# Modeling and Simulation of Extreme-Scale Fat-Tree Networks for HPC Systems and Data Centers

NING LIU, Cleversafe, an IBM Company
ADNAN HAIDER, DONG JIN, and XIAN-HE SUN, Illinois Institute of Technology

As parallel and distributed systems are evolving toward extreme scale, for example, high-performance computing systems involve millions of cores and billion-way parallelism, and high-capacity storage systems require efficient access to petabyte or exabyte of data, many new challenges are posed on designing and deploying next-generation interconnection communication networks in these systems. Fat-tree networks have been widely used in both data centers and high-performance computing (HPC) systems in the past decades and are promising candidates of the next-generation extreme-scale networks. In this article, we present FatTreeSim, a simulation framework that supports modeling and simulation of extreme-scale fat-tree networks with the goal of understanding the design constraints of next-generation HPC and distributed systems and aiding the design and performance optimization of the applications running on these systems. We have systematically experimented FatTreeSim on Emulab and Blue Gene/Q and analyzed the scalability and fidelity of FatTreeSim with various network configurations. On the Blue Gene/Q Mira, FatTreeSim can achieve a peak performance of 305 million events per second using 16,384 cores. Finally, we have applied FatTreeSim to simulate several large-scale Hadoop YARN applications to demonstrate its usability.

CCS Concepts: • **Networks** → **Network performance evaluation**; **Network performance modeling**; **Network performance analysis**; • **Computing methodologies** → *Massively parallel algorithms*; *Modeling and simulation*; Network science

Additional Key Words and Phrases: Fat-tree network, high-performance computing, parallel discrete event simulation, distributed system

**ACM Reference Format:**
Ning Liu, Adnan Haider, Dong Jin, and Xian-He Sun. 2017. Modeling and simulation of extreme-scale fat-tree networks for hpc systems and data centers. ACM Trans. Model. Comput. Simul. 27, 2, Article 13 (July 2017), 23 pages.
DOI: http://dx.doi.org/10.1145/2988231

## 1. INTRODUCTION

The wave of innovation in big-data technologies has greatly propelled the development of both high-performance computing (HPC) systems, and large-scaled distributed network systems like data centers. For example, today's leading data centers typically take up millions of square feet, host sub-millions of physical servers, and consume million watts of energy; Exascale supercomputer systems, such as Oak Ridge National Laboratory's SUMMIT to be deployed by 2018, can deliver a peak performance of



This article is partly sponsored by the Maryland Procurement Office under Contract No. H98230-14-C-0141, and the Air Force Office of Scientific Research (AFOSR) under Grant No. FA9550-15-1-0190. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
Authors' addresses: N. Liu, 222 S Riverside Plaza #1700, Chicago, IL 60606; A. Haider, D. Jin, and X.-H. Sun, 3300 S Federal St, Chicago, IL 60616.

DOI: http://dx.doi.org/10.1145/2988231

0.3 exaFLOPS. These systems are designed to provide massive parallelism with the capability to efficiently process an extremely high volume of data. One key design aspect in such systems is the interconnection networks. According to the report from Cisco [Cisco 2015], the total amount of data processed in data centers is 3.8 Zettabytes in 2014 and will reach 8.6 Zettabytes in 2018, of which, around 75% is internal traffic. Fat-tree networks have served as the mainstream network topology in many distributed systems for decades. As the system scale rapidly increases, there is a pressing need to understand the performance constraints and to identify the optimal design principles of fat-tree networks in extreme-scale settings.

In the HPC community, there is a growing interest in understanding how parallel programs, including system software like MPI/OpenMP and scientific applications, scales in extreme-scale architectures. Torus networks have been widely adopted in HPC systems because of the low cost and high delivered bisection bandwidth and throughput. For example, the Blue Gene/L and P series and the CrayXT series use 3D torus networks, and the newly delivered Blue Gene/Q system uses a 5-D torus network. However, the torus network is inherently a blocking network, that is, the total number of available paths is smaller than the total demand. This can significantly exacerbate the network performance during communication bursts, such as the MPI All-to-All and the Reduce communication. A fat tree can provide the non-blocking feature with 1:1 subscription ratio and keep the total cost within a reasonable level [Real Cost 2015]. Oak Ridge National Laboratory has recently announced that the next generation OLCF supercomputer, SUMMIT, will adopt a fat-tree topology as its interconnection network [Summit 2015].

In order to quantify the performance and design trade-offs of extreme-scale systems, such as a multi-million node communication network, many researchers resort to parallel and discrete-event simulation (PDES). PDES can provide a scalable and cost-efficient alternative for evaluating systems whose architecture is still in the research stage, or systems that are economically infeasible to deploy.

In this article, we present a parallel simulation toolkit, FatTreeSim, for supporting the design and evaluation of large-scale data center and HPC communication networks, as well as the applications running on those networks. FatTreeSim is based on two parallel simulation packages: the Rensselaer Optimistic Simulation System (ROSS) [Carothers et al. 2000] and the Co-Design of Exascale Storage System (CODES) [Cope et al. 2011]. Figure 1 illustrates the architecture of CODES. The CODES-workload module [Snyder et al. 2014] is used to model and simulate HPC and Cloud workload traces and also supports real-data trace-driven simulation. The CODES-net module provides multiple PDES-based networking models as well as unified user interfaces to use them. The CODES-net includes four submodules: a torus network model [Liu et al. 2012], a dragonfly network model [Mubarak et al. 2012], a loggp [Alexandrov et al. 1995] model, and a simple-net model. To date, several simulation systems have successfully leveraged the functionalities provided by CODES-net [Liu et al. 2015; Tang et al. 2014]. In this work, our key contribution is to develop a highly scalable PDES-based fat-tree model as shown in Figure 1. FatTreeSim has been incorporated as a submodule in CODES and has been used by the aforementioned simulation systems. We highlight the features of FatTreeSim as follows:

(1) To the best of our knowledge, FatTreeSim is the first simulation system capable of modeling and simulating fat-tree networks with sub-million nodes in a time-efficient manner. FatTreeSim has demonstrated a close-to-linear scalability on the Blue Gene/Q system up to 32,768 cores and has achieved a peak event rate of 305 M/s in optimistic mode. The task of simulating a sub-trillion events test case can be accomplished within minutes instead of days as compared with the sequential discrete event simulation.

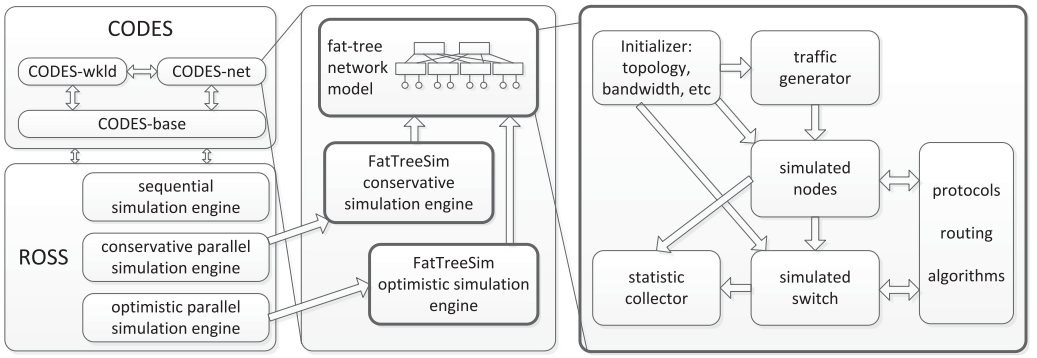



Fig. 1. The FatTreeSim system architecture in the CODES ecosystem. CODES is a platform using highly parallel simulation to explore the design of large-scale storage systems, I/O workloads, and HPC network fabrics. The underlying parallel simulation kernel is based on the Rensselaer Optimistic Simulation System (ROSS). FatTreeSim contains the scalable fat-tree model, as one of the core network models in the CODES-net module. The other components in FatTreeSim include Initializer, Traffic Generator, Nodes, Switches, Protocols & Routing Algorithms, and Statistic Collectors. FatTreeSim supports two both conservative and optimistic simulation modes.

(2) FatTreeSim models the communication-protocol-level granularity in a fat-tree network with an ECMP routing algorithm. To validate the model fidelity, we constructed the physical fat-tree networks in Emulab [Cutler et al. 2010] and developed network applications utilizing ECMP and MPI library for communication. We conducted extensive experiments with various network configurations including network size, traffic pattern, message size, and link bandwidth. The results show that errors are within 10% bounds.
(3) We demonstrate the functionalities and compatibilities of FatTreeSim through a Hadoop YARN application simulation case study. We integrated FatTreeSim with YARNsim [Liu et al. 2015] and show that the fat-tree model can effectively capture the traffic characteristics and enable the scalable simulation of multiple MapReduce benchmark applications.
(4) We extend FatTreeSim to support both conservative and optimistic parallel simulation, and conducted performance evaluation for both methods on the Blue Gene/Q Mira using up to 65,536 cores. The results show that the simulation performance can reach up to 655 million events per second on a 128-port 3-tree fat-tree network model (i.e., switch radix = 128 and number of levels = 3).

The remainder of the article is organized as follows. We present the background and motivation in Section 2. We describe the fat-tree model in Section 3. We evaluate the fidelity and the scalability of the fat-tree model in Section 4, and we discuss the related work in Section 5. Closing remarks and future works are presented in Section 6.

## 2. BACKGROUND & MOTIVATION

This section presents the necessary background information to understand the motivation of developing FatTreeSim and the principle application of FatTreeSim.

