A Position Statement: An Approach to Measuring Large-Scale Distributed Systems

Jun Li, Peter Reiher, Gerald Popek Mark Yarvis Geoffrey H. Kuenning
{lijun, reiher, popek}@cs.ucla.edu mark.d.yarvis@intel.com geoff@cs.hmc.edu
University of California, Los Angeles Intel Labs Harvey Mudd College

Abstract: Realistic measurement of large-scale distributed systems poses unique challenges. Empirical measurements can capture the true behavior of a real system, but this approach is only feasible when the system is small in scale. Simulation is more scalable, but without running real software, it is difficult for simulation tools to capture all realistic effects.

This paper explores a different approach to measuring large-scale distributed systems. We introduce an overloading technique that uses a relatively small number of physical machines but supports the deployment of a distributed system consisting of a large number of logical nodes. We discuss the challenges and advantages of this approach and demonstrate its use to measure the Revere security update dissemination system.

1. INTRODUCTION

Conventional methods of measuring the performance of a distributed system face a dilemma between scalability and realism. Realistic measurements of large-scale distributed systems are particularly challenging. While empirical measurements can capture the true behaviour of a real system, the cost of gaining access to, configuring, maintaining, and obtaining results from more than a few hundred nodes is typically prohibitive. Simulation is a more scalable approach, but it is difficult for a simulation to capture all aspects of a real system, such as hidden costs and subtle timing effects. In addition, the simulated version of a software system is typically different from the software that would actually be deployed.

We explore a different approach to measuring large-scale distributed systems in this paper. In this approach, each individual node in a distributed system runs the real code, and a fairly large number of nodes may be used. Our approach employs a technique called “overloading” in which multiple instances of a software system execute on the same physical node.

In the purely real world, a physical machine typically maps to one individual node of a distributed system. Via this overloading technique, however, a physical machine can be overloaded with many nodes of a distributed system, where each logical node still runs the real code and

communicates with other logical nodes, just as it would in the real world. Large scale can then be achieved using multiple physical machines, each supporting many logical nodes.

One fundamental issue that arises is how to run a distributed system with this overloading technique while still achieving accurate measurement results. In particular, messages between the nodes will now follow different transmission paths than would be taken in the purely real world. For example, two logical nodes that are collocated on the same physical node will now communicate without crossing a wire.

We address these and other issues that arise from overloading in this paper. Section 2 will describe how a large-scale distributed system can run with a limited number of physical machines, using a virtual topology to assign logical nodes to a smaller number of physical machines and to model the communication between those nodes. In Section 3 we discuss measurement using this overloading approach, including techniques that compensate for resource sharing between logical nodes on physical nodes. In Section 4, we illustrate our measurement approach by applying it to a specific example—measuring the performance of a security-update dissemination system. Section 5 describes open issues remaining in this approach, and we conclude the paper in Section 6.

2. RUNNING ATOP A VIRTUAL TOPOLOGY

The nodes of any distributed system must exist on top of some topology. When overloaded on top of physical machines, however, the nodes of a distributed system will have a different topology than they would in the real world. Such a topology, which may consist of a single machine, will not, by itself, reflect the characteristics of the topology of the distributed system.

A virtual topology can be employed to solve this problem. Each node of a particular distributed system can be viewed as attached to a particular location in a virtual topology. Such a node communicates through this virtual topology to another node, which is attached to the same virtual topology. A virtual topology can be generated using one of many existing topology generation tools, such as GT-ITM [1], Tiers [3], Inet [4], or Brite [5], depending on the characteristics of the distributed system.

