Traffic-based Load Balance for Scalable Network Emulation

Xin Liu and Andrew A. Chien
Computer Science and Engineering Department
University of California, San Diego
{xinliu,achien}@cs.ucsd.edu

Abstract

Load balance is critical to achieving scalability for large network emulation studies, which are of compelling interest for emerging Grid, Peer to Peer, and other distributed applications and middleware. Achieving load balance in emulation is difficult because of irregular network structure and unpredictable network traffic. We formulate load balance as a graph partitioning problem and apply classical graph partitioning algorithms to it. The primary challenge in this approach is how to extract useful information from the network emulation and present it to the graph partitioning algorithms in a way that reflects the load balance requirement in the original emulation problem. Using a large-scale network emulation system called MaSSF, we explore three approaches for partitioning, based on purely static topology information (TOP), combining topology and application placement information (PLACE), and combining topology and application profile data (PROFILE). These studies show that exploiting static topology and application placement information can achieve reasonable load balance, but a profile-based approach further improves load balance for even large scale network emulation. In our experiments, PROFILE improves load balance by 50% to 66% and emulation time is reduced up to 50% compared to purely static topology-based approaches.

1. Introduction

Historically, network simulations/emulations have been used extensively to explore the behavior of network protocols [1-4]. Because of the difficulty of modeling application behavior in detail, most of these simulations use simple application models to exercise the protocols and networks. However, with the advent of large-numbers of applications which tightly couple the use of compute, storage, and networks, techniques to study these resources together are emerging. In particular, large-scale network emulation is an important technique for studying the dynamic behavior of networks, network protocols, and emerging class of distributed applications, including Peer-to-Peer [5] and Grid applications [6] – where the network is an important contributor to application performance, applications generate large amounts of network traffic, and overall application performance is critical. A wide variety of simulation systems have been built to model network behavior based on discrete event simulation [7-10].

The MicroGrid [11], an emulation tool built by our group at UCSD to study the dynamic behavior of Grid applications, enables the execution of complete Grid applications (or reduced size models thereof) on a set of virtual grid resources. To study large applications and large network, compute, and storage resources with high fidelity, it is necessary to use scalable parallel machines. Thus, we have designed the MicroGrid to support efficient parallel emulation of distributed/grid application and resources (including compute and network). By harnessing scalable compute resources, the MicroGrid system and user applications together are themselves an interesting distributed application. Here we consider a key problem for scaling the MicroGrid, the load balance of network emulation itself.

The load balance problem has received much attention in parallel and distributed applications because good solutions are critical to achieving speedups. For network emulation, load balance is challenging because the networks generally have irregular structure, and traffic. As a result, each virtual network entity (router, etc.) poses an unpredictable load. To date, most of the existing network emulation projects [7-10] do not provides systematic techniques to achieve load balance (and therefore good scaling). A number of these projects use simple hierarchical graph partitioners [12], others use randomized clustering techniques based on network topologies [10] which have not been demonstrated to give broadly robust results. The majority of these research projects provide no automated solution, requiring users to manually partition the network. Manual or simple heuristic approaches are unsuitable for large-scale network emulations of thousands of network entities. Netbed [10] automates the solution of a closely-related but distinct

SC'03, November 15-21, 2003, Phoenix, Arizona, USA
Copyright 2003 ACM 1-58113-695-1/03/0011…$5.00

problem, mapping virtual networks to a collection of physical switches, routers, links, and compute nodes.

In this paper, we formulate the load balance problem as a graph partitioning problem and apply classical graph partition algorithms[13-17] to solve it. Our solutions are based on existing graph partition algorithms, and we focus on how to use these algorithms effectively. The challenge here is how to extract useful information from the network emulation and represent it to the graph partitioners in a way that represents the load balance requirement in the original emulation problem. We address this problem by means of three approaches using a range of static and dynamic network information, namely, Topology-Based approach (TOP), Application Placement-Based approach (PLACE), and Profile-Based approach (PROFILE). Based on a large-scale network emulation system we have built in MicroGrid, called MaSSF, we evaluate each of these three approaches for partitioning. These studies show that static topology and application placement information can be used to achieve good load balance, moreover, profile-based approach further improves the achieved load balance for even larger scale network emulations. The specific contributions of this paper include:

- formulating the emulation load balance as a graph partitioning problem, and then applying multi-objective partitioning algorithms to it,
- designing a metric based on network topology and application placement information to estimate total network traffic,
- developing a scheme to profile network traffic efficiently in the emulator and then use the profile data to estimate network traffic,
- demonstrating that compared to topology-based techniques (TOP), adding application placement information (PLACE) improves load-balance significantly, but adding profile information (PROFILE) enables improvements of 50-66%, and
- demonstrating that PROFILE also delivers up to 50% reduction in network emulation time (speeds it up).

The remainder of the paper is organized as follows. Section 2 describes the load balance problem in detail, framing the partitioning and mapping problem and presenting the classical graph partitioning algorithms we employ as tools. Section 3 provides three approaches for solving the problem, describing the pros and cons of each. In Section 4, we evaluate the approaches, using a range of topologies and traffic workloads. The results are discussed, along with related work in Section 5, and finally Section 6 summarizes our contribution and points out some future directions for research.

