting point is the application layer where a message is created and passed to TCP at simulation time t. The application message is copied to the TCP send buffer. All instances where message copying is instituted are highlighted in bold. The queued TCP message is processed after a delay d_1 determined by TCP and forwarded to the IP/NIC layer. Another copy operation results when a frame message is created which is enqueued in the IP/NIC output buffer. Queueing delay and transmission time are computed and passed to the simulation kernel along with the frame message by writing to outChannel of the network link.[1] This triggers creation of kernel event KEVT_OUTCH that is enqueued in the kernel event queue. These steps occur in zero simulation

[1] outChannel and inChannel are message interface classes defined by the simulation kernel.

time. KEVT_OUTCH is dequeued at time $t + d_1 + d_2$ where d_2 is the sum of queueing delay and transmission time. If the receiving node is on the same network partition as the sending node, KEVT_OUTCH is transformed to inChannel event KEVT_INCH that is enqueued in the kernel event queue. KEVT_OUTCH is deleted. When link latency d_3 has elapsed, KEVT_INCH is dequeued and the frame message is popped up the protocol stack at the receiving node. If the receiving node is on a different network partition, KEVT_OUTCH is mapped to an intermediate channel event that is held at the machine simulating the sending node until the epoch barrier completes (i.e., MPI packs and transfers the channel object to the machine where the receiving node is simulated). At the receiving machine the intermediate channel event is mapped to KEVT_INCH and enqueued in its kernel event queue with time stamp $t + d_1 + d_2 + d_3$.

Fig. 3 shows monitored memory consumption of message related events of a BGP simulation from a 16-machine distributed simulation. Message related events are aggregated over nodes partitioned to one of the 16 machines. Fig. 3 shows that memory consumption is variable reaching its peak at 1477.81 sec wall clock time (90.0062 sec simulation time). Events are stacked (i.e., cumulative) and

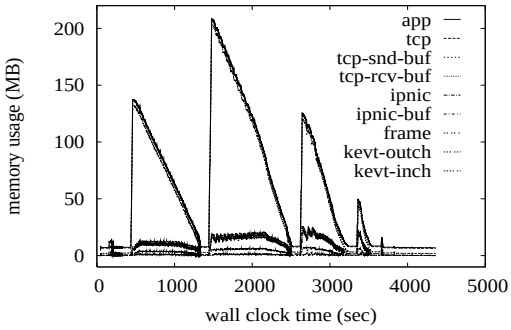


Figure 3. Dynamic monitoring of message related events. `tcp-snd-buf` dominates peak memory usage.

memory consumption is dominated by TCP send buffer queueing. We note that Fig. 2 and Fig. 3 are summary descriptions of the measurement subsystem that highlight the main components. Additional details such as timer and semaphore kernel events and per-node BGP table update events are omitted in the two figures for brevity. Per-node time series logging of measurement data for accurate memory partitioning is infeasible in large-scale network simulation. We discuss space-efficient memory monitoring in Sections 4.1.2 and 5.2 in conjunction with measurement accuracy and overhead.

3.3. Benchmark Set-up

Hardware testbed. The PC cluster used in benchmark experiments consists of 32 Intel `x86` PCs running Linux 2.4.x and 2.6.x. Ten are Pentium 4, 2 GHz machines with 1 GB memory, six are 2.4 GHz with 1 GB memory, and six are 2.53 GHz with 1 GB memory. Five machines are Pentium 2.4 GHz with 2 GB memory and five are Xeon 2.4 GHz with 4 GB memory. L1 cache is 8 KB on all machines, and L2 cache is 512 KB except on the ten 2 GHz machines where it is 256 KB. When comparing CPU balancing performance across different partitions processor speed can become a factor. Testing has shown that 16 2.4 GHz machines yield approximately interchangeable and predictable performance. They are used for CPU balancing comparisons. An even more important factor for meaningful completion time comparison is the variable influence of VM swapping whose magnitude is difficult to predict. To achieve quantitative results that are stable and repeatable, we disable VM swapping so that hidden disk I/O swapping overhead is removed. The PCs form a dedicated testbed connected by 2 GigE and 2 FE switches. Network congestion is not an issue, i.e., there are no packet drops and end-to-end latency is in the submillisecond range.

