FatTreeSim: Modeling Large-scale Fat-Tree Networks for HPC Systems and Data Centers Using Parallel and Discrete Event Simulation

Ning Liu, Adnan Haider, Xian-He Sun, Dong Jin
Illinois Institute of Technology
10 West 31st Street
Chicago, IL 60616
{nliu8,ahaider3}@hawk.iit.edu,{sun,dong.jin}@iit.edu

ABSTRACT

Fat-tree topologies have been widely adopted as the communication network in data centers in the past decade. Nowadays, high-performance computing (HPC) system designers are considering using fat-tree as the interconnection network for the next generation supercomputers. For extreme-scale computing systems like the data centers and supercomputers, the performance is highly dependent on the interconnection networks. In this paper, we present FatTreeSim, a PDES-based toolkit consisting of a highly scalable fat-tree network model, with the goal of better understanding the design constraints of fat-tree networking architectures in data centers and HPC systems, as well as evaluating the applications running on top of the network. FatTreeSim is designed to model and simulate large-scale fat-tree networks up to millions of nodes with protocol-level fidelity. We have conducted extensive experiments to validate and demonstrate the accuracy, scalability and usability of FatTreeSim. On Argonne Leadership Computing Facility's Blue Gene/Q system, Mira, FatTreeSim is capable of achieving a peak event rate of 305 M/s for a 524,288-node fat-tree model with a total of 567 billion committed events. The strong scaling experiments use up to 32,768 cores and show a near linear scalability. Comparing with a small-scale physical system in Emulab, FatTreeSim can accurately model the latency in the same fat-tree network with less than 10% error rate for most cases. Finally, we demonstrate FatTreeSim's usability through a case study in which FatTreeSim serves as the network module of the YARNsim system, and the error rates for all test cases are less than 13.7%.

Categories and Subject Descriptors

I.6.4 [**Computing methodologies**]: Modeling and simulation

General Terms

Design, Experimentation, Performance, Measurement

SIGSIM-PADS'15, June 10–12, 2015, London, United Kingdom.
© 2015 ACM ISBN 978-1-4503-3583-6/15/06 $15.00.
DOI: http://dx.doi.org/10.1145/2769458.2769474.

Keywords

Fat-tree networks; Parallel discrete event simulation; Blue Gene/Q; Datacenter interconnection network; Supercomputer interconnection networks

1. INTRODUCTION

The growing demands of Internet Services have greatly propelled the development and deployment of data centers in the past decade. According to [4], the total number of data centers deployed will reach 8.6 million by the year 2017 and then start to decline. But the total capacity of the data center will continue to increase, which indicates the size of individual data centers will continue to grow. Today's leading data centers usually take up millions of square feet, host sub-millions of physical servers and consume million watts of energy. Communication network is a key component in data centers. According to the report from Cisco[2], 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. Therefore, we need an efficient data center communication network that is capable of running at extreme-scale and processing large volumes of application data. Fat-tree has been the mainstream networking topology adopted in data centers and will continue to dominate in the near future.

In the HPC community, there is a growing interest in understanding how parallel programs, including system software like MPI/OpenMP and scientific applications, scale in extreme large scale architectures. One key performance impact factor is also the interconnection communication network. In the past decades, torus network is widely adopted in HPC systems because of its low cost and high delivered performance. For example, the Blue Gene/L and P series and the CrayXT series use a 3-D torus network and the newly delivered Blue Gene/Q system uses a 5-D torus network. However, the torus network is inherently a blocking network, meaning the total number of available paths is smaller than the total demand, which can significantly exacerbate the network performance in communication bursts such as the MPI All-to-All and the Reduce communication. A fat-tree can provide the non-blocking feature with the 1:1 subscription ratio and keep the total cost within a reasonable level [7]. Oak Ridge National Laboratory has recently announced that the next generation OLCF supercomputer, SUMMIT, will adopt a fat-tree as its interconnection network [8].