2. Problem Description

2.1 Elements of the Network Mapping Problem

In distributed network emulation, the target virtual network can be viewed as a graph, in which hosts and routers are viewed as graph nodes and network links are taken as graph edges. The virtual network is emulated by a collection of physical nodes (called ***simulation engine***) connected with high speed local network. Usually, a physical node is in charge of modeling a subset of the virtual network, and a key problem is how to partition the virtual network and assign them to the physical emulation nodes with balanced load. This problem is called *the network mapping problem*. This is a demanding problem because the workload on each physical node depends on the virtual node configurations and network traffic in that subset of virtual network. To achieve the optimal load balance even if the traffic were known is an NP-Complete problem, and in practice, a network mapping problem can be naturally modeled as a graph partitioning problem and solved with the classical graph partitioning algorithms.

2.2 Modeling Network Mapping as a Graph Partitioning Problem

Typical graph partitioning algorithms generally solves single objective partition problems such as:

Given an input graph G = (V, E) with weighted vertices and edges, we want to partition it into k parts such that,
- each part has roughly the same number of vertex weight ***(constraint)***
*- the edge-cut (*the number of edges) *that straddles partitions is minimized* ***(objective)***

By setting the vertex and edge weights appropriately, mapping an emulated network to a set of physical emulation resources can be modeled as a graph partitioning problem and solved using a generic graph partitioning algorithm.

As a well studied problem, we expect that any high quality graph partitioning package (in this case METIS[18]) should produce results comparable to other graph packages. So our challenge is how to apply the graph partitioning algorithm in METIS to solve the mapping problem by defining the suitable input graph G, constraint conditions, and optimization objectives for the graph partitioning algorithm. Our choices are discussed in the following subsections.

2.2.1 Input Graph

The input graph G is defined by two categories of parameters: network structure and traffic information. The network structure includes detailed network topology, link latency, and link bandwidth. In MaSSF, this information is stored in the network description file and can be easily translated to a vertex and adjacent edge graph. Network traffic information is used to define edge weights in the graph, and it may also affect vertex weights. In general, network traffic information includes background traffic and foreground applications traffic, derived from trace, model, or even live applications. How we approximate and model the expected traffic for the emulation is the distinguishing key characteristic of our three different load balance approaches. We will further focus on how to get those information in Section 3.

2.2.2 Constraints

In a graph partitioning problem, the constraint is the vertex weight to be balanced among multiple vertices. In the network mapping problem, the vertex weight can be defined as weighted sum of computation and memory requirement on each simulation engine node. In the MaSSF implementation, the computation requirement mainly comes from the logic for packet processing, which depends on network connection, routes, and traffic intensity. It is calculated based on the maximal bipartition flow of all traffic flowing through a network node. The memory requirement is mainly based on the routing table size. The routing table size is in the order of $O(n^2)$, where n is the number of routers in an AS (Autonomous System). We also use multiple constraints to balance different kinds of vertex weights together.

2.2.3 Objectives

In a graph partitioning problem, the objective is the edge-cut to be minimized. In the network mapping problem, the optimization can use two objectives. The first one is to maximize link latency across partitions. This can reduce the frequency of synchronization among simulation engines and maximize concurrency in the emulation, which is very important for scalability for large scale emulation. This feature is an attribute of our MaSSF system and all other network simulators based on conservative discrete event simulation engines.

The second objective is to minimize the communication of simulation events across simulation engine nodes. It is expensive to transfer a simulation event across physical nodes both in terms of computation overhead and communication latency. Also, the physical network of the simulation engine nodes is often a performance bottleneck for the whole emulation, hence, it is important to minimize this communication. With detailed traffic information, we can estimate the number of simulation events on each single link and use it to calculate the edge weight. How to get the traffic information is the major topic of Section 3.

2.3 Multi-Objective Graph Partitioning

In last subsection, we described two objectives. Both objectives are important and sometimes in opposition.

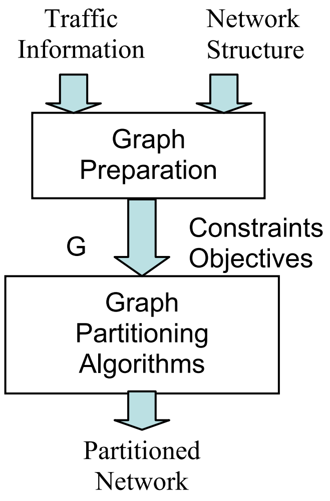

Figure 1. Process of Network Mapping

So we must figure out how to set edge weights to represent the requirement of both objectives, and still provide the user predictable control on tradeoff. A simple combination-based approach (e.g. simply add two weights to a single weight) does not make sense. Applying the algorithm presented in [18], we can combine two dissimilar weights in a predictable way and use the available single objective METIS partitioning package. This algorithm is based on the intuitive notion of what constitutes a good multi-objective partition. That is, a good solution should be close to the optimization solution for each single objective. Applying this approach on our network mapping problem, we get the following algorithm:

1) Apply the single objective algorithm for maximal link latency across partitions, get the optimization edge-cut $C_{latency}$.
2) Apply the single objective algorithm for minimal network traffic across partitions, get the optimization edge-cut $C_{bandwidth}$.
3) Assign each edge weight to

$$w_{conbined} = p\frac{w_{latency}}{C_{latency}} + (1-p)\frac{w_{bandwidth}}{C_{bandwidth}}$$

, where p is the user controllable weight of the latency objective.

4) Apply the single objective algorithm with the new normalized edge weights.

