Quasi-Dynamic Network Model Partition Method for Accelerating Parallel Network Simulation

Hiroyuki Ohsaki, Gomez Oscar and Makoto Imase
Graduate School of Information Science and Technology
Osaka University, Japan
E-mail: {oosaki,o-gomez,imase}@ist.osaka-u.ac.jp

***Abstract*— In this paper, we propose a network model partition method called *QD-PART (Quasi-Dynamic network model PARTition method)* for accelerating parallel network simulation. The key of QD-PART is to utilize the fact that a network simulation is typically repeated several times with the same parameter set for estimating the confidence interval of steady state measures. QD-PART gradually optimizes partition of a network model based on past simulation results such as the total simulation time, CPU usage of computing resources, and traffic intensity (i.e., the number of packets transmitted) of each link. At the end of each parallel simulation run, QD-PART re-partitions the network model based on such information aiming at minimizing communication overhead among computing resources and balancing load of sub-network models executed on computing resources. Through several experiments using a parallel-distributed network simulator, we show how parallel network simulation can be accelerated using QD-PART by gradually improving the network model partition.**

keywords: Parallel Simulation, Large-Scale Network, Network Model Partition Method

I. Introduction

In recent years, demand for a technique to evaluate the performance of large-scale networks has heightened along with the increasing size and complexity of the Internet [1], [2]. The Internet today is a best-effort network, and communication quality between end nodes is in no way guaranteed. Of course, robustness to some extent has been achieved by dynamic routing mechanisms such as OSPF and BGP even in the current Internet. However, the Internet itself is indispensable as society's communication infrastructure, so a technique to evaluate the performance of large-scale networks is in strong demand to ensure the reliability, safety, and robustness of networks, to allow future network expandability and design, and to assess the impact of terrorism and natural disasters.

However, conventional techniques to evaluate the performance of a network such as mathematical analysis techniques and simulation techniques are directed toward relatively small-scale networks. As an example, queuing theory [3] has been widely used in performance evaluation of conventional computer networks, but it is not readily applied to performance evaluation of the large-scale and complex Internet. When rigorously analyzing the performance of a network using queuing theory, the number of states for analysis increases exponentially according to the increase in the number of nodes connected to the network.

Simulation techniques, as opposed to numerical analysis techniques, allow performance evaluation of complex networks [4]. Performance evaluation of medium-scale networks in particular has become possible through the increasing speeds and capacities of computers in recent years. However, communication protocols for the Internet are extremely complex, so massive computer resources are required for simulation of networks, and simulation of large-scale networks is still difficult. The majority of network simulators widely used today simulate behavior at a packet level, so they use an event-driven architecture. A technique for faster speed of network simulators operating on a single computer has also been proposed [5], although a different approach is needed to simulate a large-scale network.

Research with regard to parallel simulations as a technology to allow simulation of large-scale networks has been conducted in recent years [6]–[8]. Construction of relatively inexpensive cluster computers has become easier through the faster speeds and lower prices of desktop computers and the spread of high-speed network interfaces such as Gigabit Ethernet. In addition, Grid computing using a wide-area network to integrate computer resources around the world has also attracted attention.

However, the majority of network simulators have an event-driven architecture, so parallelization of network simulators is difficult. The execution speed of parallel simulation is heavily affected by partitioning of a network model; i.e., the total simulation time depends on how a network model is divided into multiple sub-network models, each of which will be assigned to a different computing resource.

In this paper, we propose a network model partition method called *QD-PART (Quasi-Dynamic network model PARTition method)* for accelerating parallel network simulation. The key of QD-PART is to utilize the fact that a network simulation is typically repeated several times with the same parameter set for estimating the confidence interval of steady state measures. Namely, QD-PART gradually optimizes partition of a network model based on past simulation results such as the total simulation time, CPU usage of computing resources, and traffic intensity (i.e., the number of packets transmitted) of each link. At the end of each parallel simulation run, QD-PART re-partitions the network model based on such information aiming at minimizing communication overhead among computing resources and balancing load of sub-network models executed on computing resources.