Benchmark applications. We consider benchmark applications with varying traffic characteristics—BGP, worm propagation under local (i.e., topological) and global scanning, and distributed client/server system—that engage different aspects of the DaSSFNet protocol stack. BGP is a port of the Java SSF implementation of BGP-4 with both hash and trie based route table support. We use RouteViews/NLANR Internet autonomous system (AS) topologies [25] as our default benchmark network graphs. Problem size refers to the number of nodes in the network graphs. In BGP simulations ASes are treated as BGP router nodes. Worm propagation simulation is done at host IP granularity where IP addresses are mapped to individual ASes. Thus the higher the number of infected hosts at an AS, the higher the collective scan rate of the AS. The distributed client/server system assigns file server nodes to transit ASes that are accessed by clients at stub ASes. File servers possess heavy-tailed file sizes (Pareto with tail index 1.35) [1] that induce self-similar burstiness of aggregated traffic [13]. Session arrivals at a client are Poisson. BGP and distributed client/server system run over TCP whereas local and global worm propagation use UDP.

4. Memory-centric Load Balancing

In this section, we evaluate the basic features and performance traits of memory-centric load balancing. Diagnosis of distributed simulation is carried out in Section 5.

4.1. Memory Cost Metric

4.1.1. Per-node Maximum vs. Total Cost Metric Computation cost in network simulation is dominated by messages. They are proportional to the total number of messages that a PC processes over the course of a simulation. If processing cost varies significantly between message types, weighting may be necessary to arrive at normalized cost. The processing cost, C_i, associated with a single node i—for sending, receiving, and mangling messages—is the sum of all messages $X_i(t)$ processed at the node over time, $C_i = \sum_t X_i(t)$, which requires constant space to monitor. The memory cost, M_i, of node i depends on the maximum number of messages over time, $M_i = \max_t X_i(t)$, which can significantly differ from the total C_i. If the memory footprint of different message types varies, weighting is necessary for accurate accounting. All else being equal, we expect M_i to be superior for memory balancing while C_i is expected to favor CPU balancing.

4.1.2. Per-partition Memory Cost An additional issue that arises in the case of maximum cost metric but not total cost metric is when network partitioning based on M_i provided by the measurement subsystem is carried out. Suppose that a partitioning algorithm $\mathcal{A}$ which receives $M_1, \ldots, M_n$ as node weight input assigns two nodes i and j to the same partition based on their sum $M_i + M_j$. The problem with doing so is that although their individual memory peaks are M_i and M_j, their collective memory footprint is given by $\max_t\{X_i(t) + X_j(t)\}$ which need not equal $\max_t X_i(t) + \max_t X_j(t)$. This is depicted in Fig. 4 for memory consumption dynamics of two nodes in a worm propagation simulation where their memory peaks are not synchronized—both wall clock and simulation time—leading to significant overestimation of sum-of-max over max-of-sum. This problem does not arise for the total cost metric due to its closure property. The sum-of-max vs. max-of-sum problem cannot be addressed by logging per-node time series of memory consumption due to prohibitive space complexity (i.e., $O(nT)$ where T is simulation time). We use M_i as input to partitioning algorithms to evaluate memory and CPU balancing performance. In Section 5.2 we explain why M_i is effective despite the sum-of-max overestimation problem.

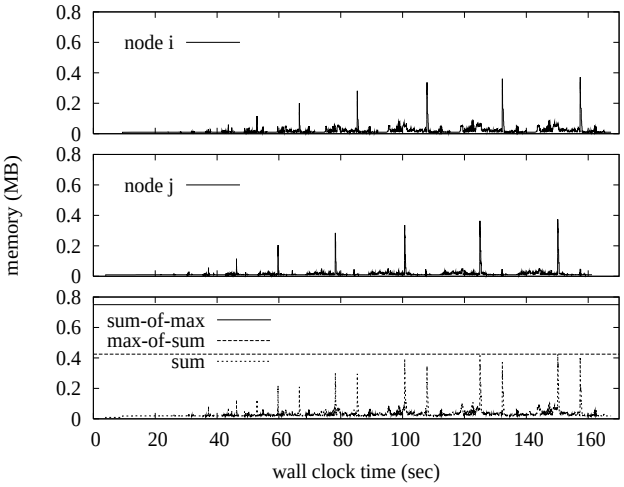


Figure 4. Network partitioning: sum-of-max vs. max-of-sum problem.