### 2.1. ROSS & CODES

FatTreeSim is based on two parallel simulation packages: ROSS [Carothers et al. 2000] and CODES [Cope et al. 2011]. ROSS is a massively parallel discrete-event simulator that has demonstrated the ability to process billions of events per second on large-scale HPC systems. A PDES system consists of a collection of logical processes (LPs), each modeling a distinct component of the system (e.g., a server). LPs communicate by

exchanging time-stamped event messages (e.g., denoting the arrival of a new I/O request at that node). The goal of PDES is to efficiently process all events in a correct global timestamp order while minimizing the processor synchronization overheads. Two well-established approaches toward this goal are broadly called conservative processing and optimistic processing. ROSS supports both approaches, and FatTreeSim also supports both conservative and optimistic parallel simulation (see Section 4.2).

CODES [Cope et al. 2011] is a simulation system based on ROSS. The goal is to enable the exploration and co-design of exascale storage systems by providing a detailed, accurate, and highly parallel simulation toolkit. Besides the two modules illustrated in Figure 1, CODES also includes two modules: CODES-bg, the Blue Gene/P storage system modle suites, and CODES-lsm, the local storage model. CODES is capable of simulating complex large-scale systems, for example, FusionFS [Zhao et al. 2014, 2013], a distributed file system for both HPC and cloud computing. The metadata in FusionFS is managed by ZHT [Li et al. 2013], a zero hop distributed hash table service. Researchers from Illinois Institute of Technology have built models of the two systems and conducted a performance evaluation at exascale. Tang et al. build a resource scheduler for multi-cloud workflows [Tang et al. 2014]. YARNsim [Liu et al. 2015] is the Hadoop YARN simulation system that aims to model and simulate extreme-scale Hadoop systems. However, currently the CODES-net module does not include fat-tree network models, which are the prevailing network topologies used in modern data centers. FatTreeSim targets the fat-tree networks and is built in the CODES-net module. Therefore, it is compatible with all the existing simulation systems built on CODES.

### 2.2. HPC & Data Centers and the Interconnection Networks

The design, evaluation, and deployment of data center and HPC system are systematic and time-consuming processes. The communication network has a significant impact on system performance. We need efficient communication networks to support a wide range of applications in both data centers and HPC systems, with different sets of communication and I/O requirements. This article concentrates on large-scale modeling and simulation of the fat-tree network, a promising candidate of the interconnection network for HPC and data centers. A fat-tree or folded-Clos topology is the conventional and yet still the most prevalent design choice for data center communication networks [Cisco 2015]. In distributed computing community, it is projected that a single data center can host millions of virtual machines and/or physical servers and serve multi-millions jobs/tasks simultaneously. The communication networks must guarantee the high availability and reliability, the desirable bisection bandwidth, and the support for multi-tenancy at that scale. It is desirable to develop a large-scale simulation toolkit that is capable of evaluating design principles and quantifying design trade-offs in a cost-effective manner.

### 2.3. Hadoop

In 2003, Google first proposed the MapReduce system [Dean and Ghemawat 2008], and since then it has been a widely accepted programming model in the distributed computing community because of its simplicity and efficiency. While Google keeps its MapReduce system proprietary, Apache Hadoop [Apache 2015] is the most popular open-source implementation of the MapReduce framework. According to a report from Gartner [Gartner 2015], 65% of the packaged data analytic applications will be built on Hadoop by 2015. Thus, it is necessary to evaluate the network performance with a wide range of Hadoop applications on a large scale.

We have built YARNsim [Liu et al. 2015]: a Hadoop YARN simulation system. Compared to other MapReduce simulation system [Wang et al. 2009], YARNsim is a parallel simulation system for extreme-scale Hadoop YARN systems. YARNsim also

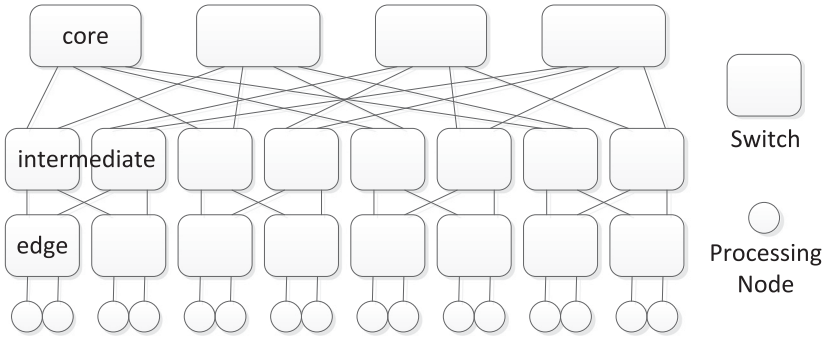



Fig. 2. A fat-tree network with the total number of processing nodes equal to 16, the fat-tree height equal to 3, the number of edge switch equal to 8, and the number of core switch equal to 4.

includes a comprehensive model for HDFS, which can mimic the I/O system behaviors of Hadoop YARN. However, YARNsim does not support a detailed parallel model of its communication network. We are motivated to develop FatTreeSim and integrate it with YARNsim, with the goal of accurately modeling and simulating Hadoop YARN at extreme-scale. We envision YARNsim and FatTreeSim to help the community to better evaluate Hadoop systems and understand the design trade-offs for various Hadoop applications.

## 3. DESIGN AND IMPLEMENTATION OF FATTREESIM

This section presents the design and implementation details of our parallel discrete-event simulation model of fat-tree networks and various modeling components in the FatTreeSim, including topology, traffic, and routing algorithms.

### 3.1. Modeling Fat-Tree Topology

We illustrate a simple topology of the fat-tree network in Figure 2, and it describes precisely how switches and hosts are interconnected. In graph representation, vertices represent switches or hosts, and links are the edges that connect them. The network topology is central to both the performance and the cost of the interconnection network. The topology affects a number of design tradeoffs, including performance, redundancy, and path diversity, which, in turn, affect the network's cost, resilience to faults, and average cable length.

A fat-tree can be described using the number of ports and the tree height. In general, a $m$-port, $n$-tree fat-tree have the following characteristics [Lin et al. 2004; Al-Fares et al. 2008]: the height of the tree is $n + 1$; $m$ is a power of 2; the tree consists of $m \cdot (m/2)^{(n-1)}$ processing nodes and $(2n - 1) \cdot (m/2)^{(n-1)}$ switches; each switch has $m$ communication ports. Thus, $m$ and $n$ determines the size of the fat-tree network. A four-port three-tree fat-tree network is given in Figure 2. Here, the tree height is 3, which means there are 3 levels of switches. The switch that connects directly to the processing nodes are called edge switch, and the topmost level switch is called the root switch. The rest switches are named intermediate switch.

### 3.2. Modeling Fat-Tree Traffic

The traffic within the network of data center or HPC system is often bursty, meaning a large volume of packets will be injected into the network within a short period of time. In HPC system, defensive checkpointping often happens after the computation phase and usually starts with a synchronization [Liu and Carothers 2011]. In data centers, the network traffic exhibits a high degree of variability and is usually non-Gaussian

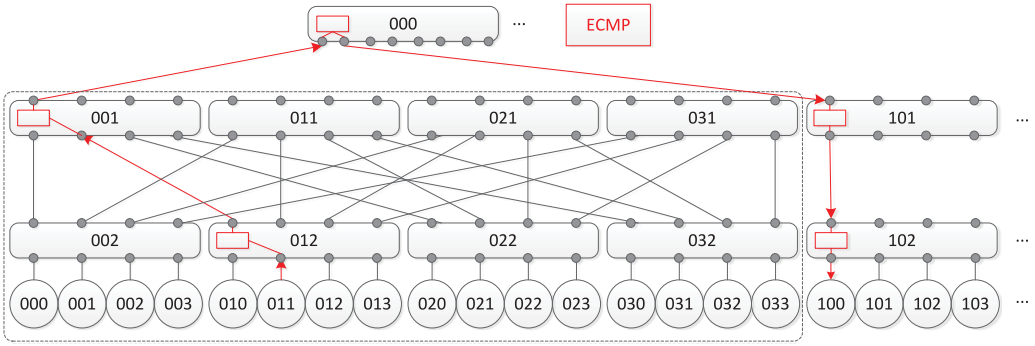



Fig. 3. A packet routing in a 128 node fat-tree network. Fat-tree setup: $m = 8$, $n = 3$. The total number of switches is 80. The source node is 011, and the destination node is 100.

[Mori et al. 2004]. In fact, a network that is only 10% utilized can see a lot of packet drops when running a web-search application [Abts and Felderman 2012].

In HPC and data centers, the communication pattern can be categorized as follows: (1) $N$ to 1; (2) 1 to $N$; (3) $N$ to $N$; (4) 1 to 1. In the first two scenarios, the application performs a global reduction or broadcast, which is a common traffic pattern. The third scenario is an $N$ round global reduction or broadcast, which can be categorized as the complex case of scenario (1) or (2). The latter traffic pattern is simply a random point-to-point communication. To represent and validate a variety of traffic patterns and test the scalability of FatTreeSim, we pick two traffic patterns: random-destination traffic and nearest-neighbor traffic [Liu and Carothers 2011; Mubarak et al. 2012]. With the random-destination traffic pattern, each source node randomly picks a destination node and generates a packet stream with the intervals apply to the exponential distribution. In fact, the generated packet is a Poisson stream. With nearest-neighbor traffic pattern, each source node picks the nearest-neighboring node or the second-nearest-neighboring node as the destination node. This traffic pattern generates packets at a fixed time interval and the ratio of packets passing through the core routers is also fixed. It is challenging to simulate random-destination traffic in discrete event simulation, because the generated packet flows have very low locality. ROSS optimistic simulation mode can handle this scenario in an efficient manner. The nearest-neighbor traffic is more balanced and thus is more amenable to parallel simulation performance.