The structure of this paper is as follows. First, Section II describes related work regarding network model partition methods. Section III explains QD-PART, our network model partition algorithm. Section IV illustrates an example partition of a network model using QD-PART. In Section V, we present several experimental results of QD-PART, showing how parallel network simulation can be accelerated by gradually improving the network model partition. Finally, Section VI concludes this paper and discusses future research topics.

II. Related Work

In the literature, there have been several studies on a network model partition method for accelerating parallel network simulation. In [9], the authors have proposed a network model partition method for cluster computers. The key of the proposed partition method is to view a network model as a graph, and divides it into several sub-graphs using a simple min-cut algorithm [10]. The weight of each edge is assigned as the traffic intensity of the corresponding link. The traffic intensity is estimated using TCP steady-state analysis. Although the proposed partition method tries to minimize communication overhead among computing resources, the partition result might not be good since it simply uses rough estimation of the traffic intensity.

In [11], the authors have proposed a benchmark-based network model partition method for cluster computers. The proposed partition algorithm utilizes a graph partitioning tool called METIS [12] for dividing a network model into several sub-network models. Similar to [9], the proposed partition algorithm utilizes simple estimation of traffic intensity of each link, so that the partition result might not be good.

III. Quasi-Dynamic Network Model Partition Method

In this section, we first explain the basic idea of our network model partition method called QD-PART (Quasi-Dynamic network model PARTition method), followed by its algorithm description.

A. Basic Idea

The key of QD-PART is to utilize the fact that a network simulation is typically repeated multiple times with the same parameter set for estimating the confidence interval of steady state measures.

In many network simulation studies, performance analysts generally need to measure statistics, such as throughput, mean and variance of delay, and packet loss rate, in steady state or transient state [13]. In particular, statistics in steady state are generally of great concern. Since most packet-level simulation uses a Monte Carlo simulation technique and the simulation time is finite, measured statistics in steady state cannot be a precise estimate of real statistics. Hence, for validating accuracy of measured statistics, the confidence interval of steady state measures are typically calculated by sampling independent steady state measures.

There are several techniques for estimating the confidence interval of steady state measures, such as the independent replications method, the batch means method, the autoregressive method, the spectrum analysis, and the regenerative method [13]. In theory, advanced techniques, such as the autoregressive method and the regenerative method, are effective for estimating the confidence interval since the simulation time required can be much smaller than that of other estimation techniques. However, their application are not straightforward since, for instance, choosing regenerating point needs careful consideration. In practice, the simplest technique of the independent replications method has been widely used in many network simulation studies. In the independent replications method, for obtaining samples of steady state measures, a network simulation is repeated multiple times with the same parameter set but with a different seed for the random number generator.

In QD-PART, we utilize this — a network simulation is typically repeated several times with the same parameter set for estimating the confidence interval of steady state measures. Namely, QD-PART gradually optimizes partition of a network model based on past simulation results such as the total simulation time, CPU usage of computing resources, and traffic intensity (i.e., the number of packets transmitted) of each link. At the end of each parallel simulation run, QD-PART repartitions the network model based on such information aiming at minimizing communication overhead among computing resources and balancing load of sub-network models executed on computing resources.

B. Algorithm

First, we define several terms used throughout this paper (see Fig. 1); *node* is either a host or router in simulation, *link* is a transmission link connecting nodes in simulation, and *flow* is a stream of packets sent from a source node to a destination node. A set of nodes, links, and flows with their attributes (i.e., system and control parameters) to be simulated is called *network model*. A network model partition method splits a network model into two or more *sub-network models*. Sub-network models are assigned to *computing resources*, each of which runs a parallel network simulator and performs simulation by coordinating with other computing resources. A network device between computing resources is called *networking resource*.

In QD-PART, a network model is first represented as an undirected graph $G = (V, E)$, where $V = \{1, 2, \ldots\}$ and $E = \{1, 2, \ldots\}$. Vertices in V correspond to nodes (hosts or router), and edges in E correspond to links between nodes. The weight of edge (i, j) is denoted by $w(i, j)$. Furthermore, the total number of partitions of the network model (i.e., the number of sub-network models) is N. Also, let T_k be the total simulation time of the k-th parallel simulation, $P_k(n)$ be the CPU usage of the computing resource n in the k-th parallel simulation, and $l_k(i, j)$ be the traffic intensity of a link corresponding to edge (i, j).