With the notion of a virtual topology, a distributed system can be created as follows. After generating a virtual topology, treat each logical node in the virtual topology as an individual node of the distributed system. For each virtual node, run the software of the distributed system on top of a physical machine, where multiple instances of the software program may be invoked on the same machine. As a result, the performance of this distributed system

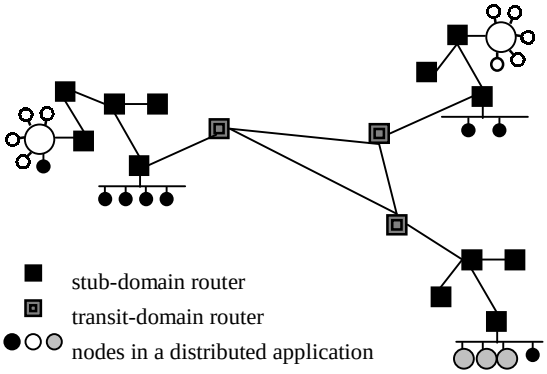

Fig. 1. A virtual topology with nodes of a distributed system

The nine hollow circles attached to token-ring networks represent nodes assigned to the same physical machine, the three shaded circles attached to an Ethernet represent nodes on another physical machine, and the eight solid circles are to a third physical machine.

can be measured. (We will discuss the measurement procedure in the next section.)

While it may be possible to map all nodes of a virtual topology to a single physical machine, multiple machines will typically be required for larger scalability, each assigned a subset of nodes from the common virtual topology. The node assignment can be fulfilled by contacting a virtual topology server that keeps track of which nodes are already assigned and which are still outstanding.

Figure 1 shows a virtual topology in which a distributed application runs with 20 nodes that communicate across transit-domain routers and stub-domain routers. As shown in the figure, these 20 nodes are assigned to three physical machines.

One important issue is to ensure that the real software still functions in this new mode of execution. One side effect of overloading is the identification of each node in the distributed system. In a real system, the address of the underlying physical machine can be used to identify a logical node. Since each physical node is overloaded with multiple logical nodes, logical nodes can no longer be identified using the machine address. To solve this, each node now has to be identified using the machine address coupled with some unique number, such as a TCP port number bound to the logical node (which is unique since two logical nodes will not be allowed to use the same port number).

3. MEASURING ATOP A VIRTUAL TOPOLOGY

Since overloading typically maps several logical nodes onto a single physical node, the logical nodes must share the resources of the physical node. This resource sharing can, but does not necessarily, affect the performance of individual logical nodes.

Many results obtained in a virtual topology will not differ from those obtained while running atop a real topology with the same structure. For example, whether the underlying topology is real or virtual, the storage cost or bandwidth cost incurred at an individual node of a distributed system will not typically be affected.

Also, the characteristics of the communication paths between any two nodes of a distributed system can be easily determined based on the specification of a virtual topology. For instance, if the length of every link in a virtual topology is known, the shortest path between any two nodes on the virtual topology can be calculated using Dijkstra's algorithm [2], instead of being measured.

However, logical nodes on the same physical node must share the processor and memory. Thus, the processing time of each individual node performing a particular task will be affected. Due to the overloading of the underlying physical machine, multiple nodes, if running concurrently, will cause resource contention and result in longer processing times.

This problem can be solved in three ways. The first approach is to remove the resource contention, thus causing the measured processing time on an overloaded node to be the same as the real value. If only a single logical node at a time is allowed to proceed with full usage of system resources, the time spent by this node on a task should incur approximately the same amount of time as it would in the real world. However, this approach may require a logical node to wait for access to the resources to perform a particular task. If latency is important, this approach will not be appropriate.

The second approach is to calculate a slowdown factor and apply that to the measured processing latency. A slowdown factor can be estimated by overloading with a different number of nodes on a physical machine and comparing the impacts. For example, if a task consumes t_0 seconds when n nodes of a distributed system are evenly loaded into n physical machines, but t seconds if all n nodes are overloaded on one physical machine, we then can obtain a slowdown factor t/t_0 for physical nodes overloaded by a factor of n. This method works well when the processing time slows down linearly; otherwise, it must be carefully applied. To gain a more accurate understanding of the slowdown factor of a distributed system, measurement of overloading factors should be performed.

The third approach, using a divide-and-conquer method, is to divide the task being measured into several disjoint subtasks that are more easily measured. Here, some conditions must be met: (1) every subtask must be independent of the others, (2) subtasks must not overlap in terms of processing latency, and (3) the sum of all subtasks must be the total processing latency. For example, to evaluate the delay of forwarding a packet from source to destination, literally measuring the interval from sending time to receiving time is inaccurate when machines are overloaded. On the other hand, by dividing the whole delay into transmission delay along the wire, processing delay at each router, and queuing delay at each router, each component can be measured separately.