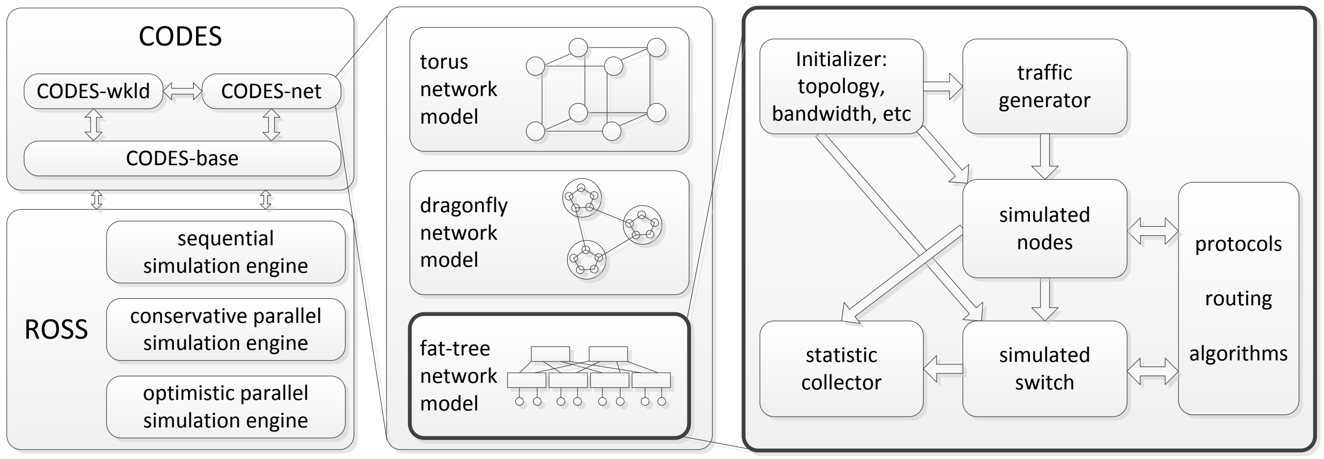

CODES
CODES-wkld
CODES-net
CODES-base
ROSS
sequential simulation engine
conservative parallel simulation engine
optimistic parallel simulation engine
torus network model
dragonfly network model
fat-tree network model
Initializer: topology, bandwidth, etc
traffic generator
simulated nodes
protocols routing algorithms
statistic collector
simulated switch

Figure 1: FatTreeSim system architecture in CODES ecosystem. CODES is based on ROSS and is comprised of multiple modules: CODES-base, CODES-workload and CODES-net. CODES-net is comprised of multiple network models. FatTreeSim is designed to be part of the CODES-net module and is the fat-tree network model illustrated in this Figure. FatTreeSim is comprised of multiple components: Initializer, Traffic Generator, Nodes, Switches, Protocols & Routing Algorithms and Statistic Collectors.

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 paper, 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) [14] and the Co-Design of Exascale Storage System (CODES) [15]. Fig. 1 illustrates the architecture of CODES.The CODES-workload [31] module provides the functionalities to model and simulate HPC and Cloud workload traces and directly applying existing traces to drive the underlying models. The CODES-net module provides multiple PDES-based networking models as well as a unified user interfaces that facilitate the use of the underlying networking models. The CODES-net includes four submodules: a torus network model [23], a dragonfly network model [28], a loggp[13] model, and a simple-net model. To date, many simulation systems [25, 32] that are based on CODES have successfully leveraged the functionalities provided by CODES-net. In this work, our contribution is to develop a highly scalable PDES-based fat-tree model, abbreviated as *FatTreeSim*, as shown in Fig 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 that is capable of modeling and simulating a fat-tree network with sub-million nodes in a time efficient manner. FatTreeSim has demonstrated a close-to-linear scalability on the Blue Gene/Q system and has achieved a peak event rate of 305 M/s. 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.
2. FatTreeSim models the protocol-level granularity in a fat-tree network with an ECMP routing algorithm. Its performance accuracy is validated through comparisons with the real system developed in Emulab, and results show that errors are all within 10% bounds.
3. We demonstrate the functionalities and compatibilities of FatTreeSim through a case study. We run YARNsim [25] with FatTreeSim and show that the network model can effectively capture the network traffic characteristics and enable the scalable simulation of MapReduce benchmark applications.