4.2. Memory Balancing Performance

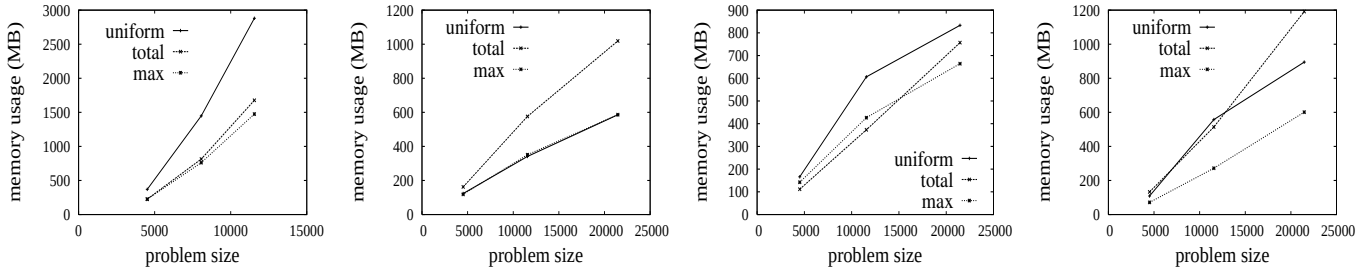
We use Metis [10] which implements a multilevel recursive k-way partitioning algorithm [11] as the default network partitioning tool. Multilevel recursive bisection methods have been shown to yield improved partitioning vis-à-vis spectral methods [9], and their fast running time has made Metis and Chaco [7] commonly used benchmark tools for network partitioning.

4.2.1. Comparison of Maximum vs. Total Cost Metric Fig. 5 compares memory balancing performance with respect to memory usage between M_i (max) and C_i (total) for the BGP, worm, and client/server benchmark applications for a range of problem sizes. Memory usage is with respect to the maximum across all machines. We give $M_1, \ldots, M_n$ or $C_1, \ldots, C_n$ as node weight input to Metis which tries to find a k-way partitioning that minimizes edge cut while balancing node weight across the k partitions. As a reference point, we include memory balancing under uniform node weight in which premium is assigned to reducing edge cut.

We use up to 32 machines (i.e., $k \leq 32$) in the distributed simulations, with 32 machines used for the largest problem instances. Fig. 5 shows that, overall, M_i outperforms C_i with the magnitude of the gap depending on benchmark application and problem size. Worm local and distributed client/server show the biggest gaps with BGP and worm global exhibiting marginal difference between maximum and total cost metrics. We also find that C_i can lead to memory imbalance that is significantly worse than the uniform cost metric (cf. Fig. 5(b)).

4.2.2. Memory vs. CPU Balancing Trade-off As indicated in Section 3.3, to carry out meaningful CPU balancing comparison we need to ensure that processor speeds across different PCs are comparable. This limits us to 16 PCs which also curbs the largest problem instances we can run without engaging VM swapping.

Fig. 6 summarizes memory balancing performance comparing the maximum and total cost metrics. Problem sizes are specified in parentheses. We find that the maximum metric, overall, outperforms the total cost metric



(a) BGP (b) Worm local (c) Worm global (d) Distributed client/server

Figure 5. Memory balancing performance of M_i (max), C_i (total), and uniform cost metrics as a function of problem size for different benchmark applications.

with the gap being highest for worm local and distributed client/server benchmark applications consistent with the above results.

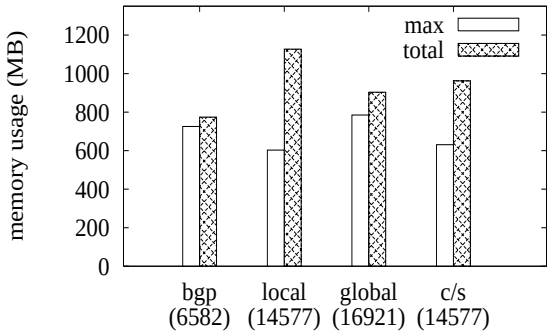


Figure 6. Memory balancing performance with $k = 16$ homogenous machines.