In summary, the mapping process can be modeled as shown in Figure 1. First, it takes the network structure and traffic information as input, creates a graph G, and builds objectives and constraints of graph partitioning algorithms. Then it applies partitioning algorithms to get the partitioned network. The partitioned network incapacitates the mapping of emulated network nodes to physical resources. We may have different abstraction of network mapping problems and use different constraints and objectives in the graph partitioning algorithm, however, we believe what we present above is straightforward and should have reasonable results with small overhead. The problem left is how to collect and use the traffic information, which will be discussed in the following section.

3. Traffic Based Network Mapping

We explore three different approaches for network mapping. These approaches vary how network topology, background traffic, and application traffic are represented and used in the partition. The more accurately an approach predicts the actual simulation work (i.e. network traffic), the better partitioning, and thereby better load balance are expected. However, there are tradeoffs between the specificity of the information used and the generality of the partition produced.

3.1 Network Topology-Based Mapping

Our first approach only considers the virtual network topology, link bandwidth, and latency. In this approach, TOP, each virtual node is weighted with the total bandwidth in and out of it. The optimization objective is to maximize the link latency between simulation engine nodes, as discussed in Section 2.2.3. This maximizes decoupling, supporting efficient parallel emulation.

This basic approach is simple and fast, therefore, it forms a performance baseline for our experiments. It should work well for well-engineered networks with evenly distributed traffic. In such networks, the link bandwidth usually determines the routes that are placed over the links, and since networks are typically engineered to match the demand, link bandwidth is closely related to real traffic. For example, this model is expected to be effective when we want to study the web traffic on Internet, which is composed of lots of small web browsing flows.

3.2 Application Placement-Based Mapping

To achieve a better network mapping, we need precise traffic information. The second approach is based on the observation that emulated network traffic typically consists of a background and a foreground load. Foreground traffic is created by the target application that a user wants to study, and background traffic is used to provide realistic network conditions. We estimate both traffic loads separately, then combine them to estimate the aggregated traffic data for better network mapping. We call this approach PLACE.

The background traffic is generated using simple traffic models based on the network topology, and can be explicitly controlled by the user of the network emulator. In this case, it is reasonable that all traffic generators can provide some prediction of their generated traffic load, for example, specifying the average traffic bandwidth between two endpoints. Because the background traffic represents an aggregate of traffic, such a gross characterization can be reasonably accurate.

The foreground load is typically the live traffic from a small set of application programs. Unlike background traffic prediction, it is difficult for users to predict the traffic of the real application. First, the live traffic has complex dynamic behavior that is hard to model (that is why we need a network emulator to study it). Second, users may not have the required knowledge to describe this information (lacking either application knowledge or the computer systems knowledge). As an approximation, we determine the traffic injection points of the application, where its processes attach to the emulated network, assuming that the application fully utilizes the network link at each injection point and every node talks to all other nodes with evenly distributed bandwidth. While this approximation may seem coarse at first glance, it is acceptable when considering that most target applications in emulation are complex and network intensive.

With the source/destination pairs of all traffic flows, we can compute the aggregated traffic on each link by summing the contribution from each flow. To identify the routes used in the emulated network, we instantiate the emulated network and detect the actual routes used (based on dynamically generated routing tables and routing protocols). To get the routing information, we implement the ICMP protocol inside the MaSSF, and use the real Linux *traceroute* tool to discovery the routing paths between each source-destination pair. To reduce the number of *traceroute* execution required, we could use one representative endpoint for each sub-network and only discover the route paths between those sub-network representatives.

With this predicted traffic information, we can improve the approach in Section 3.1 by recalculating vertex/edge weights. This extra information also enables another objective, which is to minimize the traffic across partitions. In the approach, we use the multi-objective graph partitioning algorithm described in Section 2.3.

3.3 Profile-Based Mapping

The third approach uses profiling techniques to obtain traffic information automatically from emulation experiments (PROFILE). The profiles are then used to estimate future network use, and to improve the network mapping. Typically this involves an initial emulation experiment using an initial partition and traffic monitoring. The emulation yields detailed traffic information and the network can be repartitioned based on this information.

The critical challenge for this approach is the efficient collection and representation of traffic information during profiling, and the use of this information to repartition the network. In MaSSF, we implement the Cisco NetFlow-like [19] function on each emulated router. This functionality is used to record every traffic flow on each router to a local file. The dump files record the average bandwidth and duration of every flow on every router. Parsing the dump files allows computation of the aggregated traffic on every router and link in the network. By tuning the granularity of the NetFlow, we can get detailed network traffic information with small overhead.

In our implementation, the real network traffic data does not actually travel through the emulator; only packet references are processed by it. Instead of using the real network bandwidth (MB/s) as the bandwidth measurement, we use the number of packets in a flow, since the real load in the emulator depends on the number of packets it processes. We also measure the live traffic injection overhead by the number of requests coming from the application.

Using profile data, we have much more accurate traffic information about the virtual network. We could apply the same multi-objective graph partitioning algorithm described in Section 2.3. However, that approach uses the average traffic intensity over the whole emulation period, in the process, ignoring a wealth of traffic detail available now. For example, the load imbalance pattern may vary at emulation stages (as shown in Figure 2), and different nodes dominate the load imbalance at different stages. Also, at some stages, the traffic load is so low that even heavy load imbalance has no appreciable affect on overall emulation performance.