The algorithm of QD-PART is as follows.

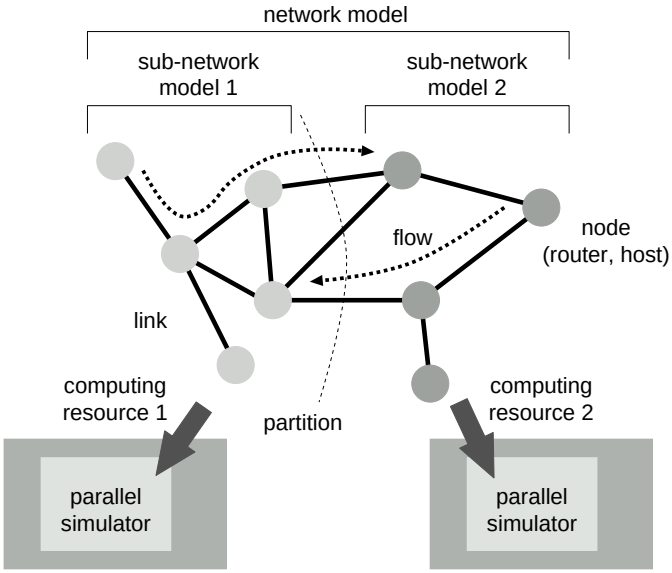

Fig. 1: Network model partition overview

1) *Make initial partition*
Before running the first simulation, we have no information on the traffic intensity of each link. Hence, the initial partition is determined by assuming that all edges have the same traffic intensity. Namely, weight $w(i,j)$ of edge (i,j) is defined simply as

$$w(i,j) \quad = \quad \frac{1}{\tau(i,j)^{\alpha}}, \tag{1}$$

where $\tau(i,j)$ is the propagation delay of a link corresponding to edge (i,j), and $\alpha(>0)$ is a control parameter determining sensitivity of the link propagation delay on a network model partition.
A graph partition algorithm [14] is applied to G, which results in N sub-graphs $G_1,\ldots,G_N$. These N sub-network models are assigned to N computing resources, and parallel network simulation is performed. At the end of simulation, several results such as the total simulation time T_1, CPU usage of each computing resource $P_1(n)$, and traffic intensity of each link $l_1(i,j)$ are obtained.
2) *Make second partition based on measured traffic intensity*
Since we now have several statistics of the first parallel simulation such as the total simulation time, the CPU usage of computing resources, and the traffic intensity of each link, a better partition than the initial one can be determined.
It is known that the communication overhead among computing resources is almost proportional to the number of simulation events exchanged between computing resources. In a packet-level network simulation, almost all simulation events are regarding packet processing. Hence, the communication overhead among computing resources should be almost proportional to the number of packets exchanged between nodes in different partitions. The second partition is therefore determined by taking account of the traffic intensity $l_1(i,j)$. More specifically, weight $w(i,j)$ of edge (i,j) is redefined as