Fig. 7 (top) shows CPU balancing performance with respect to computation time—`usr` and `sys` time of the slowest (i.e., highest processing load) machine—for the same benchmark runs. The memory and CPU balancing results are averages of 5 runs, per problem instance, under different random seeds in Metis where randomization is used to affect improved bisectioning and matching. As expected, the total cost metric, overall, outperforms the maximum cost metric with respect to CPU balancing. The biggest gaps occur in the cases of worm global and BGP. Fig. 7 (bottom) shows CPU balancing performance with respect to completion time which includes synchronization penalty among PCs in distributed simulation stemming from per-epoch barriers. An across-the-board upward shift is accompanied by a decrease in the maximum vs. total cost metric performance gap. Communication cost—processing time expended for sending and receiving of messages across network partitions—is dominated by MPI packing and unpacking operations which are accounted for in both computation and completion time.

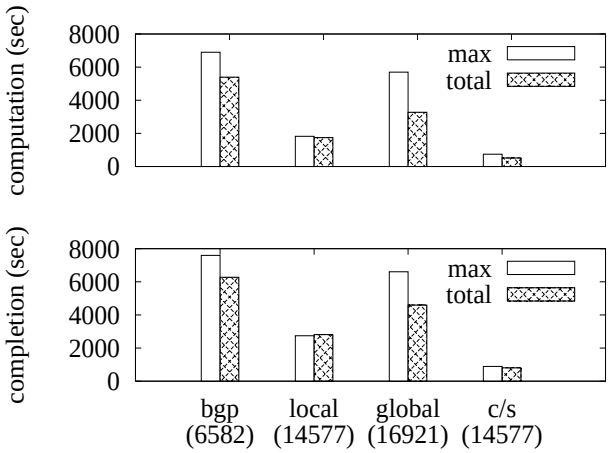


Figure 7. CPU balancing: computation time (top) and completion time (bottom).

5. Simulation Performance Diagnosis

In this section we discuss the causes underlying the memory and CPU balancing trade-off results, and why the maximum cost metric works effectively for memory balancing despite the sum-of-max overestimation problem.

5.1. Effect of Topology and Application

5.1.1. Influence of Power-law Connectivity We compare BGP and worm local benchmark applications for

which the performance gaps between maximum and total cost metrics are small and large, respectively. Fig. 8 (top) shows per-node M_i and C_i load distribution as a function of node rank—nodes are ranked by their degree with rank 1 indicating a node with the largest number of neighbors—for the BGP benchmark application. The abscissa is shown in log-scale to highlight the load values of high-degree nodes. We observe a pronounced skew in the load distribution—both for the maximum and total cost metrics—which stems from traffic concentration at high-degree nodes. Traffic skewness, in turn, is induced by power-law tendencies characteristic of Internet measurement topologies [4].[2] The total cost metric C_i exhibits a greater skew than the maximum metric M_i since the former is a sum over time whereas the latter is the maximum. Fig. 8 (bottom) shows the corresponding plots for worm local which exhibit similar power-law skews.

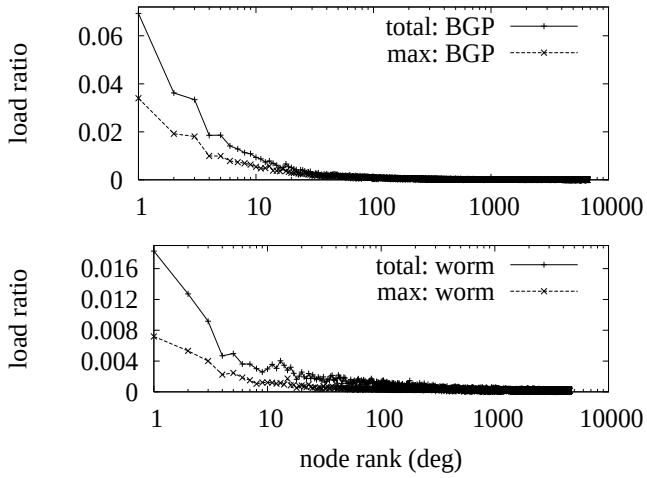


Figure 8. Node load distribution as a function of node rank: BGP (top) and worm local (bottom).