### 3.3. Modeling Fat-Tree Routing Algorithm

The routing algorithm determines the route a packet traverses through the network. Non-blocking fat-tree network provides abundant path diversity in which multiple possible egress ports exist. To take the advantage of the path's multiplicity, equal-cost multi-paths routing algorithm (ECMP) is widely used [Al-Fares et al. 2008]. ECMP is a load-balancing routing protocol based on RFC 2991 [Hopps and Thaler 2015] that optimizes flows over multiple best paths to a single destination. ECMP applies load-balancing routing on flows such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), and can potentially increase the bandwidth between two endpoints by spreading the traffic over multiple paths. Path selection is based on hashing of the packet header. In recent years, networking researchers have pointed the limitations of ECMP routing and proposed dynamic routing algorithm like Hedera [Al-Fares et al. 2010]. A path through the network is called minimal path if no shorter path, less number of hops, exists. Different from other network topology, for example, dragonfly, a fat-tree, has multiple minimal paths. In Figure 3, we illustrate an exemplar packet routing procedure in a eight-port three-tree fat-tree network. The total number

of processing nodes equals to 128 and total number of switches equals to 80. Here, the packet starts from the source node 011 and tries to reach the destination node 100. As we can see, the packet reaches a different level at each hop, thus the maximum number of hops for the complete route is two times the tree height. In each step, the ECMP routing algorithm is used to determine the next hop node. In FatTreeSim, we model ECMP routing and focus on minimal path. Network flow control is also key to the network model. It dictates how the input buffers at each switch or router are managed. FatTreeSim buffers currently use the store-and-forward technique. The end-to-end delay, that is, the time taken for a packet to be transmitted across a network from source to destination, is broken up into a sequence of nodal delays. In fat-tree networks, the number is typically equal to the number of hops in a shortest path. Each nodal delay is the time between the arrival of a packet at a node and its arrival at the next node. Equation 1 decomposes the nodal delay into components to simplify the analysis.

$$T_{nodal} = D/B + T_{queue} + T_{prop} + T_{proc} \tag{1}$$

The processing delay $T_{proc}$ is the time that a node spends processing a packet, including time for error checking, time for reading the packet header, and time for looking up the link to the next node. $D$ is the packet size, $B$ is the link bandwidth, and $D/B$ is the transmission delay required to put an entire packet into the communication media. The queuing delay $T_{queue}$ is the time that a packet spends in a queue at a node while waiting for other packets to be transmitted/processed. The propagation delay $T_{prop}$ is the time that it takes a signal change to propagate through the communication media from a node to the next node. Those parameters are also configurable in FatTreeSim.

### 3.4. Parallel and Discrete-Event Simulation Model of Fat-Tree Networks

The key components in a fat-tree network system are switches and processing nodes. In FatTreeSim, we use LP to model switch and processing nodes. FatTreeSim only focuses on the network topology and its related features and simplifies the hardware components such as I/O system, CPU, and memory. The processing node LP can be considered as a network interface card (NIC) in CODES system where detailed hardware models are provided. We also use an additional LP (App LP) type to model an application software, for example, an MPI process or MapReduce task. The purpose is to accurately capture the application layer behavior and thus quantitatively model its effects on the network layer. For example, a group of MPI processes running on terminals can issue a collective communication call that generates a burst of packets in the network layer. In FatTreeSim, switch LPs are classified as a core switch LP, intermediate-switch LP, and edge switch LP. This resembles a real fat-tree network system. Edge-switch LP connects to processing-node LP. The same group of switch LP and processing-node LP share the same address prefix. For the convenience of presentation, we use procedures to describe the typical events used in FatTreeSim and illustrate them in Figures 4, 5, 6, and 7.

The packet routing in FatTreeSim is based on the addressing system. A $m$-port, $n$-tree fat-tree network has a total of $m \cdot (m/2)^{(n-1)}$ processing nodes and $(2n-1) \cdot (m/2)^{(n-1)}$ switches. Each node LP is assigned a unique $n$-bit address. The first bit indicates the group number. $m$-port means the total number of groups is $m$. The rest $n-1$ bits vary from 1 to $(m-1)/2$. Thus, the total number of processing nodes inside each group is $(m/2)^{(n-1)}$. A switch LP is also assigned a unique $n$-bit address. The first bit also indicates the group number. The last bit of the address indicates the layer number, where 0 represents the core layer and $n-1$ represents the edge layer. The rest $n-2$ bits vary from 1 to $(m-1)/2$. Thus, the total number of switches in each layer is $2 \cdot (m/2)^{(n-1)}$ with the exception that the core layer has $(m/2)^{(n-1)}$ switches. The routing starts at the edge switch LP and iterates through all the layers. At any layer, if the

```
procedure GT                                          ▷ generate packet stream
    t =processing delay
    τ = rng(I)
    if RandomDestinationTraffic then
        dst = rng(maxnodeID)
        Generate packet (header contains dst )
    else if NearestNeighborTraffic then
        dst = neighborID
        Generate packet (header contains dst )
    else
        Unsupported traffic
    end if
    Call NSP procedure with t
    Call GT procedure with τ
end procedure
```

Fig. 4. Procedure GT.

```
procedure NSP                                         ▷ node send packet
    t = D/B + T_p
    dst =my connected router
    Call Procedure GT with t and dst
end procedure
```

Fig. 5. Procedure NSP.

```
procedure RFR                                         ▷ router receives flit
    t =processing delay
    Check flit dst
    Call RFS procedure with t
end procedure
```

Fig. 6. Procedure RFR.

```
procedure RFS                                         ▷ router sends flit
    Parse flit dst
    nextHop =ADDRESS(dst,flit)
    t = D/B + T_p
    Call RFR procedure with t and nextHop
end procedure
```

Fig. 7. Procedure RFS.

```
procedure ADDRESS                                          ▷ find next hop node
    Parse flit dst
    Get my address adr
    Find gcp address greatest common prefix (dst,adr)
    if gcp == L_c then
        Route down, hash packet header
        nextHop =ECMP()
    else if gcp < L_c then
        Route up, hash packet header
        nextHop =ECMP()
    else
        Error
    end if
    return nextHop
end procedure
```

Fig. 8. Procedure ADDRESS.

first $k$ bits of the destination node address matches the first $k$ bits of the address of the current switch, then the packet starts to go down to the lower layer of the switch or the processing node. Otherwise, the packet continues to go to upper layer switch. When packets go up, there are multi-paths to choose from. ECMP algorithm hashes the packet header and find the corresponding egress port based on the hash value. In Lin et al. [2004], the authors validated the routing algorithms with analytical proof and experiments. With the aforementioned scheme, the packet routing is based on table look-up rather than pre-allocation, which could save memory for storing LP state variables in FatTreeSim.

In ROSS and CODES, the LP is addressed through a global ID in the form an unsigned long integer. This is different from the bit-format address assigned to the LP in routing. Thus, we convert addresses between the two formats and guarantee the events are forwarded to the correct LPs. We leverage the internal event queue in ROSS to conveniently model the queuing effects of packets in the network system. Specifically, each LP has its own queue, and we use LPs to model the system components and thus control granularity of queuing effects of the system. As part of the ongoing work, we are working on building multiple channels and finite buffers to enhance FatTreeSim.

The most important event in an App LP is the packet generation event. We describe this event in Figure 4. GT procedure models the communication patterns of an application. As described in 3.2, FatTreeSim support two types of traffic: random destination and nearest neighbor. GT procedure calls itself with a random interval. The intervals applies to exponential distribution, therefore, the GT procedure is capable of generating a Poisson input stream.

The NSP procedure illustrates a packet has been generated in an App LP and is injected into the fat-tree network. NSP further triggers the flits generation event that models the protocol level details of network traffic. Users can customize the flit sizes to evaluate how different network configurations can affect the performance. When a switch receives a flit, it parses the flit header and calls the ADDRESS procedure to get the exact next-hop address. Routing algorithms such as ECMP is implemented in ADDRESS procedure. In this study, we only implemented and evaluated the ECMP. It is our future work to develop models for other routing algorithms and evaluate the performance under different applications at large scale.

At the processing node LP, the flits that belong to the same packet are assembled and then further forwarded to the destination App LP. We use additional events to model this process and the details are omitted in the discussion.

## 4. EXPERIMENTAL EVALUATION

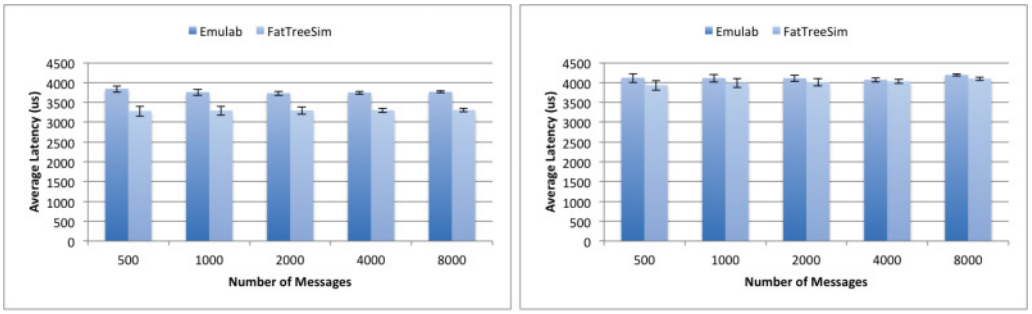
We have conducted extensive performance evaluation of FatTreeSim in terms of fidelity, scalability, and application running atop fat-tree models. To validate the accuracy of FatTreeSim, we established various physical network scenarios in Emulab, and compared the measurements against the simulation results (see Section 4.1). To evaluate the scalability, we ran FatTreeSim on Blue Gene/Q supercomputer with a variety of configurations up to 65,536 cores, with both conservative and optimistic parallel simulation modes. We quantitatively provide the guidelines to efficiently use the two modes in different scenarios (see Section 4.2). To demonstrate the usage of FatTreeSim, we conducted several Hadoop application simulation experiments using FatTreeSim on YARNsim [Liu et al. 2015] (see Section 4.3).