Using the average load (a single number) over the entire emulation period neglects the critical dynamic behavior of the network traffic and may create poor load balance results. To solve this problem, we employ a clustering algorithm, which automatically detects and splits the whole emulation period into multiple segments. Each segment is used to calculate a group of vertex weight, which is taken as a constraint of the graph partitioning algorithm. Using the multi-constraint graph partitioning algorithm (METIS), it tries to balance the requirement from different emulation stages and achieve better results.
The clustering algorithm first removes segments that have little traffic. Then it gets a smooth load curve for each physical node by calculating the average load of each node over a larger period of time. The dominating node of special point is the node with the maximal load. The change of dominating node identifies a major load variation of the emulation system. So we can split the whole emulation period at these odd points and use each segment as a constraint to the graph partitioning algorithm.

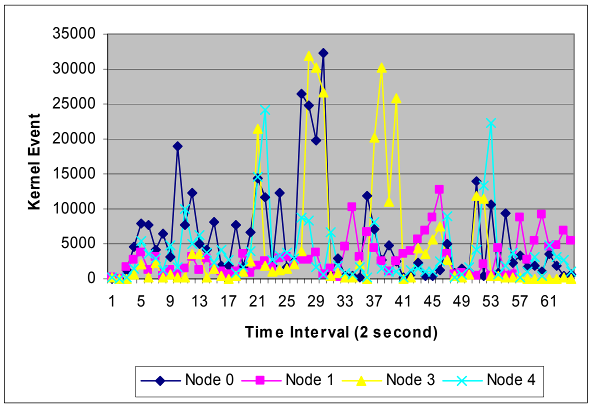

Figure 2. Load Variation Over the Lifetime of an Emulation

4. Experimental Evaluation

4.1 Methodology and Experimental Setup

To evaluate these mapping approaches, we implement them in the MaSSF network emulator of the MicroGrid Project [11]. MaSSF is a distributed network emulator which supports direct execution of real applications and also provides background traffic generators to setup a user-controllable network traffic conditions. We apply these approaches on a range of different emulated network topologies and background traffic conditions.

4.1.1 Metrics

Three evaluation metrics are used in the experiments: load imbalance, application emulation time, and network emulation time. We define the load of a simulation engine node as the simulation kernel event rate (essentially one per packet). Using these counters, we calculate the overall ***load imbalance*** across all the physical nodes. Assuming the simulation kernel event rates are k_1, k_2, …, k_n, for n nodes used by the simulation engine, the load imbalance is calculated as the normalized standard deviation of {k}.

The second metric is the ***application emulation time.*** If load balance is improved, this improvement should reduce the execution time of the application emulation. Since communication is typically the performance bottleneck for only part of the execution time, the application emulation time is not always directly correlated to network emulation load balance. Nevertheless, as faster emulation is the ultimate goal of load balance, it is an important criterion.

The third metric is ***network emulation time***, which directly measures how much time is required to emulate the traffic created by the application. MaSSF records all network traffic trace of an emulation execution, and then replays it without real computation in the application. When replaying, it tries to send out traffic as fast as possible, but still follows the real application casualty and message logic order. This is a direct measurement of the mapping approaches.

Network Topology	Router	Host	Emulation Engine Node
Campus	20	40	3
TeraGrid	27	150	5
Brite	160	132	8

Table 1. Network Topology Setup

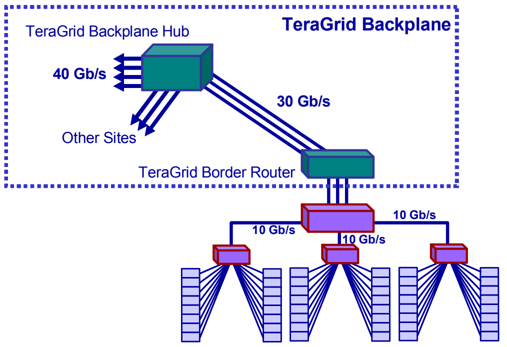

Figure 3. TeraGrid Site Network Architecture for any of the five sites, connected with 40Gbps network.

4.1.2 Hardware Configuration

The experiments use two RedHat Linux clusters. The first cluster includes 24 dual 550MHz Pentium-II processors, linked with 100Mbps Ethernet switch, with 2Gbps backbone bandwidth. This cluster is used for the network simulation engine. The second cluster consists of 8 dual 1.6GPentium-III processors, linked with 1 Gbps Ethernet switch (with 24 Gbps backbone bandwidth). It is mainly used for the real application execution. Two clusters are connected by a single, full duplex gigabit Ethernet link.

4.1.3 Network Topologies

Three network topologies are used in our experiments. The first two represent real networks, such as the TeraGrid (Figure 3 and see http://www.teragrid.org/) and a section of a university campus network (Campus). To explore more complex network structures, our third network topology Brite is created by a generic topology generator (adapted from the BRITE[20] toolkits), which creates Internet-like topologies and also provides background traffic support.

4.1.4 Traffic Workloads

The experiments use aggregated traffic flows to create background traffic. Users provide background traffic description, such as:

```
Traffic [name              HTTP
         request_size      200KByte
         think_time        12
         client_per_server 10
         server_number     10]
```

Here HTTP clients and severs are selected randomly from endpoints in the virtual network. In this study, a HTTP traffic generator is used, which has been well-studied by other researchers [21]. While this background traffic model is not perfect, it exercises some range of network dynamics, allows user control of load intensity by changing those parameters, and is widely used [22-24]. Detailed realistic traffic generation itself is a hot research topic and is beyond the scope of this paper.