5.1.2. Influence of Application Behavior The key difference between BGP and worm local is shown in Fig. 9 which plots the cumulative node load distribution corresponding to Fig. 8. In BGP, we observe that both total and max increase rapidly initially with C_i climbing a bit higher than M_i. In worm local the initial rate of increase of M_i is significantly slower than that of C_i, almost resembling a linear curve. The sharp increase in cumulative M_i in BGP is caused by a sharp rise in message memory which dominates table memory (i.e., BGP and IP routing tables). In both BGP and worm local, cumulative table memory increases gradually.

[2] Power-law tendencies also exist in router-level topologies [4, 14] although their detailed structure and causality differ.

All else being equal, the more skewed a node load distribution, the harder it is to balance. In both Fig. 8 and 9, total cost metric has a higher node load skew than maximum cost metric. The key difference is that worm local has a significantly bigger skew gap between total and maximum cost metrics as shown in Fig. 9 (bottom) which leads to a commensurately large memory imbalance under total vs. maximum cost metric seen in Fig. 6. The difference in initial ramp-up in message memory can be explained by differences in application behavior of BGP and worm local.

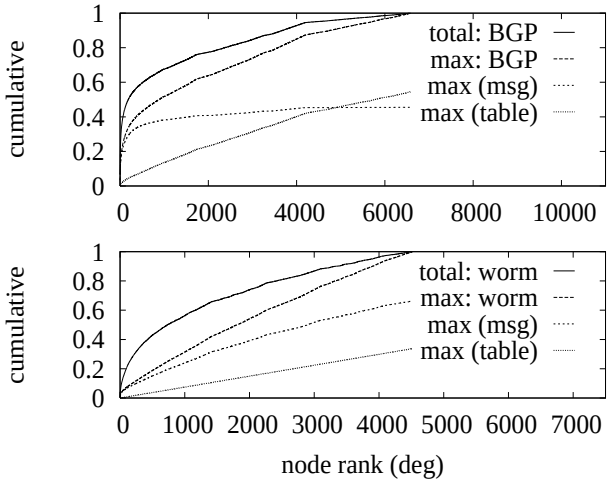


Figure 9. Cumulative node load distribution: BGP (top) and worm local (bottom).

5.2. Measurement Accuracy and Overhead

5.2.1. Effect of Scale In Section 4.1.2 we described the sum-of-max vs. max-of-sum problem where the maximum memory metric can significantly overestimate actual memory load during partitioning. Despite the fact that the total cost metric does not have this problem due to its closure property, the maximum cost metric, overall, outperforms the total cost metric with respect to memory balancing. This is due to two factors.

First, in large-scale network simulation where the number of nodes n is large, the law of large numbers helps mitigate the sum-of-max overestimation problem by averaging out the time instances where individual nodes reach peak memory. Fig. 10 quantifies this effect by showing increasing prowess of sum-of-max in predicting actual memory consumption (i.e., max-of-sum) as problem size is increased from 300 to 3213 for worm global. The number of partitions is held constant at 24 (i.e., 24 machines participate in distributed simulation). To highlight the law of large number effect, we use random graphs of same edge density

as power-law graphs where all nodes have approximately similar degrees. This removes complications introduced by high-degree nodes whose memory peaks are significantly higher than other nodes (the "elephants and mice" feature).

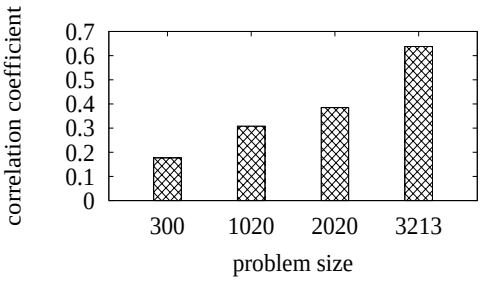


Figure 10. Correlation between sum-of-max and max-of-sum balancing as a function of problem size.