### 4.1. Accuracy Validation Using Emulab

Emulab is a network testbed that allows users to flexibly allocate the physical devices and virtual machine resources to build the desired networking experiment environment [Cutler et al. 2010]. To validate the fat-tree model, we constructed the physical fat-tree networks in Emulab with user-specified configurations, including topology, network bandwidth, message size, and developed network applications utilizing ECMP and MPI library for communication. We used the MPI Ping-Pong benchmark for all the experiments. The message size was set to 1,024 bytes to verify the MPI eager protocol. Similar to UDP, the eager protocol features a fire-and-forget communication pattern, in which no acknowledgment messages are generated. An MPI message smaller than 2,048 bytes can automatically trigger the eager protocol. This experiment setting guarantees that FatTreeSim has the correct configurations for each node, link and switch.

We conducted the accuracy evaluation by comparing the experimental results from both FatTreeSim and Emulab. We used an identical set of networking parameters for experiments running on both platforms to ensure that on an experiment-by-experiment basis, we were comparing precisely the same network scenario. FatTreeSim allows users to modify the fat-tree structure, including link bandwidth, topology, and traffic pattern and volume. These settings are used by system designers to test different fat-tree configurations, and therefore a comprehensive fidelity evaluation requires us to test FatTreeSim by varying all those parameters. Using Emulab to validate simulation models has its own limitation, because the hardware-based environment makes reproducibility impossible. However, it is still a feasible and widely acceptable way owning to the good controllability, flexibility, and functional fidelity. In addition, it is still technically impossible to configure supercomputer systems at the user level to valid simulation models.

*4.1.1. Number of Messages.* We first conducted experiments by varying the number of outgoing messages, from 500 to 8,000. Two traffic patterns, nearest neighbor and random destination, were tested. In FatTreeSim, we set up an exact fat-tree network with identical configurations used in Emulab. We repeated each set of experiments 10 times and calculated the average and the standard deviation of message latency. The experimental results with the nearest-neighbor test are reported in Figure 9(a). We observe that the standard deviation decreases when the number of messages grows, and its maximum value is 2.86% in a 500-message test. This demonstrates that the system noise has minimal impact on the experiments. We conducted identical experiments on FatTreeSim, in which we manually introduced random noise to match the real system.




(a) Nearest Neighbor Traffic Pattern. (b) Random Destination Traffic Pattern.

Fig. 9. Message latency comparison between Emulab measurements and FatTreeSim results using MPI Ping-Pong benchmark. The message size is 1,024 bytes.

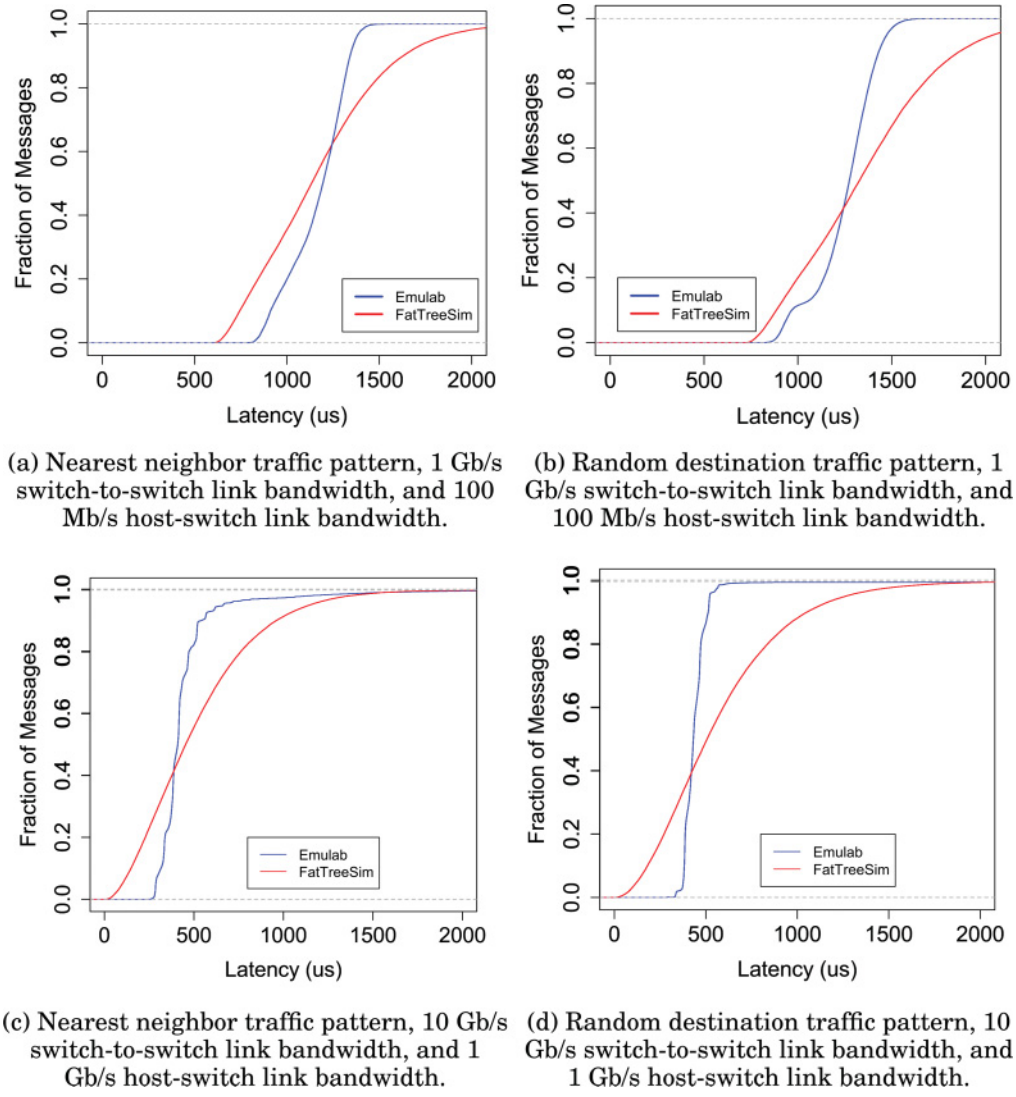
The error range of the simulation model is around 10–13% for all test cases. Figure 9(b) shows the experimental results with random destination traffic. We observe that the simulation can achieve a better accuracy with less than 3% error range for all test cases.

*4.1.2. Link Bandwidth.* A key parameter of a fat-tree network is the link bandwidth on each level, specifically, the host-to-switch and switch-to-switch link bandwidth at multiple layers of the fat-tree network. Therefore, we evaluated the model accuracy by varying the link bandwidths from 100Mb/s to 1Gb/s in the host-to-switch links and from 1Gb/s to 10Gb/s in the switch-to-switch links. The detailed results are provided in Figure 10.

We observe that for all the tested bandwidth configurations, the average message latency in FatTreeSim is within 10% difference from the message latency measured in Emulab. This shows that FatTreeSim can accurately account for the performance impact of altering link bandwidth. We also modified the communication pattern for each bandwidth configuration. For the random communication pattern, we used the same random number generator for both Emulab and FatTreeSim to determine a message's source and destination. The variation of traffic pattern has little impact to the accuracy of message latency.

*4.1.3. Fat-Tree Network Structure.* We also tested the model accuracy by varying fat-tree structures. We upgraded the four-port, two-tree test cases to the four-port, three-tree fat-tree network topology in Emulab and FatTreeSim. We observed that in both cases the FatTreeSim's calculated average latency is within 10% difference from the actual average latency measured in Emulab. In fact, the four-port, three-tree case in Emulab actually used multiple types of hardware, each with their own processing capabilities. FatTreeSim can still accurately estimate message latency even when the devices in Emulab were not exactly same.

To further evaluate the accuracy of FatTreeSim, we recorded the latency for each message from both the Emulab cluster and FatTreeSim and reported the results using the CDF plots in Figure 11. We observed that the curve for simulation is smoother than the curve for Emulab. This is attributed to the fact that we modeled only one outgoing buffer in each outgoing port. We also observed the gap between the two CDF curves in the high latency zone. We attribute the gap to the congestion model overuse in FatTreeSim. This gap explains the average latency error in Figures 10(a) and 10(b). In the four-port, three-tree test, we observed a better CDF curve match. However, FatTreeSim cannot generate the steep increase or plateau as observed in the Emulab




(a) Nearest neighbor traffic pattern, 1 Gb/s switch-to-switch link bandwidth, and 100 Mb/s host-switch link bandwidth.

(b) Random destination traffic pattern, 1 Gb/s switch-to-switch link bandwidth, and 100 Mb/s host-switch link bandwidth.

(c) Nearest neighbor traffic pattern, 10 Gb/s switch-to-switch link bandwidth, and 1 Gb/s host-switch link bandwidth.

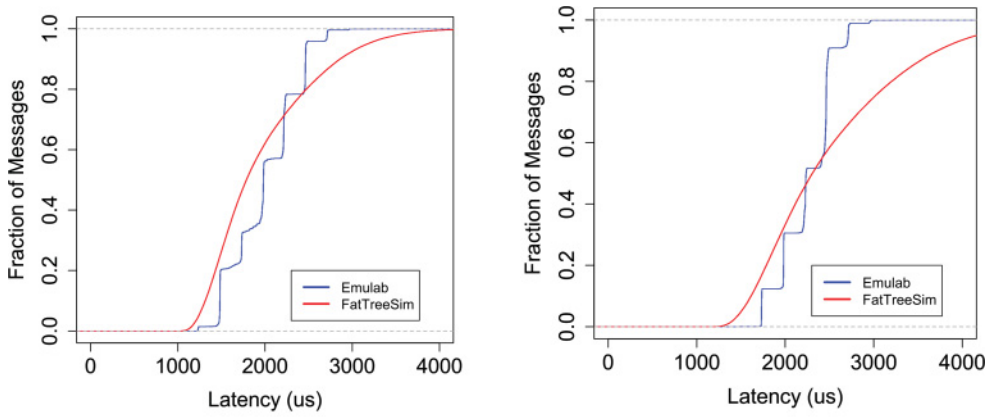
(d) Random destination traffic pattern, 10 Gb/s switch-to-switch link bandwidth, and 1 Gb/s host-switch link bandwidth.