The first approach usually requires a new resource-control mechanism to coordinate the usage of system resources. Thus, this approach will be easier to implement for some distributed systems than it is for others. The third approach is preferable to the second if a task can be easily divided into several subtasks, and each subtask can be easily and accurately measured. It may also be possible to combine these approaches. For example, a subtask may be measured by applying a corresponding slowdown factor. In the next section, we will illustrate the use of the first and the third approach in an example system.

4. EXAMPLE—MEASURING REVERE

Revere is a system that provides secure information dissemination at Internet scale [6]. For instance, Revere can be used to distribute virus signature updates from a secure dissemination center. Participating Revere nodes organize themselves into an overlay network on top of the Internet, so that each node is able to both hear security updates and forward updates to others. This overlay network uses redundant data paths to provide fast and resilient service.

Rather than first deploying Revere into the Internet, Revere's performance can be measured using the overloading technique. In this section we introduce several key metrics for measuring Revere, describe our measurement procedure, justify our measurement method, and show several results.

4.1 Metrics

1. Join latency: Revere allows a new node to join Revere by attaching itself to one or more parent nodes on the overlay network. Join latency is the

time that a new node spends becoming a participant in the Revere overlay network.
2. Join bandwidth: The bandwidth spent to join the Revere overlay network.
3. Dissemination latency: The latency for a security update to reach an individual Revere node. Also relevant is the time needed to reach a certain percentage of all Revere nodes.
4. Overlay network resiliency: The percentage of Revere nodes that can still receive security updates, given that each node has a particular probability of failure.

4.2 Applying the Measurement Methodology to Revere

To overload different numbers of Revere nodes onto physical nodes, we used a testbed that consisted of ten machines. Every machine was equipped with an AMD Thunderbird 1.333 GHz CPU, 1.5GB SDRAM, and a 100 Mbps Ethernet interface.

To create virtual topologies, we used GT-ITM [1] to generate a router-level topology and designated certain numbers of Revere nodes to each router node. In each experiment, a topology server assigns every testbed machine the same number of Revere nodes.

We artificially divided the lifetime of Revere into three phases: the join phase, the dissemination phase, and the resiliency test phase. During the join phase, nodes sequentially join Revere and gradually form an overlay network. After all nodes have joined, the system advances into the dissemination phase, during which the dissemination center sends security updates through the overlay network to individual nodes for ten rounds. Finally, in the resiliency test phase, dissemination is tested in the face of broken nodes.

During the join phase, the join latency will be artificially increased if every physical machine is overloaded with a number of Revere nodes. The join bandwidth should not be affected by overloading. To gain accurate results, a token-controlled mechanism was designed to evaluate the join performance of each individual node, corresponding to the first approach discussed in Section 3. A Revere node can only begin running after it is granted a token by a token server, and it must return the token after it joins Revere. By enforcing only one token for all Revere nodes on all physical machines, only one node will be in the process of the join procedure at any time during the joining phase. Other nodes may be temporarily activated when requested to interact with the joining node. In doing so, the measured results of join latency and join bandwidth should be approximately the same as the real cost of joining a single node.

During the dissemination phase, each node behaves in a store-and-forward manner. But again, because many Revere nodes are running on a physical machine, simply measuring the interval between sending a security update and receiving it at a node cannot reflect the realistic value of dissemination latency. Given the artificially heavy load on the physical machines, both the processing delay and the kernel-space-crossing delay will be lengthened.

To solve this problem, we employed the divide-and-conquer method as described in Section 3. First, we divided the latency of disseminating a security update into three parts: the security update processing delay at each hop, the transmission delay of crossing the virtual topology, and the kernel-space-crossing delay. Second, we evaluated each part separately. Without overloading a physical node, the true processing delay per hop can be measured in a separate experiment. In the same manner, the kernel-space-crossing delay per hop can also be measured. The communication latency can be calculated using Dijkstra's algorithm over the virtual topology graph underneath. To transmit a 1-kilobyte security update over the virtual topology we used, the router-to-router latency ranges from 1ms to 70ms, with the average 23ms. Third, we added all parts of the dissemination latency together. Notice that with a given overlay network structure, the hops that a security update travels to reach a node are invariant, no matter how many nodes are simultaneously running on the same physical node. By multiplying the processing delay per hop and the kernel-space-crossing delay per hop, and adding the communication latency, we can obtain a very good approximation of the dissemination latency in large-scale scenarios.