Foreground traffic is created live from real Grid applications, including ScaLapack[25] and GridNPB3.0 [26]. ScaLapack is a linear algebra package widely used for scientific computing, and is implemented atop MPICH-G (which in turn uses a network of TCP/IP connections). In our experiments, it calculates a matrix equation of size 3000x3000 using 10 nodes, and runs for about 10 minutes on our emulation platform. GridNPB3.0 is a widely used set of grid benchmarks in a workflow style composition in data flow graphs encapsulating an instance of a slightly modified NPB task in each graph node, which communicates with other nodes by sending/receiving initialization data. GridNPB includes a range of computation types and problem sizes, and in our experiments we use the combination of Helical Chain (HC), Visualization Pipeline (VP), Mixed Bag (MB) applications, all run at class S size. These programs run for about 15 minutes on our platform.

4.2 Experiment Results

4.2.1 Load Imbalance

Application workloads are executed on three network topologies (Campus, TeraGrid, and Brite) with moderate background traffic, and the measured load imbalance for two applications (ScaLapack and GridNPB) is shown in Figures 4 and 5. The figures report the normalized load imbalance across the physical simulation engine nodes for each combination of mapping approach and network topology. Each mapping approach produces significantly different results. The application placement-based mapping (PLACE) improves significantly on topology-based mapping (TOP) for both ScaLapack and GridNPB applications. The profile-based mapping (PROFILE) further improve the load imbalance up to 66% and 48% for ScaLapack and GridNPB respectively. For both workloads, the profile-based mapping approach delivers the best performance among three approaches. This is as expected, since it uses detailed traffic information from previous simulation execution to partition the network.

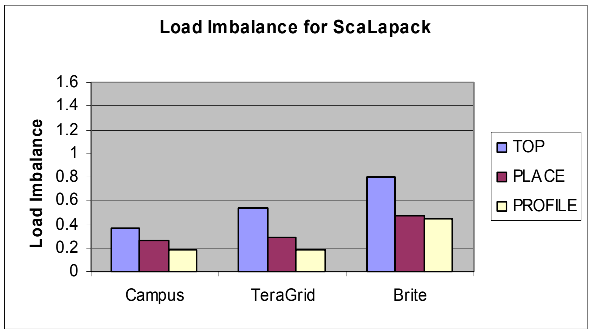

Figure 4 Load Imbalance for ScaLapack

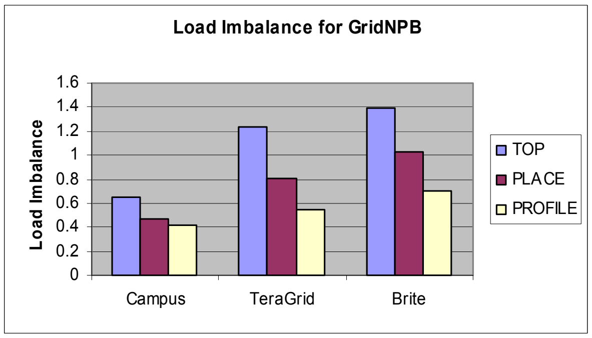

Figure 5 Load Imbalance for GridNPB

The improvement of profiled-based mapping over placement-based mapping for GridNPB is more significant than that for ScaLapack. This is due to the fact that for ScaLapack, the application-placement based traffic prediction is very close to the actual traffic pattern, so there is little improvement to be had for PROFILE. For GridNPB, in contrast, the traffic is more irregular and the application-placement based prediction is less accurate. As a result, significant load imbalance remains for PLACE, leaving more room for improvement for PROFILE.

We can also see that the scale of the emulation affects the achieved load balance. The Campus network uses 3 simulation engine nodes, the TeraGrid uses 5 nodes, and the Brite network uses 8 nodes. The normalized load imbalance increases when the number of simulation engine nodes is increased, as you'd expect if work were held constant (it is not across these experiments). When the emulation scales up, load balance is more critical to achieving high performance.

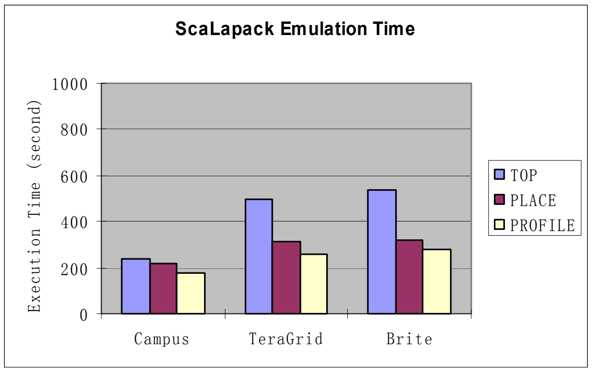

Figure 6 Emulation Time for ScaLapack

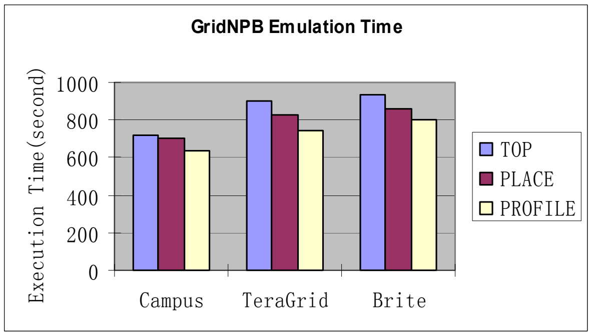

Figure 7 Emulation Time for GridNPB

4.2.2 Application Emulation Time

The emulation time of both applications is shown in Figures 6 and 7. For ScaLapack, the use of application placement-based mapping (PLACE) reduces overall emulation time significantly (about 40%), and the use of the profile-based mapping (PROFILE) further reduces the emulation up to 50%. For the GridNPB workload, we can see the benefits of both PLACE and

PROFILE mappings, but the improvement is much smaller (about 17%). As we have mentioned before, the emulation time is not a direct measurement of load imbalance, and because the execution time of GridNPB is computation rather than communication-intensive, improvement of the emulator gives little overall runtime benefit.