Fig. 10. CDF of MPI Ping-Pong Test Message Latency. The message size is 1,024 bytes. The number of messages is 1,000. The number of processing nodes is 4. The number of switches is 3.

real system tests. One approach to model this effect is to introduce the multi-thread and multi-channel model in FatTreeSim.

### 4.2. Scalability Validation on Blue Gene/Q Supercomputing Systems

We conducted the strong-scaling experiment of FatTreeSim on Mira, a Blue Gene/Q supercomputing system in Argonne National Laboratory. As of November 2014, Mira ranks 4th in the top 500 lists. Mira consists of a total of 48 racks and 786,432 processors and is capable of 10 quadrillion calculations per second. The total memory of Mira is 768 terabytes. Each rack consists of 1,024 nodes and each node consists of 16 cores with a total of 16 gigabytes of shared memory. Users can choose to run on different modes, thus allocating different size of memory for each MPI process. The




(a) Nearest Neighbor Traffic Pattern. (b) Random Destination Traffic Pattern.

Fig. 11. CDF of MPI Ping-Pong Test Message Latency. The number of processing nodes is 16. The number of switches is 20. The message size is 1,024 bytes. The number of messages is 1,000. The switch-to-switch link bandwidth is 1Gb/s. The host-to-switch link bandwidth is 100Mb/s.

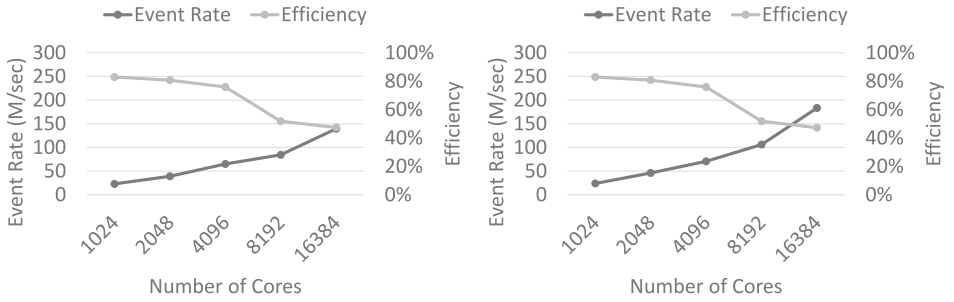
interconnection network is a 5D torus, which provides a fast collective communication for global reduce operations. This is ideal for ROSS to perform the synchronization algorithm. FatTreeSim leverages the fast 5D torus interconnection network and achieves a performance of 305 million events per second using 16,384 cores.

*4.2.1. Optimistic Mode.* We use two metrics, the committed event rate and the event efficiency, to evaluate the simulation performance of FatTreeSim in optimistic parallel simulation mode. In ROSS, the event efficiency determines the amount of useful work performed by the simulation. It is defined in Equation 2 [Carothers et al. 2000]:

$$efficiency = 1 - \frac{rolled\_back\_event}{total\_committed\_events}. \tag{2}$$

The simulation efficiency is inversely proportional to the number of rollbacks and is a rigorous indicator of the performance of a simulation system. The higher the efficiency, the faster the simulation performs. Another factor that affects the efficiency is the percentage of remote events, which is defined as the event transmitted between physical processes. The delay of remote event is unpredictable and can cause the logically erroneous events. The percentage of remote event is inherent to the model and usually increases with the increase of the number of physical processes. Global synchronization can effectively reduce the number of rollbacks. However, the global communication is usually expensive, especially in large-scale systems like Mira. There is a tradeoff in determining how frequently the simulation system performs the synchronization. ROSS uses global virtual time (gvt) interval and batch to control the frequency of gvt computation. FatTreeSim leverages the functionalities provided by ROSS and can achieve its peak performance in large-scale through parameters tuning and system configuration. The strategy for obtaining the optimal gvt interval and batch size for the Dragonfly network module is discussed in Mubarak et al. [2012].

Specifically, we configured a 128-port, 3-tree fat-tree network (i.e., switch radix = 128 and number of levels = 3) in FatTreeSim. The total number of processing-node LP and App LP was 524,288, and total number of switch LP was 20,480. We selected random destination traffic as it could generate the worst case scenario for parallel simulation because of the large portion of remote events. Each node continuously generated a packet stream and each packet randomly selected a destination. The time




(a) Packet arrival interval equals 200 ns.

(b) Packet arrival interval equals 400 ns.

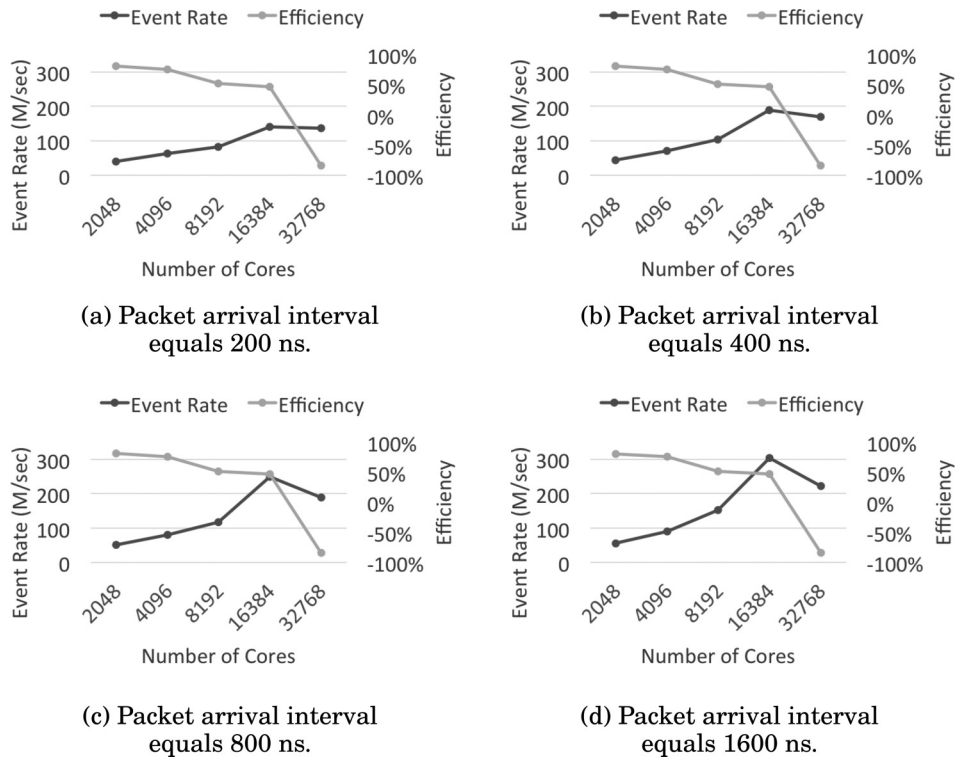
(c) Packet arrival interval equals 800 ns.

(d) Packet arrival interval equals 1600 ns.

Fig. 12. FatTreeSim Scalability Experiment on Blue Gene/Q. The fat-tree model consists of 524,288 processing nodes and 20,480 switches. The total number of committed events is 567 billion. In each subfigure, we varied the number of cores from 1,024 to 16,384 through running experiments on c1, c2, c4, c8, and c16 modes. From the top-left subfigure to the bottom-right subfigure, we varied the packet arrival interval from 200 ns to 1,600 ns. Experiments were conducted using 1 Blue Gene/Q rack. The traffic pattern was random destination.

interval between two packets followed the exponential distribution, thus the packets were modeled as a Poison stream. Our early work has pointed out that this interval has an impact on the simulation performance [Liu and Carothers 2011]. To perform the strong scaling experiments, we fixed the simulation size by setting the number of packets to 5,000 on each node. The total number of committed events is 567 billion.

We performed the experiments on Mira using one rack and two racks of nodes, respectively. Each Blue Gene/Q node has 16 processors and 16GB of shared memory, the job can run on six different modes, in which the each node can host 1, 2, 4, 8, 16, 32, and 64 MPI processes. In the last two modes, an MPI process runs as a hyper-thread, and the 32/64 threads share 16 physical cores. We focused on the first five modes, because a parallel simulation is usually memory intensive rather than CPU intensive. As a result, each MPI process got more memory. We varied the modes and packet arrival intervals and reported the performance of FatTreeSim on Mira in Figures 12 and 13. In the tests that used one rack of nodes, FatTreeSim nearly achieves a linear speedup up to 16,384 cores, and the peak event rate is 297 million per second. The efficiency decreased as the number of cores increased because of the increasing percentage of remote events. The maximum percentage we observed was 37%. Comparing horizontally, we can see that the event rate increased with the enlarged packet arrival




(a) Packet arrival interval equals 200 ns.

(b) Packet arrival interval equals 400 ns.

(c) Packet arrival interval equals 800 ns.

(d) Packet arrival interval equals 1600 ns.

Fig. 13. FatTreeSim Scalability Experiment on Blue Gene/Q. The fat-tree model consists of 524,288 processing nodes and 20,480 switches. The total number of committed events is 567 billion. In each subfigure, we varied the number of cores from 2,048 to 32,768 through running experiments on c1, c2, c4, c8, and c16 modes. From the top-left subfigure to the bottom-right subfigure, we varied the packet arrival interval from 200 to 1600ns. Experiments on the top subfigures were conducted using 2 Blue Gene/Q racks. The traffic pattern was random destination.

interval. This is because the intensive packet arrivals could cause the simulation engine to generate more out-of-order events, which significantly contributed to the total rollbacks. The takeaway message is that the performance of FatTreeSim will decrease when simulating a burst of communication or I/O operations. On the experiments that used two racks of nodes, we observe the negative efficiency when the scale reached 32,768 cores. This phenomenon is inherent in the model. The efficiency will increase if we (a) use the nearest-neighbor traffic instead of random destination traffic, and/or (b) increase the problem size of the simulation, for example, the total number of LPs, and/or (c) tune the gvt interval and the batch parameter in a finer-grained manner, and/or (d) perform a better mapping of LPs to MPI processes to better balance the workload. We have yet to use experiments to corroborate the above assertions. The gvt interval and batch used in the experiments were 32 and 8, respectively. The peak event observed in this set of experiments was 305 million per second using 16,384 cores.