During the resiliency test phase, each node on the overlay network was assigned a uniform probability of failure to test how many nodes are still reachable during the dissemination procedure. The divide-and-conquer method was again used to evaluate the latency of disseminating security updates toward the remaining nodes. The dissemination latency was divided, as before, into three parts, and measurement was performed as in the dissemination phase.

4.3 Selected Measurement Results

Figure 2 shows the outbound bandwidth that each node incurs during the join phase, for various sizes of Revere networks. This bandwidth cost includes the messages that a node sends when joining the overlay network and the messages sent responding to the join requests of other nodes.

Figure 3 shows the latency experienced by a node joining Revere, in Revere networks of various sizes. In this experiment, each node completes the join procedure after successfully attaching itself to two existing Revere

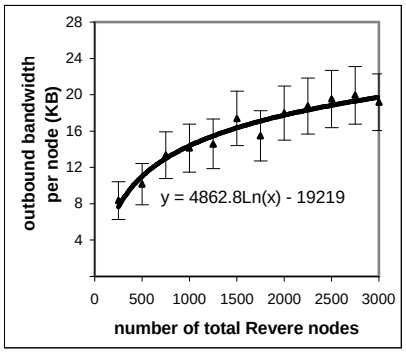

Fig. 2. Outbound bandwidth per node during joining phase (confidence level: 95%)

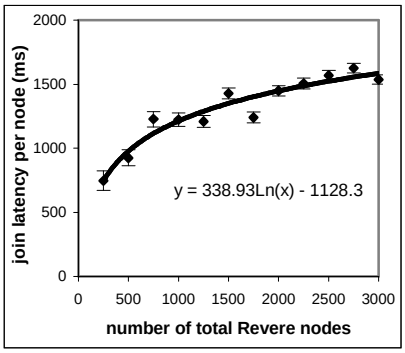

Fig. 3. Join latency per node (confidence level: 95%)

nodes. The cost of both outbound join bandwidth and join latency are acceptable, basically following logarithmic trends as the number of nodes grows.

To understand the dissemination property of Revere, we first measured the average and maximum hop count for disseminating security updates (Figure 4). Both also follow logarithmic trends versus the total number of nodes.

We obtained the dissemination latency of each node and derived the percentage of nodes covered as the dissemination proceeds. Figure 5 shows the dissemination coverage over time for a 3000-node dissemination. In this case, 100% of the nodes are reached in a short time (less than 1 second).

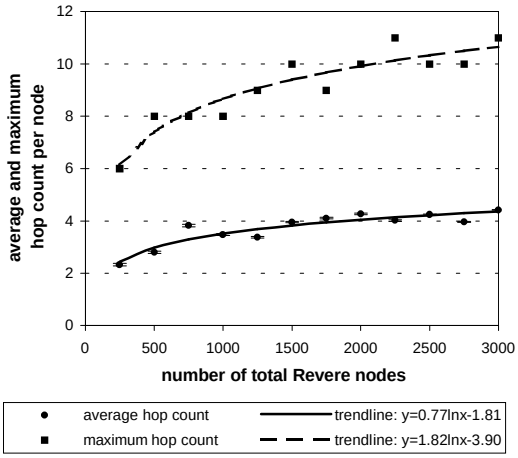

Fig. 4. Hop count of security update dissemination (confidence level for average hop count: 99.9%)

Figure 6 depicts the resiliency characteristics of the Revere network. After the failure of as many as 15% of the nodes, a high percentage (93%) of the remaining nodes are still able to receive security updates without readjusting the structure of the dissemination overlay network.