To provide further insight, we show the fine-grained load imbalance of the Campus network emulation in Figure 8. We collected the actual load of simulation engine nodes in two second intervals and calculate the load imbalances for each period. As shown in Figure 8, the load imbalance of profile-based approach is actually greatly improved compared to the topology-based mapping approach, even the overall execution time is not significantly improved.

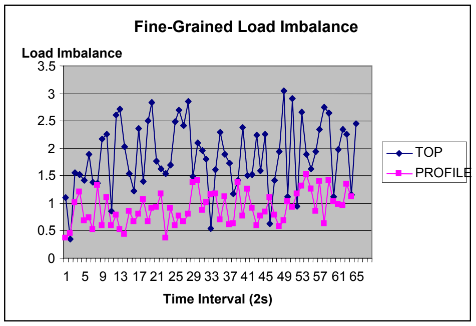

Figure 8 Fine-Grained Load Imbalance of GridNPB

4.2.3 Scalability

To evaluate the effectiveness of our three mapping approaches for larger network emulations, we use BRITE to build a network topology with 200 routers and 364 hosts. The emulation itself uses 20 simulation engine nodes and ScaLapack uses 10 additional nodes. Since the current BRITE tool cannot create networks using BGP routers, all the routers are created in a single AS. The routing table size increases rapidly with the number of routers in the network, so our hardware infrastructure currently limits us to networks with about 200 routers. However, we can still increase the number of hosts and the background traffic intensity to produce a larger network and increase the size of the emulation. The result is presented in Table 2.

ScaLapack	TOP	PLACE	PROFILE
Load Imbalance (Std. Deviation)	1.019	0.722	0.688
Execution Time(second)	559.332	484.573	460.544

Table 2. Results of ScaLapack on Larger Network

Since the network topology and background traffic are quite different, we cannot compare the execution time directly to the experiments in Section 4.2.1. But by comparing the result of the different network topologies, it is clear that the profile-based approach still creates the best partition for this large network emulation.

4.2.4 Network Emulation Time in Isolation

All experiments above use the emulated application as targets, and the computation and communication are mixed together. To further understand the direct effect on network emulation, we use the MaSSF replay function to study the network emulation performance in isolation, as mentioned in Section 4.1.1. Figures 9 and 10 show that the emulation time for network traffic is improved significantly for ScaLapack replays, in consistent with the result of overall emulation time in Figure 6. For GridNPB, the network emulation time is also reduced by 30%, even when the execution time for the whole application shows less difference in Figure 7.

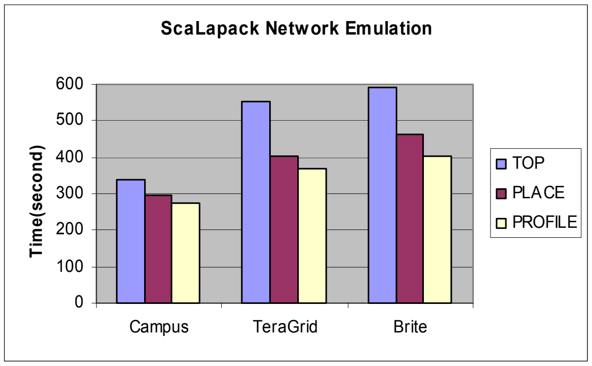

Figure 9 ScaLapack Isolated Network Emulation

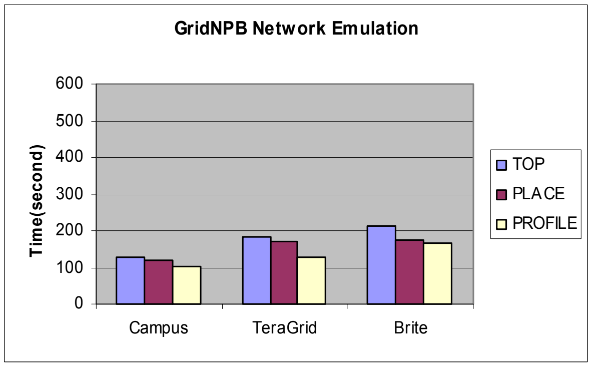

Figure 10 GridNPB Isolated Network Emulation

4.3 Summary

Experimental results show that network mapping using static network topology and predicted traffic information can improve load balance in large scale network emulation. The topology-based approach (TOP) is fast and simple, and the placement-based approach

(PLACE) can improve the performance for application with evenly distributed traffic load. For more irregular application and real large emulation, the profile-based approach (PROFILE) is most effective. Depending on the special network structure and traffic load, it can improve load balance by up to 66% and speed up the emulation up to 50%.