*4.2.2. Conservative Mode.* Lookahead is necessary to allow concurrent processing of events with different time stamps unless optimistic event processing is used. Typically, optimistic simulation is more resilient to poor lookahead than conservative methods, and a good lookahead could significantly improve the simulation speed in conservative simulation. In this section, we applied different lookahead values in the conservative parallel simulation mode in FatTreeSim and evaluated the impact on performance.

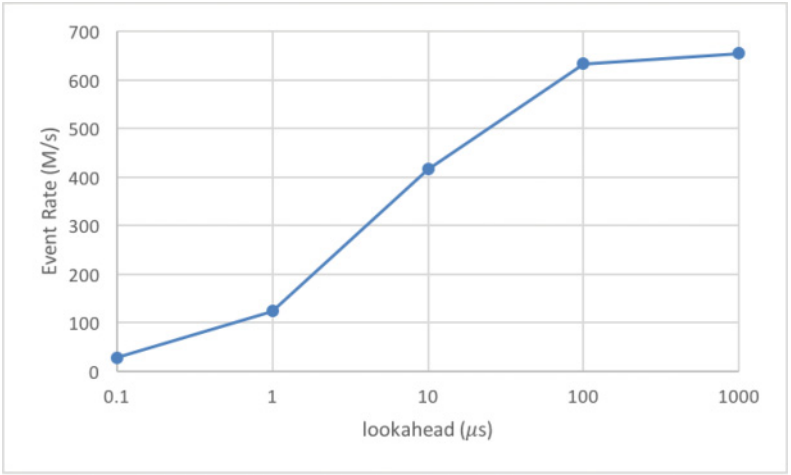


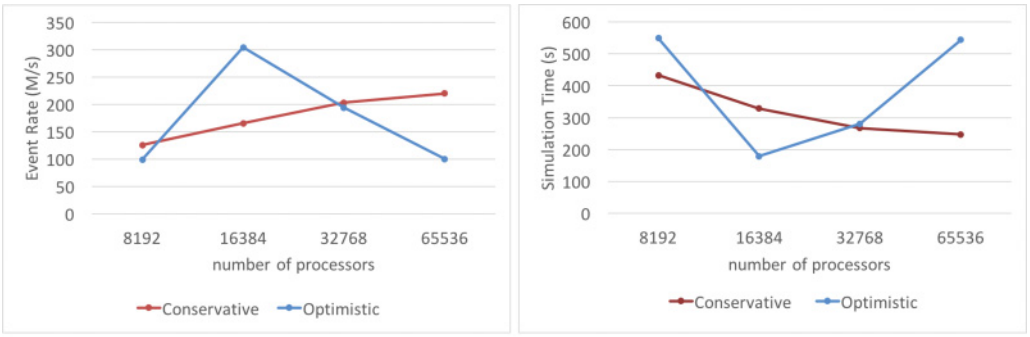

Fig. 14. Lookahead impact on FatTreeSim in conservative simulation mode. This set of experiments were executed on 2,048 nodes using c16 modes, that is, 32,768 cores. The fat-tree model was 128-3 tree and the traffic pattern was random destination.

The results are reported in Figure 14. An optimal lookahead value is often extracted by carefully analyzing the model configuration. Specifically, in FatTreeSim, the lookahead value is designed as the sum of the node processing time and the packet transmission time.

In Figure 14, the model had 524,288 nodes and 20,480 switches in total. The packet arrival interval was 200ns. The experiments used 2,048 nodes under the c16 mode, thus the total number of processors was 32,768. As the lookahead value increases, the event rate significantly increases because of the reduction in synchronization overhead. The peak performance reaches 655 million events per second with the 1ms lookahead value. Given the setting of one-switch-per-logic-process, a large lookahead like 1ms is not a realistic value for modeling switch delay in the context of HPC environment. FatTreeSim offers APIs to allow users to flexibly set delay values for various simulation experiments. The lookahead value can significantly grow if we group multiple entitles in the simulation model into a single logical process, or carefully extract the flow-level information from the transport layer and the domain knowledge from the application layer. We will investigate those topics in our future work.

*4.2.3. Performance Comparison between the Optimistic and Conservative Simulation Modes in FatTreeSim.* FatTreeSim supports two modes of simulation, but the optimal selection of the two modes is a challenging question. As observed in Figure 14, selecting the right simulation mode could save a significant amount of resources, for example, CPU hours in supercomputers.

Figure 15 depicts the FatTreeSim performance comparison in two simulation modes. We use two metrics to measure performance, the simulation time measured in seconds and event rate measured in millions per second. The lookahead was set to 1$\mu$s in the conservative mode. The performance of the optimistic mode varies significantly in large-scale, which is attributed to the increasing number of remote events exchanged across physical processors. In fact, the optimal performance is achieved by using 16K processors, and this result has been verified by the experiments shown in Figures 12 and 13. Achieving the optimal performance is equivalent to seeking for the maximum parallelism in optimistic mode. In this experiment, the maximum parallelism of the fat-tree model is dictated by the number of switches in each layer. In the 128 port,




(a) Measured Event Rate. (b) Measured Simulation Time.

Fig. 15. Simulation speed comparison between the optimistic and conservative simulation modes in FatTreeSim on Blue Gene/Q Mira using up to 65,536 cores.

3 tree model, each layer has a total of 8,192 switches, thus the optimal performance is related to this number. As illustrated in Figures 12 and 13, the best performance configuration uses 16K processors concurrently. Using more processors actually degrades the performance, and the efficiency becomes a negative value. In the extreme case of using 64K processors, the actual efficiency was −338.1%, which indicates that a significant amount of events had been rolled back. We recommend that the number of processors in use should be upper bounded by twice the number of switches in the optimistic simulation model. The size of the model determines the scalability of the simulation. In conservative mode, we observed a sub-linear increase in event rate and a decrease in simulation time. This is based on a model-dependent lookahead value. The performance could vary significantly, as already observed in Figure 14, or when we increase the complexity of the network model of FatTreeSim. If the lookahead value is large enough, then the performance of conservative simulation outperforms the optimistic simulation. Overall, both simulation modes can achieve a reasonably fast simulation while being capable of capturing the characteristic of a large-scale, fat-tree network.

### 4.3. Hadoop YARN Application Simulation Case Study Using FatTreeSim

YARNsim [Liu et al. 2015] is a comprehensive Hadoop YARN simulation system that is capable of evaluating both the hardware and software stack performance under a wide range of applications. YARNsim is built on CODES and ROSS and leverages the fast 5D torus network provided by Blue Gene/Q system for global reduction and synchronization. The performance of YARNsim has been evaluated through comprehensive Hadoop benchmarks and a bioinformatic application study [Liu et al. 2015]. The details regarding the design, implementation, and usage of YARNsim are beyond the scope of this article. In this section, we demonstrate the usability of FatTreeSim through various Hadoop application simulation in YARNsim. FatTreeSim serves as the network layer module in CODES, which is used by YARNsim to evaluate Hadoop benchmarks and applications. We recorded the YARNsim performance and compared against the results collected from HEC. HEC is a 51-node Sun-Fire-Linux-based cluster, consisting of one head node and 50 computing nodes. The head node uses Sun Fire X4240, equipped with dual 2.7GHz Opteron quad-core processors, 8GB memory, and 12 500GB, 7200 RPM SATA-II hard drives configured as a RAID5 disk array. The computing nodes are Sun Fire X2200 servers. Each node has dual 2.3GHz Opteron quad-core processors, 8GB memory, and one 250GB, 7200 RPM SATA hard drive. All

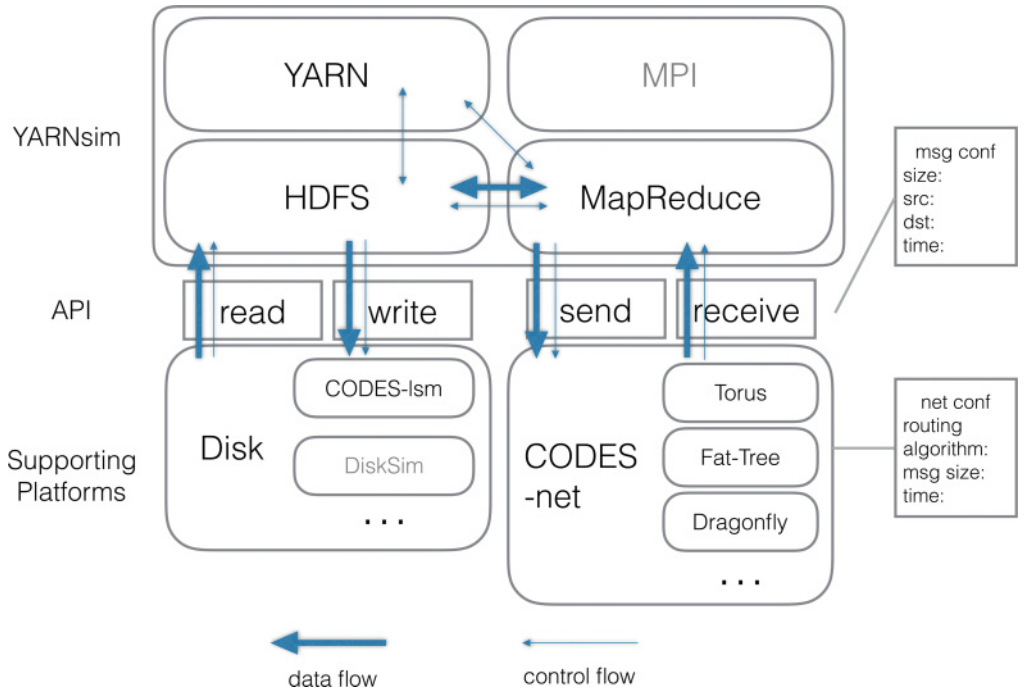



Fig. 16. YARNsim System Architecture with FatTreeSim Integration.

the 51 nodes are connected by gigabit Ethernet. We used Hadoop YARN 2.5.0 for all the experiments.