$$w(i,j) \quad = \quad \frac{l_1(i,j)}{\tau(i,j)^{\alpha}}. \tag{2}$$

Similar to the previous step, a graph partition algorithm [14] is applied to G with redefined edge weights, resulting in N sub-graphs $G_1,\ldots,G_N$. These N sub-network models are again assigned to N computing resources, and the second parallel network simulation is performed. At the end of simulation, several statistics such as the total simulation time T_2, CPU usage of a computing resource $P_2(n)$, and the traffic intensity of each link $l_2(i,j)$ are obtained.
3) *Improve partition using measured CPU usage*
In subsequent parallel simulations, a network model partition $G_1,\ldots,G_N$ is gradually improved based on the measured statistics of past simulations. Namely, in the k-th parallel simulation, a network model partition is determined by modifying the partition in the $(k-1)$-th parallel simulation.
The idea for improving the previous partition is to move a node from a more loaded computing resource to a less loaded one. For this purpose, CPU usage of computing resources in the previous simulation, $P_{k-1}(n)$, are examined, and the most loaded computing resource (i.e., one with the largest $P_{k-1}(n)$) is identified. Let G_n be the most loaded sub-network model, and $V'(G_n)$ be a subset of vertices in G_n, which are connected to one or more nodes in other sub-network models. Namely, a node i in $V'(G_n)$ and a node j in $V(G-G_n)$ are connected by edge (i,j). A node in $V'(G_n)$ is randomly chosen, and it is moved to the other sub-network model. By moving a node from a more loaded computing resource to less loaded one, we expect that load balancing among computing resources is improved, possibly leading to a shorter total simulation time.
N sub-network models $G_1,\ldots,G_N$ are assigned to N computing resources, and the k-th parallel network simulation is performed. At the end of simulation, several statistics such as the total simulation time T_k, CPU usage of a computing resource $P_k(n)$, and the traffic intensity of each link $l_k(i,j)$ are obtained.
If the total simulation time T_k is shorter than the previous parallel simulation T_{k-1}, step (3) is repeated. Otherwise, the network model partition is fixed at that of the previous simulation, which results in the least total simulation time.

IV. Partition Example

In this section, an example partition of a network model with $N=2$ (i.e., partition into two sub-network models) using QD-PART is presented.

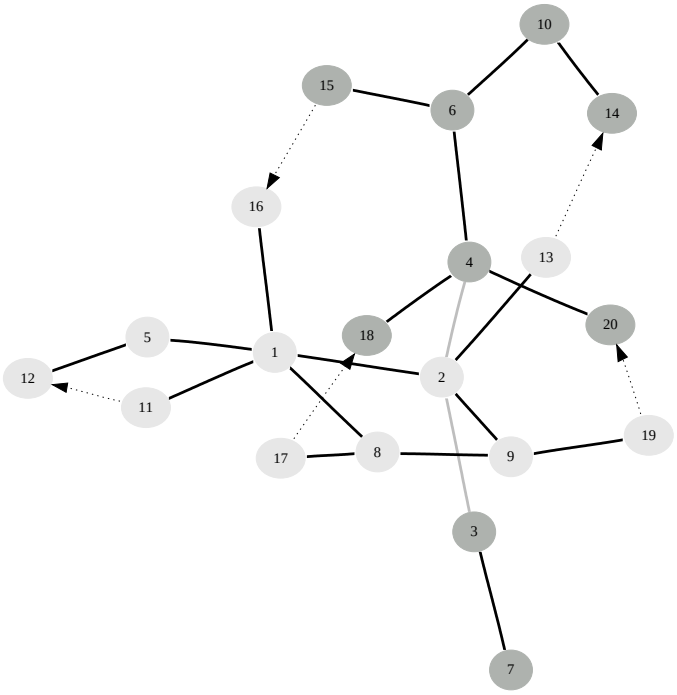

Fig. 2: Initial partition is determined by assuming that all edges have the same traffic intensity. Dark-gray nodes are in G_1, and light-gray nodes are in G_2. Gray lines represent crossing links.

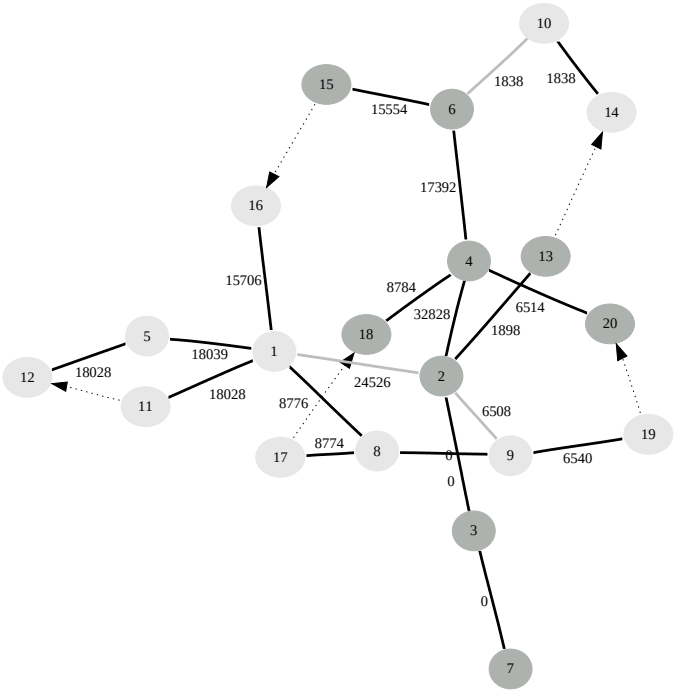

Fig. 3: The second partition is determined by taking account of the traffic intensity measured in the first simulation. The number at an edge is the traffic intensity.