5. Discussion and Related Work

Load balance is known to be an important problem for the scalability of distributed network simulations or emulations, however there are only a few efforts in network simulation/emulation community [7-10] to solve this problem. Many projects use either manual partitioning or simple graph partitioning based on network topology. The DaSSF [27] simulator uses the METIS graph partitioning package and link latencies for load balance. It does not use link capacities or any further detailed traffic information. Others[10] uses the greedy k-cluster algorithm: for k nodes in the core set, randomly selects k nodes in the virtual topology and greedily selects links from the current connected component in a round-robin fashion.

Netbed's *assign* [28] maps virtual topologies which include endpoint resources as well as network structures onto a heterogeneous combination of routers, switches, and computers. Critical issues are time to compute mapping, physical resources used, and sufficient link capacity. Thus, *assign* chooses specific endpoint and network resources and subject to the constraints optimizes their quantity. Load balance is not a direct focus. In contrast, our MaSSF partitioner focuses on balancing the computational cost of the network simulation, while also optimizing core bandwidth use. Load balance is computed on the basis of network structure, predicted irregular traffic load, and profiled traffic, to improve efficiency. The MaSSF partitioner currently assumes homogeneous physical resources for network simulation. Improved emulation efficiency is used to achieve the same real-time performance with less resources or larger system emulation with fixed resources.

In this paper, we address load balance problem in a more systematic fashion. Both placement-based and profile-based approaches exploit detailed network topology and traffic information to partition the virtual network, and therefore are expected to achieve better load balance for most situations when compared to simple hierarchical partition. For example, by minimizing the network traffic across partitions, it attempts to limit a large traffic flow to small number of partitions. Our experimental results confirm this intuition. To get good results from the placement-based approach, an application should have balanced traffic amongst all application nodes, such as the ScaLapack program. For more irregular or complex application, the profile-based approach works better. Our experiments also show that NetFlow information can accurately represent the traffic information in the network, thus be used to create a well-balanced network partition. This provides a great chance for scalable network emulation.

However, there are still two user decided magic numbers in our three approaches. The first is the tradeoff between the maximal latency and minimal edge-cut traffic. Since the effect of maximal latency changed on different physical resources and number of physical nodes used in the emulation, the user may try different ratios himself and find the best priority ration. In current implementation, the default latency/traffic priority ratio is 6:4. The performance is not very sensitive to this ratio, and this ratio should be good for a switch connected cluster with less than 100 nodes. Usually the more simulation engine nodes used, the greater the importance of latency priority. Another magic number is the tradeoff between computation requirement and memory requirement. When the simulation engine has enough physical memory, the weight of memory should be small. However, when the result partition is so uneven that some partitions gets too many virtual nodes and it is likely to run out of memory on those nodes. So we must increase the weight of memory when the physical memory becomes a possible bottleneck of the system. In our experiments, we use the $m=10+x*x$ as the memory requirement for a router, where x is the size of an AS. It will be part of our future work to adjust these parameters automatically. For example, given a partition, MaSSF can predict more accurate memory requirements on every simulation engine node. If the memory imbalance will hurt performance or correctness, then it can adjust the memory weight and repartition automatically.

Mapping results of these approaches also largely depend on the quality of graph partitioning algorithms. There are a large number of graph partitioning packages targeted at parallel computing since the early 90's, including METIS[18], Chaco[17], Jostle[13], PARTY[15], and Zoltan[29]. While the standard graph partitioning algorithms create high-quality partition results for calculation on structured graph and mesh, their expressibility is still quite limited to address complex applications [16]. It is critical for users to setup correct application models to get the good results out of these partitioning algorithms, and this is exactly what we are doing in this paper. Our choice of METIS is mainly due to its flexibility in supporting multiple constraints and multiple objectives.

6. Conclusion and Future Work

Load balance is critical to achieve scalability for large network emulation and it is also a challenging task. By carefully mapping the virtual network to physical resources using multi-objective graph partitioning algorithms, we achieve good load balance and better scalability in network emulation. The accurate prediction of network traffic is critical to the success of this approach. Our studies show that the static network topology and application placement information can be exploited to achieve good balance for some application. In our experiments, it reduces the load balance by up to 66%. The profile-based mapping uses detailed traffic information and further reduces the application emulation time up to 50%, and this approach is promising to achieve scalable network emulation.

Currently, profiling is required for each emulation -- the specific topology and application. This is quite an overhead for large scale emulation or parameter sweep experiments. Moreover, it is not accurate if the application shows great dynamic behavior under different network conditions. It is desirable if we can figure out the application traffic pattern after a couple of profile runs and then we can use the profile data for other similar emulations.

Load imbalance happens due to burst/variation of traffic injected from the application. Static partitions are fundamentally limited for large emulation if traffic varies widely. Even the clustering and multi-constraint partitioning algorithm discussed in Section 3.3 won't solve the basic problem. Dynamic remapping the virtual network during the emulation is the only solution. Such dynamic remapping is a major challenge for distributed emulators like MaSSF.

Acknowledgements

The authors and research are supported in part by the Defense Advanced Research Projects Administration through United States Air Force Rome Laboratory Contracts AFRL F30602-99-1-0534 and the National Science Foundation thru NSF EIA-99-75020 Grads and NSF Cooperative Agreement ANI-0225642 (OptIPuter) to the University of California, San Diego Support from Hewlett-Packard is gratefully acknowledged.

The authors also acknowledge the contributions of Alex Olugbile and Huaxia Xia to the system infrastructure which made this work possible. We also appreciate the feedback from anonymous reviewers and Andrew Lawrence Wendelborn, our paper shepherd, who all helped to significantly improve the paper.