*4.3.1. YARNsim Integration.* The YARNsim experiments are based on the integration of YARNsim and FatTreeSim. Figure 16 depicts the system integration. At the top layer, YARNsim serves as the driver of the entire simulation experiments. YARNsim consists of several modules that model the corresponding components of Hadoop YARN. The details of YARNsim are beyond the scope of this article. Here, we focus on how YARNsim drives the simulation and communicates with other modules of the entire simulation system. YARNsim is a system model of Hadoop YARN and its success relies on a set of efficient and highly scalable hardware models. A key motivation of FatTreeSim is to provide the supporting hardware model for YARNsim, as shown in the bottom layer of Figure 16. CODES-net provides the network layer simulation models and CODES-lsm provides the disk layer simulation models. The APT layer connects YARNsim and FatTreeSim. The API defines the abstract behavior of the underlying system components. Each module, including FatTreeSim, implements the interfaces. The interfaces are defined by CODES-net, which contains a rich set of predefined network behaviors and suffices to support the hardware model requirements of YARNsim. In Figure 16, we present *send* and *receive* as two examples of how YARNsim and FatTreeSim are integrated. The network is also configurable. For example, if the network module implements multiple routing algorithms, users can specify a particular algorithm used for experiments. In Figure 16, we also list our ongoing work including using DiskSim as the disk module and implementing the MPI module in the YARNsim framework.

*4.3.2. Experimentation.* We chose Terasort and Wordcount benchmarks for the experiments, because they are often used to represent a class of Hadoop applications. To

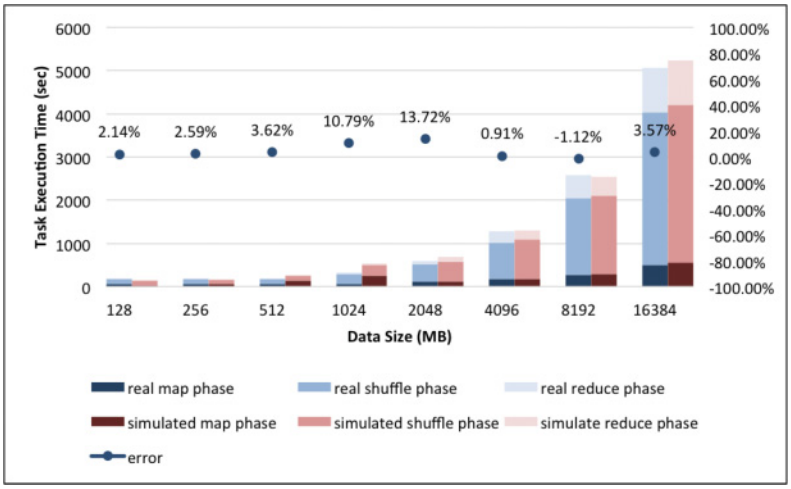



Fig. 17. Performance comparison of the Terasort benchmark between the real system measurements and the simulation results. The simulation uses FatTreeSim as the network module: the input data size varies from 128MB to 16GB; the number of nodes is 16. The blue stacks are the reported performance of each MapReduce phase on HEC, the red stacks are the reported performance of each MapReduce phase on YARNsim.

further analyze the application performance, we decomposed each job into three phases, the map phase, the shuffle phase and the reduce phase, assuming that the map phase and the reduce phase contain the merge-sort operations. In HEC, we used a total of 16 nodes and varied the input data size from 128MB to 16GB. To accurately record the performance of each phase in the real system, we leveraged the job history service provided by Hadoop, in which the detailed performance of each phase was reported. We collected and reported these numbers and compared them against the numbers collected from the YARNsim system. In YARNsim, we used the same configuration in HEC to configure the simulated clusters.

We reported the results of Terasort benchmark experiments in Figure 17. We compared the performance results collected from HEC and YARNsim. In most test cases, the error of the accumulated task execution time is within 5%. YARNsim can achieve a good accuracy in modeling Terasort benchmark and the Hadoop system with the FatTreeSim network module deployed. In Figure 18, we reported the Hadoop Wordcount benchmark performance results on both HEC and YARNsim. YARNsim can also achieve a good accuracy in modeling Wordcount benchmark and the Hadoop system with the FatTreeSim network module deployed. The observed error is within 10% for all the test cases.

In the field of bioinformatics, large dataset clustering is a challenging problem. Many biological scientists resort to Hadoop MapReduce for parallel processing solutions. Researchers from the University of Delaware have developed an octree based clustering algorithm for classifying protein-ligand binding geometries [Zhang et al. 2013]. The proposed method is implemented in Hadoop MapReduce and is divided into a two-phase MapReduce job. The geometry reduction and key generation is the first phase of a MapReduce job, where large datasets are read by the map tasks. The output of the first phase is the input of the second phase of a MapReduce job. An iterative octree based clustering algorithm is implemented as a chain of MapReduce jobs. This implies that the search has iterated to the deep level of the search tree. In the first phase, the output data size is about 1% of the input data size. Therefore, the MapReduce job spends most of the time on the map and the shuffle phase. To effectively model this application, we

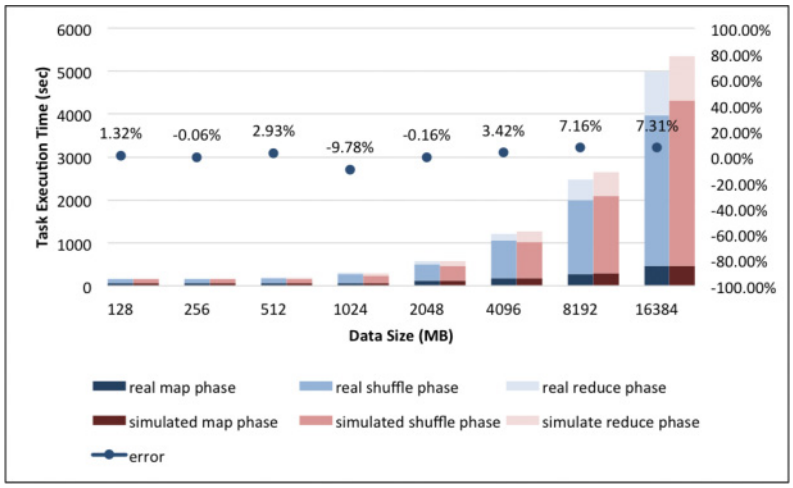



Fig. 18. Performance comparison of the Wordcount benchmark between the real system measurements and simulation results. The simulation uses FatTreeSim as the network module: the input data size varies from 128MB to 16GB; the number of nodes is 16. The blue stacks are the reported performance of each MapReduce phase on HEC, and the red stacks are the reported performance of each MapReduce phase on YARNsim.

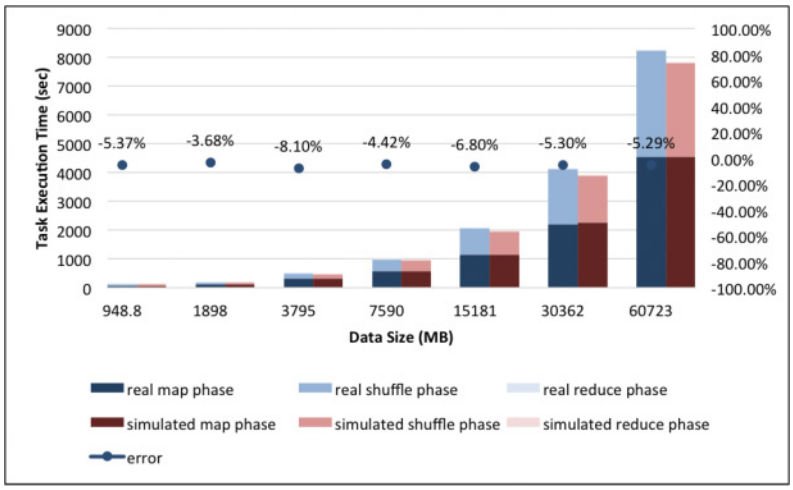



Fig. 19. Performance comparison of the Bio-application between the real system measurements and the simulation results. The simulation uses FatTreeSim as the network module: the input data size varies from 128MB to 16GB; the number of nodes is 16. The blue stacks are the reported performance of each MapReduce phase on HEC, and the red stacks are the reported performance of each MapReduce phase on YARNsim.

identified the sizes and locations of all data blocks in each phase and used them as inputs to the modeled MapReduce jobs. We varied the input file of protein geometry data from 948MB to 60GB and ran the experiments on HEC using 16 nodes. We also built a model for this clustering application and run it on YARNsim with a different configuration. The performance on HEC and YARNsim were reported in Figure 19. FatTreeSim also served as the underlying network topology in this set of experiments. The experimental results show that the error is within 10% for all the test cases. With FatTreeSim, we are capable of building a large-scale model of the data centers where the Hadoop system is deployed. The results indicate that it is reasonable to evaluate

and optimize a large-scale Hadoop application from the simulation perspective. It is our future work to continue investigating this topic.