Figure 2 shows an example network model, which is composed of 20 nodes and 5 flows. In the figure, circles, thick lines, and dotted lines with arrow heads represent nodes, links, and flows, respectively.

1) *Make initial partition*
   The initial partition is determined by assuming that all edges have the same traffic intensity. By applying a graph partition algorithm in [14], two sub-network models, G_1 and G_2, are obtained.
   Figure 2 shows the initial partition with $N = 2$. In Fig. 2, dark-gray nodes represent nodes in G_1, and light-gray nodes represent nodes in G_2. Gray lines represent crossing links (i.e., links between two sub-network models of G_1 and G_2).
   Two sub-network models, G_1 and G_2, are assigned to two computing resources, parallel network simulation is performed, and several statistics are measured.
2) *Make second partition based on measured traffic intensity*
   The second partition is determined by taking account of the traffic intensity measured in the first simulation. Figure 3 shows the result of the second partition. In this figure, the number shown at each edge represents the traffic intensity (i.e., the number of packets transmitted) of the corresponding link. With edge weights defined by Eq. (2), a graph partition algorithm [14] is applied to G, resulting in two sub-network models G_1 and G_2. Similar to Fig. 2, a network model partition is represented by the color of nodes and links.
   In the second partition, the network model is split at edges (1, 2), (2, 9), and (6, 10) since these edges have relatively small weights (i.e., relatively small traffic intensity).
3) *Improve partition using measured CPU usage*
   A network model partition is gradually improved based on the measured statistics of past simulations. Suppose that in the second simulation (Fig. 3), the computing resource processing G_2 is more loaded than that processing G_1 (i.e., $P_2(1) < P_2(2)$). Hence, in this step, a node in G_2 is moved to G_1. From a subset of vertices in G_2 connected to G_1, $V'(G_2) = \{1, 9, 10\}$, a node is randomly chosen. In Fig. 3, the node 1 is chosen, and moved to G_1. The resulting partition is shown in Fig. 4. Two refined sub-network models, G_1 and G_2, are assigned to two computing resources, parallel network simulation is performed. This step is repeated until moving node does not contribute to shorten the total simulation time.

V. Experiments

In this section, we evaluate the performance of QD-PART through several experiments using a parallel-distributed network simulator.

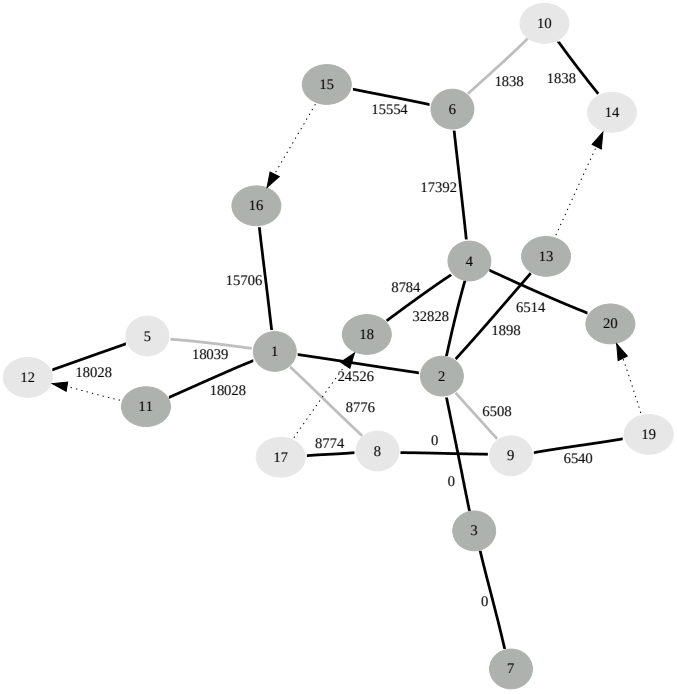

Fig. 4: A network model partition is gradually improved based on the measured statistics of past simulations. Since the computing resource processing G_2 is more loaded (i.e., $P_2(1) < P_2(2)$), the node 1 in G_2 is moved to G_1.