The second factor is the increased role of table memory in large problem instances. In worm local, per-node route table size is fixed (in BGP table sizes are variable), and in the absence of table memory reduction optimization per-node table memory grows linearly in n. Consider Fig. 9 (bottom) which shows that per-node message memory is about twice that of table memory. Although this is true on a per-node basis, when multiple nodes are mapped to a common partition the sum-of-max overestimation problem kicks in which reduces their actual collective message memory footprint. The per-partition make-up of message vs. table memory of Fig. 9 (bottom) is about half-half. This dampens node load skewness during partitioning which is conducive to easier memory balancing.[3]

5.2.2. Memory Leakage and Garbage Collection Accurate monitoring is essential for effective load balancing. As indicated in Section 3.2, memory accounting at the system level is complicated by memory leakage and garbage collection that, in general, are system dependent. Coding based memory leakage can be prevented by deallocating message related memory in accordance with reference message evolution diagrams (e.g., Fig. 2 for TCP).

Fig. 11 quantifies measurement overhead and accuracy with respect to memory consumption monitoring for the BGP simulation of Fig. 3. The bottom two curves show measured memory usage with and without memory overhead needed to maintain per-node memory, computation, and communication load. Overhead is small due to constant per-node space needed of M_i and C_i. The same goes for communication cost which requires constant memory per link—in power-law measurement topologies the number of links is about 2–3 times the number of nodes.

[3]If table memory were dominant, memory balancing would be trivial since all nodes would have approximately equal weight.

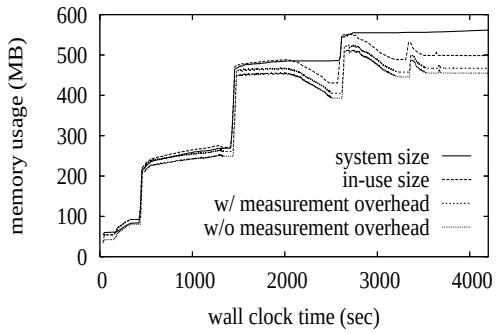


Figure 11. Measurement subsystem memory monitoring overhead and accuracy.

In our version of DaSSF we use the malloc library (part of standard C library) to manage dynamic memory allocation and deallocation by-passing Hoard and QuickMem. In Fig. 11 "in-use" indicates memory used by a user process. The gap between measured memory usage with overhead and in-use represents memory that is not tracked as part of reference message evolution diagrams (i.e., BGP, TCP, and UDP). The gap tends to be small, in this instance, less than 5.2%. "system size" specifies total memory that malloc has allocated. The gap between in-use and system size arises when deallocation calls are made which do not necessarily result in actual garbage collection due to malloc's internal memory pool management. The gap between the two is not small and variable. Since at time instances of maximum memory usage they coincide, impact on memory estimation and balancing is limited. Linux keeps track of memory usage in `/proc` which is almost identical to system size.

6. Joint Memory-CPU Balancing

Sections 4 and 5 studied basic properties of memory balancing, established a trade-off between memory and CPU balancing, and considered their causes. In this section we examine joint memory-CPU balancing.

6.1. Multi-constraint Optimization

Multilevel recursive bisection methods including Metis [10] and Chaco [7] that implement k-way partitioning seek to find heuristic solutions to the NP-hard constrained optimization problem: minimize edge cut subject to node weight balancing. Multilevel recursive bisection heuristics may be extended to include a memory

balance constraint in which case every node i has a 2-dimensional weight vector (M_i, C_i). Metis has support for multi-dimensional node weights and we use this set-up for joint memory-CPU balancing.

6.2. Joint Balancing Performance

In general, a trade-off between two objectives implies that to improve one there has to be a sacrifice of the other. We show that in network simulation this need not be the case. Fig. 12 shows memory balancing performance for the benchmark set-up of Section 4 under joint memory-CPU balancing. We find that joint memory-CPU balancing performs as well as memory-centric load balancing which uses only node weight M_i. Fig. 13 shows computation and completion time results under joint memory-CPU balancing. As with memory balancing, we observe that joint memory-CPU balancing performs as well as CPU-centric balancing that uses node weight C_i only. This appears to be a case of "can have the cake and eat it too."

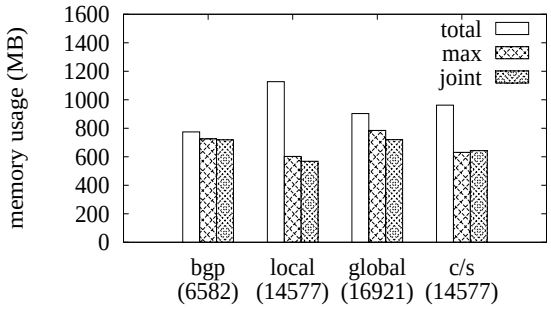


Figure 12. Memory balancing performance under joint max-total cost metric.