Reference

1. Fugui Wang, P.M., Sarit Mukherjee, Dennis Bushmitch, *A Random Early Demotion and Promotion Marker for Assured Services.* IEEE Jour. on Selected Areas in Communications, 1999.
2. Tao Ye, S.K., David Harrison, Biplab Sikdar, Bin Mo, Hema Tahilramani Kaur, Ken Vastold, Boleslaw Szymanski, *Network Management and Control Using Collaborative On-line Simulation.* Proc. IEEE International Conference on Communications,, June 2001.
3. D. Katabi, M.H., and C. Rohrs. *Internet congestion control for future high bandwidth-delay product environments*. in *Proc. ACM SIGCOMM*. 2002. Pittsburgh, PA.
4. Christina Parsa, J.J.G.-L.-A. *Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media*. in *Proceedings of the 7th IEEE International Conference on Network Protocols (ICNP)*. 1999.
5. Oram, A., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. March 2001: O'Reilly.
6. Ian Foster, C.K.e., *The Grid: Blueprint for a New Computing Infrastructure*. 1999: Morgan Kaufmann.
7. Pei Zheng, L.N. *EMPOWER: A Network Emulator for Wireless and Wireline Networks*. in *Infocom 2003*. 2003. San Francisco.
8. Rob Simmonds, R.B., and Brian Unger. *Applying parallel discrete event simulation to network emulation*. in *14th Workshop on Parallel and Distributed Simulation (PADS 2000)*. May 28-31, 2000. Bologna, Italy.
9. Brian White, J.L., Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. *An Integrated Experimental Environment for Distributed Systems and Networks*. in *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*. December 2002.
10. Amin Vahdat, K.Y., Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. *Scalability and Accuracy in a Large-Scale Network Emulator*. in *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*. December 2002.
11. H. Song, X.L., D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. *The MicroGrid: a Scientific Tool for Modeling Computational Grids*.

in *IEEE Supercomputing (SC 2000)*. 2000. Dallas, USA.

12. Jason Liu, a.D.M.N. *Learning Not to Share*. in *Proceedings of the 15th Workshop on Parallel and Distributed Simulation (PADS 2001)*. 2001. Lake Arrowhead, CA.
13. C. Walshaw, M.C., S. Johnson, and M. Everett. *JOSTLE: Partitioning of Unstructured Meshes for Massively Parallel Machines*. in *Parallel CFD'94*. 1994. Tyoto, Japan.
14. F. Pellegrini, J.R. *SCOTCH: a software package for static mapping by dual recursive bipartitioning of process and architecture graphs*. in *High-performance Computing and Networking, Proc. HPCN'96*. 1996. Springer, Berlin.
15. Preis R., D.R., *PARTY - A Software Library for Graph Partitioning.* Advances in Computational Mechanics with Parallel and Distributed Processing, 1997: p. 63-71.
16. Hendrickson, B., *Graph Partitioning Models for Parallel Computing.* Parallel Computing Journal, 2000. **26**(12): p. 1519--1534.
17. B. Hendrickson, R.L., *The Chaco User's Guide: Version 2.0*. 1994, Sandia Tech.
18. Kirk Schloegel, G.K., and Vipin Kumar. *A New Algorithm for Multi-Objective Graph Partitioning*. in *Euro-Par'99 Parallel Processing*. 1999. Springer Verlag, Heidelberg.
19. Cisco Systems, *NetFlow*. 2001.
20. Alberto Medina, A.L., Ibrahim Matta, and John Byers. *BRITE: An Approach to Universal Topology Generation*. in *In Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems-MASCOTS '01*. 2001. Cincinnati, Ohio.
21. Paul Barford, M.C. *Generating Representative Web Workloads for Network and Server Performance Evaluation*. in *Measurement and Modeling of Computer Systems 1998*. 1998.
22. David P. Olshefski, J.N., and Dakshi Agrawal. *Inferring Client Response Time at the Web Server*. in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2002)*. 2002. Marina del Rey, CA.
23. Rong Pan, B.P., Konstantinos Psounis, and Damon Wischik. *SHRINK: A Method for Scalable Performance Prediction and Efficient Network Simulation*. in *IEEE INFOCOM*. 2003.
24. Jaeyeon Jung, B.K., and Michael Rabinovich. *Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites.* in *Proceeding of 11th World Wide Web conference*. 2002. Honolulu, Hawaii.
25. A.Petitet, S.B., J.Dongarra, B.Ellis, G.Fagg, K.Roche, and S.Vadhiyar. *Numerical Libraries and the Grid: The GrADS Experiment with ScaLAPACK*. in *International Journal of High Performance Computing Applications*. 2001.
26. Frumkin, R.F.V.D.W.a.M., *NAS Grid Benchmarks Version 1.0*. 2002, NASA Ames Research Center.
27. James Cowie, H.L., Jason Liu, David Nicol and Andy Ogielski. *Towards Realistic Million-Node Internet Simulations*. in *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*. June 28 - July 1, 1999. Las Vegas, Nevada.
28. Robert Ricci, C.A., Jay Lepreau, *A Solver for the Network Testbed Mapping Problem*. 2002, University of Utah Flux Group.
29. K. Devine, B.H., E. Boman, M. St.John, and C. Vaughan. *Design of Dynamic Load-Balancing Tools for Parallel Applications*. in *Proceedings of the International Conference on Supercomputing*. 2000. Santa Fe.