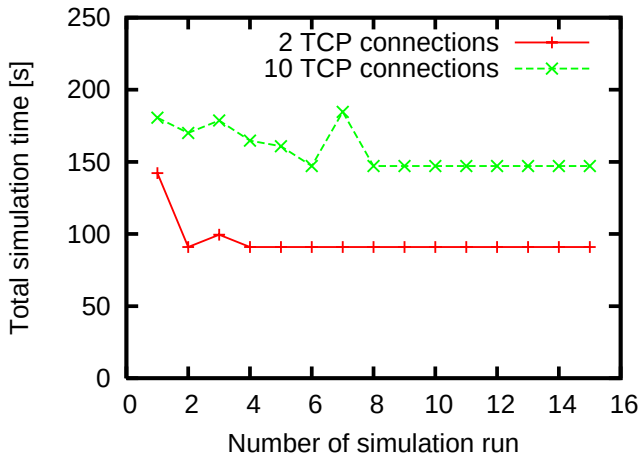

Fig. 5: Total simulation time vs. number of simulation run for two TCP connections and ten TCP connections in the homogeneous case.

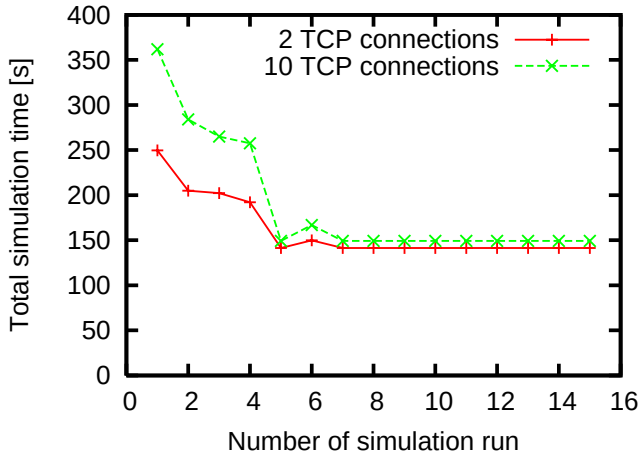

Fig. 6: Total simulation time vs. number of simulation run for two TCP connections and ten TCP connections in the heterogeneous case.

In our experiments, we use the following devices and software: two identical computers (Intel Xeon 2.4GHz processor with 1,024 Mbyte memory running Linux 2.4.30 and PDNS [15] version 2.27-v1a) as computing resources and a gigabit Ethernet switch as a networking resource.

We have implemented a network model partition software, which translates a NS2 [16] simulation script into multiple PDNS simulation scripts using QD-PART method.

A network model is generated as a random graph with 20 nodes. We consider two cases: homogeneous case and heterogeneous case. In the homogeneous case, the bandwidth and the propagation delay of each link is equally set to 1 [Mbit/s] and 1 [ms], respectively. In the heterogeneous case, the bandwidth and the propagation delay of each link is randomly set to values between 0 and 1 [Mbit/s], and between 0 and 1 [ms], respectively. Either two or ten TCP connections are generated between randomly-chosen nodes. All routers are DropTail with 100 [packet] buffer.

In what follows, due to space limitation, only experiment results with $\alpha = 0$ are presented.

Figure 5 shows the total simulation time at each simulation run for two TCP connections and ten TCP connections in the homogeneous case. This figure shows that using QD-PART the total simulation time is gradually reduced. One can find from this figure that the total simulation time is converged by the 8-th simulation. Regardless of the number of TCP connections, the total simulation time is significantly reduced from the initial partition.

Figure 6 shows the total simulation time at each simulation run for two TCP connections and ten TCP connections in the heterogeneous case. This figure shows QD-PART is quite effective particularly in the heterogeneous case. Also this figure shows that the total simulation time is converged at the 7-th simulation.