6.3. Overcoming Memory-CPU Trade-off

How is it possible for joint memory-CPU using the max-total metric to circumvent the balancing trade-off? The answer lies with network simulation having a tendency to induce strong positive correlation between M_i and C_i. For high-degree nodes, this can be gleaned from Fig. 8 for both BGP and worm local. For worm local the correlation coefficient, corr(M_i, C_i), is 0.94. Individually they face balancing trade-off difficulties as discussed in Section 5. When combined, their individual feasible regions have non-empty intersection due to strong correlation such that a partitioning is found that can balance both memory and CPU well. That this is not a property available to discrete-event simulation in general can be surmised from Fig. 14 which shows memory and CPU balancing performance across 16 machines participating in distributed simulation when

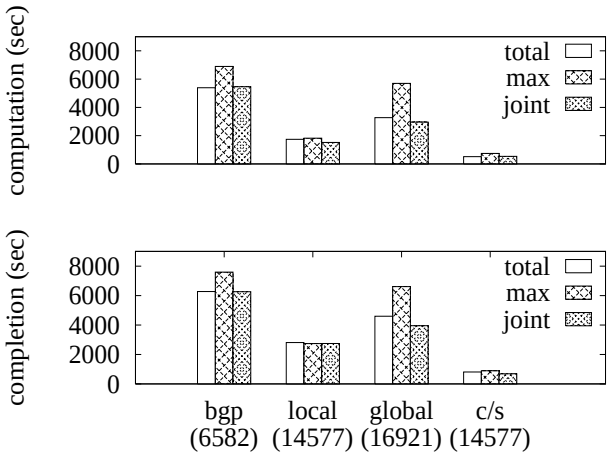


Figure 13. CPU balancing under joint max-total cost metric. computation time (top) and completion time (bottom).

corr(M_i, C_i) is strong—original worm local benchmark—or weak. In the latter, node load skew of M_i in Fig. 8 (bottom) is severely flattened such that corr$(M_i, C_i) = 0.01$. The ordinate of Fig. 14 shows the average and spread of aggregate node weight sum—both memory (M_i) and CPU (C_i)—across the 16 partitions with node weight normalized for comparison. We find that when M_i and C_i are weakly correlated memory and CPU balancing trade-off may not be overcome.

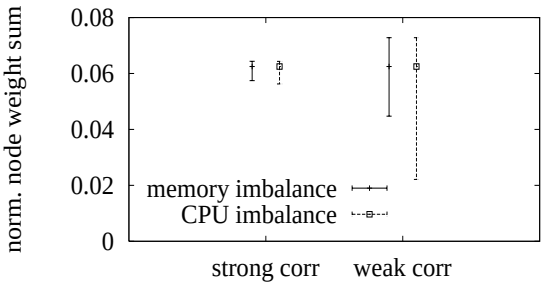


Figure 14. Joint memory-CPU balancing when corr(M_i, C_i) is strong and weak.

7. Conclusion

In this paper we studied the memory balancing problem in large-scale network simulation where a memory-overloaded machine can become a bottleneck of distributed simulation. We identified two issues pertinent to memory

load balancing: (i) sum-of-max vs. max-of-sum problem in memory usage estimation for network partitioning, and (ii) memory vs. CPU balancing trade-off. We advanced diagnostic tools to explain distributed network simulation performance, including why joint memory-CPU balancing in network simulation is feasible and can achieve the best of both worlds. Although our measurement subsystem implementation is DaSSF dependent, our measurement architecture and performance evaluation methodology are applicable to other distributed network simulation platforms including PDNS [20].

Acknowledgment

We would like to thank the reviewers whose comments have helped improve the paper. This research was supported in part by grants from DARPA (AFRL F30602-01-2-0539), ETRI, and NSF (EIA-9972883).

References

[1] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proc. ACM SIGMETRICS '96*, pages 160–169, 1996.

[2] P. Denning. Working sets past and present. *IEEE Trans. on Software Engineering*, 6(1):64–84, 1980.