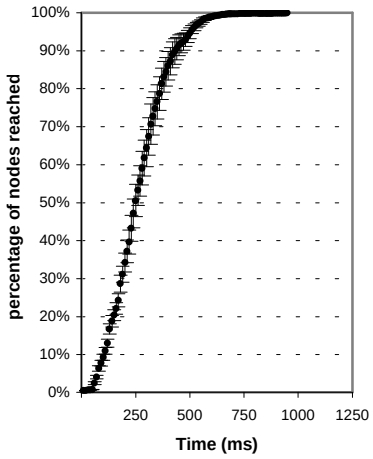

Fig. 5. Security update dissemination coverage for a 3000-node dissemination (confidence level of coverage: 99%)

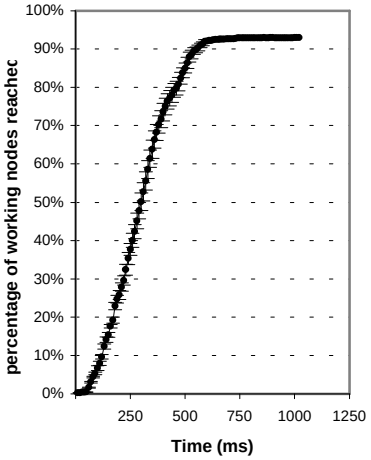

Fig. 6. Resiliency test for a 3000-node dissemination with 15% of nodes failed (confidence level of coverage: 99%)

5. OPEN ISSUES

This approach to measuring large-scale distributed systems requires that multiple nodes of a distributed system, if collocated on a physical machine, can still perform correctly. In practice, however, distributed systems are typically designed with the presumption that a single instance of the software executes on each physical node. Slight modifications to the distributed system may be necessary to allow it to be measured using this approach. As we pointed out earlier, for instance, systems that use an IP address as a node name will require modification.

Theoretically, it is also possible to build a common framework based on this approach to support measurement of differing distributed applications, and a specific distributed system can be measured by simply plugging it into such a framework. Designing an interface between the framework and the application being measured must be carefully considered.

Another issue is the scalability of this approach itself. Given that multiple nodes under this approach can contend for resources of the same physical machine, some resource locking mechanism is needed to obtain accurate results. An example of this approach is the token mechanism used in measuring the join performance of Revere. However, this technique slows down the measurement process. The token-controlled mechanism used to measure joins in Revere, for example, required about 100 minutes of measurement for 3000 nodes.

6. CONCLUSIONS

As more distributed systems run at Internet scale, understanding the performance of a system at large scale is important. Unfortunately, it can be difficult to measure a system that consists of very large numbers of nodes that are part of a large-scale network.

Without actual deployment, measurement of a large-scale system can be performed in two ways: simulation or the overloading approach described in this paper. Simulation is a popular approach for large-scale systems. However, since a simulation does not typically use the actual software and cannot accurately emulate all environmental factors, it is very hard for simulation tools to capture all the real effects of the system.

Our overloading methodology collocates a large number of nodes of a distributed system on a machine, while still allowing each node to run the real software. This methodology can accurately report those metrics that are invariant with respect to overloading, and can minimize those inaccuracies introduced due to overloading and resource contention.

Meaningful results can be obtained. We demonstrated this using the overloading approach for a security update dissemination system. While the measurements reported in this paper correspond to a 3000-node network, the results were obtained using only 10 nodes. We believe that Internet-scale results can be obtained using only a few hundred or a few thousand nodes. In addition, we believe that this approach can be further generalized into a common framework to support measurement of different distributed systems.

REFERENCES

[1] K. L. Calvert, M. B. Doar, and E. W. Zegura. "Modeling Internet Topology," *IEEE Communications Magazine* 35, 6 June 1997.

[2] E.W. Dijkstra. "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1:269--271, 1959.

[3] M. B. Doar. "A better model for generating test networks," *Proceedings of Global Internet*, November 1996.

[4] C. Jin, Q. Chen, and S. Jamin. "Inet: internet topology generator," University of Michigan Technical Report CSE-TR-433-00, 2000.

[5] A. Medina, I. Matta, and J. Byers. "On the origin of power laws in Internet topologies," *ACM Computer Communication Review*, 30(2), April 2000.

[6] Revere project home page. http://lasr.cs.ucla.edu/revere.