The reason of QD-PART's effectiveness in the heterogeneous case can be explained by heterogeneity in the bandwidth and the propagation delay of links. If the bandwidth and/or

the propagation delay of all links are almost the same, existing graph partition algorithms such as [14] are effective. However, if the bandwidth and/or the propagation delay of links are quite different with each other, those graph partition algorithms are not good for finding a partition. This is because the traffic intensity is difficult to estimate and computational burden of a sub-network model is difficult to predict. Since QD-PART uses the measured statistics of past simulation, it can gradually improve the partition, resulting in a shorter total simulation time.

VI. Conclusion

In this paper, we propose a network model partition method called QD-PART for accelerating parallel network simulation. QD-PART utilizes the fact that a network simulation is typically repeated several times with the same parameter set for estimating the confidence interval of steady state measures. At the end of each parallel simulation run, QD-PART re-partitions the network model based on past simulation results such as the total simulation time, CPU usage of computing resources, and traffic intensity (i.e., the number of packets transmitted) of each link. Through several experiments using the PDNS network simulator, we have shown that QD-PART significantly reduces the total simulation time, in particular, when the network model is heterogeneous.

We believe QD-PART have made an important step toward realizing parallel simulation of large-scale networks, but there remains several problems to be solved. Our future work includes through performance evaluation of QD-PART (e.g., with other types of network models and more computing resources) and tuning of its control parameter α. Also it is important to extend QD-PART to support heterogeneous computing resources and networking resources such as Grid computing.

References

[1] *Workshop on New Visions for Large-Scale Networks: Research and Applications*. Large Scale Networking (LSN) Coordinating Group of the Interagency Working Group (IWG) for Information Technology Research and Development (IT R&D), Mar. 2001, available at http://www.nitrd.gov/iwg/lsn/lsn-workshop-12mar01/index.html.

[2] V. Paxson and S. Floyd, "Why we don't know how to simulate the Internet," in *Proceedings of the 1997 Winter Simulation Conference*, Dec. 1997, pp. 1037–1044.

[3] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, New Jersey: Prentice-Hall, 1987.

[4] A. M. Law and M. G. McComas, "Simulation software for communications networks: the state of the art," *IEEE Communications Magazine*, vol. 32, pp. 44–50, Mar. 1994.

[5] V. S. Frost, W. W. Larue, and K. S. Shanmugan, "Efficient techniques for the simulation of computer communications networks," *IEEE Journal of Selected Areas in Comuunications*, vol. 6, no. 1, pp. 146–157, Jan. 1988.

[6] H. T. Mouftah and R. P. Sturgeon, "Distributed discrete event simulation for communications networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 9, pp. 1723–1734, Dec. 1990.

[7] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, "A generic framework for parallelization of network simulations," in *Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Oct. 1999, pp. 128–135.

[8] S. Bhatt, R. Fujimoto, A. Ogielski, and K. Perumalla, "Paralell simulation techniques for large scale networks," *IEEE Communications Magazine*, vol. 38, no. 8, pp. 42–47, Aug. 1998.

[9] H. Ohsaki, S. Yoshida, and M. Imase, "On network model division method based on link-to-link traffic intensity for accelarating parallel distributed simulation," in *Proceedings of 4th International Conference on Networking (ICN' 05)*, Apr. 2005, pp. 749–757.

[10] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM*, vol. 44, no. 4, pp. 585–591, July 1997.

[11] D. Xu and M. Ammar, "BencHMAP: Benchmark-based, hardware and model-aware partitioning for parallel and distributed network simulation," in *Proceedings of 12th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, 2004, pp. 455–463.

[12] G. Karypis and V. Kumar, "METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," http://www-users.cs.umn.edu/~karypis/metis/.

[13] R. Jain, *The Art of Computer Systems Performance Analysis*. New York: Wiley-Interscience, Apr. 1991.

[14] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, Aug. 1998.

[15] PADS (Parallel and Distributed Simulation) Research Group, "PDNS - Parallel/Distributed NS," http://www.cc.gatech.edu/computing/compass/pdns/.

[16] "The network simulator – ns2," available at http://www.isi.edu/nsnam/ns/.