## 5. RELATED WORK

There exists a plethora of works in modeling and simulating large-scale systems, each with a different focus. As part of the exascale co-design process, there is a growing interest in understanding how parallel systems software such as MPI/OpenMP and the associated supercomputing applications scale on extreme-scale computing systems. To this end, researchers have turned to parallel discrete-event simulation. For example, Perumalla's $\mu\pi$ [Perumalla and Park 2014] system allows MPI programs to be transparently executed on top of the MPI modeling layer and simulate the MPI messages. Each MPI task is realized as a thread in the underlying $\mu$silk simulator. Thus, $\mu\pi$ captures the true direct execution behavior across millions of MPI tasks that are part of a massively parallel application. Similar systems, such as BigSim [Zheng et al. 2010], have not achieved such a level of scaling. The Structural Simulation Toolkit (SST) uses a component-based parallel discrete-event model built on top of MPI. SST models a variety of hardware components including processors, memory and networks under different accuracy and details. To the best of our knowledge, none of these systems are designed to perform packet-level simulation of the underlying network at scale.

Wang et al. have built the Hadoop simulator MRperf [Wang et al. 2009]. MRperf use NS2 [Issariyakul and Hossain 2008] as the network module. NS2 is a well-established network simulator in the community because of the rich set of supported protocols and network functionalities. However, NS2 does not support parallel simulation and thus is not designed to simulate large-scale network models. The NS3 [NS3 2015] project supports conservative parallel simulation, and its performance on simulating extreme-scale networks has yet to be evaluated.

Researchers have focused on modeling large-scale network topology models, such as torus [Liu and Carothers 2011] and dragonfly [Mubarak et al. 2012]. These network models also leverage the functionalities provided by ROSS platform and have already been ported to the CODES [Cope et al. 2011] platform. Currently, the network models can support a wide range of application models that run on CODES. For example, Tang et al. [2014] built a data-aware resource scheduler on CODES. YARNsim [Liu et al. 2015] is the Hadoop YARN system simulator that runs on CODES. However, to the best of our knowledge, there is no large-scale model for fat-tree networks. FatTreeSim targets the fat-tree networks, which have already been widely used in the distributed computing community and are being considered as a promising candidate for interconnection networks in the next generation HPC systems.

## 6. CONCLUSIONS AND FUTURE WORK

In this article, we have presented FatTreeSim, a toolkit for modeling and simulating fat-tree networks at extreme scale. The core of FatTreeSim is a set of PDES-based models of fat-tree networks, supporting both optimistic and conservative parallel simulation. Fat-tree network topologies have been widely deployed in many traditional networks, and today they are facing new challenges with the advent of extreme-scale computing [Amarasinghe et al. 2009], where systems feature millions of physical cores and the potential billion-way concurrency. FatTreeSim is a timely work to equip system designers with the right tools to cope with deploying large-scale fat-tree networks in HPC systems and/or data centers. Our future work on FatTreeSim include developing a multi-channel model and a buffer management mechanism to improve model fidelity. We also plan to conduct extensive evaluation of many large-scale Hadoop applications using the FatTreeSim-integrated YARNsim platform.

## ACKNOWLEDGMENTS

The authors thank Dr. Misbah Mubarak, Dr. Jonathan Jenkins, and Dr. Robert Ross from Argonne National Laboratory for their valuable suggestions and help throughout this work.

Disclaimer: Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Maryland Procurement Office, AFOSR and DOE.

## REFERENCES

Dennis Abts and Bob Felderman. 2012. A guided tour through data-center networking. *Queue* 10, 5, Article 10. DOI:http://dx.doi.org/10.1145/2208917.2208919

Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.* 38, 4, 63. DOI:http://dx.doi.org/10.1145/1402946.1402967

Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, Berkeley, CA, 19.

Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. 1995. LogGP: Incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'95)*. ACM, New York. 95–105. DOI:http://dx.doi.org/10.1145/215399.215427

Saman Amarasinghe, Dan Campbell, William Carlson, Andrew Chien, William Dally, Elmootazbellah Elnohazy, Robert Harrison, William Harrod, Jon Hiller, Sherman Karp, Charles Koelbel, David Koester, Peter Kogge, John Levesque, Daniel Reed, Robert Schreiber, Mark Richards, Al Scarpelli, John Shalf, Allan Snavely, and Thomas Sterling. 2009. *1 ExaScale Software Study: Software Challenges in Extreme Scale Systems* (2009).

C.D. Carothers, D. Bauer, and S. Pearce. 2000. ROSS: A high-performance, low memory, modular time warp system. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'00)*. 53–60. DOI:http://dx.doi.org/10.1109/PADS.2000.847144

J. Cope, N. Liu, S. Lang, P. Carns, C. D. Carothers, and R. Ross. 2011. CODES: Enabling co-design of multilayer exascale storage architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies (WEST'11)*.

Cody Cutler, Mike Hibler, Eric Eide, and Robert Ricci. 2010. Trusted disk loading in the emulab network testbed. In *Proceedings of the 3rd International Conference on Cyber Security Experimentation and Test (CSET'10)*. USENIX Association, Berkeley, CA, 1–8.

Cisco. 20015. Global Cloud Index: Forecast and Methodology, 2013–2018. Retrieved from http://cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html (2015).

ClusterDesign.org. 2015. Real Cost Comparison of Fat-tree and Torus Networks, ClusterDesign.org. Retrieved from http://clusterdesign.org/2013/01/real-cost-comparison -of-fat-tree-and-torus-networks/.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51, 1, 107–113. DOI:http://dx.doi.org/10.1145/1327452.1327492

Christian E. Hopps and Dave Thaler. 2015. Multipath Issues in Unicast and Multicast Next-Hop Selection. Retrieved from https://tools.ietf.org/html/rfc2991.

Apache Hadoop. 2015. Homepage. Retrieved from http://hadoop.apache.org.

Teerawat Issariyakul and Ekram Hossain. 2008. *Introduction to Network Simulator NS2* (1 ed.). Springer.

Tonglin Li, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, and Ioan Raicu. 2013. ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'13)*. 775–787.

Xuan-Yi Lin, Yeh-Ching Chung, and Tai-Yi Huang. 2004. A multiple LID routing scheme for fat-tree-based InfiniBand networks. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*.. 11. DOI:http://dx.doi.org/10.1109/IPDPS.2004.1302913

N. Liu, C. Carothers, J. Cope, P. Carns, and R. Ross. 2012. Model and simulation of exascale communication networks. *J. Simulat.* 6, 4, 227–236. DOI:http://dx.doi.org/10.1057/jos.2012.4

Ning Liu and Christopher D. Carothers. 2011. Modeling billion-node torus networks using massively parallel discrete-event simulation. In *Proceedings of the 2011 IEEE Workshop on Principles of*

*Advanced and Distributed Simulation (PADS'11)*. IEEE Computer Society, Washington, DC, 1–8. DOI:http://dx.doi.org/10.1109/PADS.2011.5936761

Ning Liu, Xi Yang, Xian-He Sun, Jonathon Jenkins, and Robert Ross. 2015. YARNsim: Simulating hadoop YARN. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid'15)*.

Tatsuya Mori, Masato Uchida, Ryoichi Kawahara, Jianping Pan, and Shigeki Goto. 2004. Identifying elephant flows through periodically sampled packets. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC'04)*. ACM, New York, 115–120. DOI:http://dx.doi.org/10.1145/1028788.1028803

M. Mubarak, C. D. Carothers, R. Ross, and P. Carns. 2012. Modeling a million-node dragonfly network using massively parallel discrete-event simulation. In *Proceedings of the High Performance Computing, Networking, Storage and Analysis (SCC'12)*. 366–376. DOI:http://dx.doi.org/10.1109/SC.Companion.2012.56

NS3. 2015. Homepage. Retrieved from https://www.nsnam.org/.

Kalyan S. Perumalla and Alfred J. Park. 2014. Simulating billion-task parallel programs. In*Proceedings of the Summer Simulation Conference International Symposium on Performance Evaluation of Computer and Telecommunication Systems*.

Gartner Report. 2015. Homepage. Retrieved from http://www.gartner.com/newsroom/id/2313915 (2015).

S. Snyder, P. Carns, J. Jenkins, K. Harms, R. Ross, M. Mubarak, and C. Carothers. 2014. A case for epidemic fault detection and group membership in hpc storage systems. In *The 5th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS'14)*. New Orleans, LA, USA, 237–248.

Summit. 2015. Scale new heights. Discover new solutions. Retrieved from http://www.olcf.ornl.gov/summit/.

W. Tang, J. Jenkins, F. Meyer, R. B. Ross, R. Kettimuthu, L. Winkler, X. Yang, T. Lehman, and N. L. Desai. 2014. Data-aware resource scheduling for multicloud workflows: A fine-grained simulation approach. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom'14)*. Singapore, 887–892.

Guanying Wang, A. R. Butt, P. Pandey, and K. Gupta. 2009. A simulation approach to evaluating design decisions in mapreduce setups. In *Proceedings of the IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS'09)*. 1–11. DOI:http://dx.doi.org/10.1109/MASCOT.2009.5366973

Boyu Zhang, Daniel T. Yehdego, Kyle L. Johnson, Ming-Ying Leung, and Michela Taufer. 2013. Enhancement of accuracy and efficiency for RNA secondary structure prediction by sequence segmentation and MapReduce. *BMC Struct. Biol.* 13, Suppl 1, S3. DOI:http://dx.doi.org/10.1186/1472-6807-13-S1-S3

Dongfang Zhao, Da Zhang, Ke Wang, and Ioan Raicu. 2013. Exploring reliability of exascale systems through simulations. In *Proceedings of the 21st ACM/SCS High Performance Computing Symposium (HPC'13)*.

Dongfang Zhao, Zhao Zhang, Xiaobing Zhou, Tonglin Li, Ke Wang, Dries Kimpe, Philip Carns, Robert Ross, and Ioan Raicu. 2014. FusionFS: Toward supporting data-intensive scientific applications on extreme-scale distributed systems. In *Proceedings of IEEE International Conference on Big Data*.

G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale. 2010. Simulating large scale parallel applications using statistical models for sequential execution blocks. In *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS'10)*.

Received November 2015; revised May 2016; accepted August 2016