[3] X. Dimitropoulos and G. Riley. Efficient large-scale BGP simulations. *Computer Networks*, 50(12):2013–2027, 2006.

[4] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *Proc. ACM SIGCOMM '99*, pages 251–262, 1999.

[5] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley. Large-Scale Network Simulation: How Big? How Fast? In *Proc. IEEE MASCOTS '03*, pages 116–123, 2003.

[6] G. Glass and P. Cao. Adaptive page replacement based on memory reference behavior. In *Proc. ACM SIGMETRICS '97*, pages 115–126, 1997.

[7] B. Hendrickson and R. Leland. The Chaco User's Guide, version 2.0. Technical Report, SAND94-2692, Sandia National Laboratories, 1994.

[8] A. Hiromori, H. Yamaguchi, K. Yasumoto, T. Higashino, and K. Taniguchi. Reducing the size of routing tables for large-scale network simulation. In *Proc. IEEE PADS '03*, 2003.

[9] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[10] G. Karypis and V. Kumar. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. Technical Report, Department of Computer Science, University of Minnesota. Available at `http://www.cs.umn.edu/~metis`, 1998.

[11] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[12] S. Lee, J. Leaney, T. O'Neill, and M. Hunter. Performance benchmark of a parallel and distributed network simulator. In *Proc. IEEE PADS '05*, 2005.

[13] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. In *Proc. ACM SIGCOMM '93*, pages 183–193, 1993.

[14] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the Internet's router-level topology. In *Proc. ACM SIGCOMM '04*, pages 3–14, 2004.

[15] J. Liu and D. Nicol. Learning not to share. In *Proc. IEEE PADS '01*, pages 46–55, 2001.

[16] Y. Liu, B. Szymanski, and A. Saifee. Genesis: A scalable distributed system for large-scale parallel network simulation. *Computer Networks*, 50(12):2028–2053, 2006.

[17] R. Mills. *Dynamic adaptation to CPU and memory load in scientific applications*. PhD thesis, The College of William and Mary, 2004.

[18] D. Nicol. Scalability of network simulators revisited. In *Proc. CNDS '03*, 2003.

[19] D. Nicol and J. Liu. Composite synchronization in parallel discrete-event simulation. *IEEE Trans. Parallal and Distributed Systems*, 13(5):433–446, 2002.

[20] G. T. P. NS. pdns 2.1b7a. Available in `http://www.cc.gatech.edu/computing/compass/pdns/`, February 2001.

[21] H. Ohsaki, G. Oscar, and M. Imase. Quasi-dynamic network model partition method for accelerating parallel network simulation. In *Proc. IEEE MASCOTS '06*, pages 255–264, 2006.

[22] E. Page, D. Nicol, O. Balci, R. Fujimoto, P. Fishwick, P. L'Ecuyer, and R. Smith. Panel: Strategic directions in simulation research. In *Proc. 1999 Winter Simulation Conference*, pages 1509–1520, 1999.

[23] SSF. Ssfnet 1.5. Available in `http://www.ssfnet.org/homePage.html`, May 2003.

[24] D. SSF. Dassf 3.1.5. Available in `http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/`, August 2001.

[25] University of Oregon. Oregon Route Views. `http://www.routeviews.org/` and `http://archive.routeviews.org/`.

[26] L. Xiao, S. Chen, and X. Zhang. Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):223–240, 2002.

[27] D. Xu and M. Ammar. BencHMAP: Benchmark-based, hardware and model-aware partitioning for parallel and distributed network simulation. In *Proc. IEEE MASCOTS '04*, pages 455–463, 2004.

[28] D. Xu, G. Riley, M. Ammar, and R. Fujimoto. Enabling large-scale multicast simulation by reducing memory requirements. In *Proc. IEEE PADS '03*, pages 69–76, 2003.

[29] K. Yokum, E. Eade, J. Degesys, D. Becker, J. Chase, and A. Vahdat. Toward scaling network emulation using topology partitioning. In *Proc. IEEE MASCOTS '03*, pages 242–245, 2003.

[30] X. Zhang, Y. Qu, and L. Xiao. Improving distributed workload performance by sharing both cpu and memory resources. In *Proc. IEEE ICDCS '00*, pages 233–241, 2000.