The remainder of the paper is organized as follows. We discuss the background and motivation in Section 2. We describe the fat-tree model in Section 3. Section 4 presents the validation experiments of the fat-tree model, and Section 5 discusses the related work. Closing remarks and future works are presented in Section 6.

2. BACKGROUND & MOTIVATION

In this section, we present the background information and motivation of developing FatTreeSim. Then we discuss two related systems, including the knowledge and techniques, used by FatTreeSim.

2.1 HPC & Data Centers and the Interconnection Networks

The design, evaluation and deployment of data center and HPC system is a systematic and time-consuming process. As the key component, the communication network has a significant impact on system performance. Large-scale data center and HPC system network architecture need to support a wide range of applications, each with different communication and I/O requirements. This paper concentrates on discussing an interconnection network candidate for HPC and date centers: the fat-tree network, and its model and simulation in large-scale. In distributed computing community, it is projected that a single data center can scale out to host millions of virtual machines or even physical servers and serve multi-millions jobs/tasks. The requirements for building a data center network at such a scale also differ with that of the traditional data centers. The communication network must guarantee the high availability and reliability, desirable bisection bandwidth, and support for multi-tenancy. To quantify the design trade-offs of a network at a scale, it is desirable to build a large scale simulation toolkit that is capable of evaluating different design points in an efficient and cost-effective manner. A fat-tree or folded-Clos topology is the conventional and yet still the most prevalent design choice for data center communication networks [2].

2.2 Hadoop

In 2003, Google first proposed the MapReduce system [17] and since then it has become the most well accepted programming models in the distributed computing community because of its simplicity and efficacy. While Google kept its MapReduce system proprietary, Apache Hadoop [1] is the most popular open-source implementation of MapReduce framework. According to a report from Gartner [3], 65% of the packaged data analytic applications will be built on Hadoop by 2015. Thus, it is necessary to evaluate the large-scale network performance under a wide range of Hadoop applications.

Because of Hadoop's popularity, we have built YARNsim [25]: a Hadoop YARN simulation system. Compared to other MapReduce simulation system [33, 26, 18, 9, 5], YARNsim is advantageous in that it is a parallel simulation system that is potentially capable of simulating extreme-scale Hadoop YARN systems. Similar systems are constrained, for example, MRperf uses ns-2 as its network module. While ns-2 has a rich set of functionalities that can simulate the network at fine-granularity, it lacks the mechanism to run in parallel, thus is constrained in terms of modeling large-scale system. YARNsim includes a comprehensive model for HDFS, which can mimic the I/O system behaviors of Hadoop YARN. However, YARNsim's capabilities are limited in that it lacks the support of a detailed parallel network model. We are motivated to develop FatTreeSim to support YARNsim in modeling and simulating Hadoop YARN at extreme-scale with minimal cost and good accuracy. We expect YARNsim and FatTreeSim to help the community to better understand the advantages and disadvantages of Hadoop system and quantitatively measure the trade-offs between different design points.

2.3 ROSS & CODES

FatTreeSim is based on two parallel simulation packages: ROSS and CODES. ROSS is a massively parallel discrete-event simulator that has demonstrated the ability to process billions of events per second by leveraging large-scale HPC systems. A parallel discrete-event simulation (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 global timestamp order while minimizing any processor synchronization overheads. Two well-established

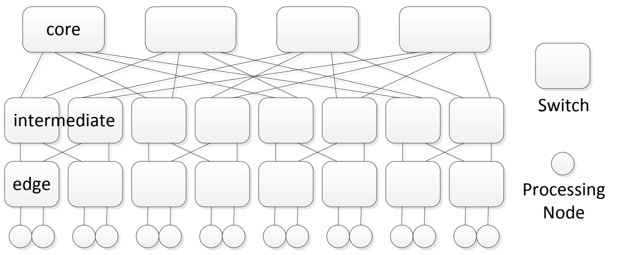


Figure 2: A fat-tree network with the total number of processing nodes equals to 16, the fat-tree height equals to 3, the number of edge switch equals to 8, and the number of core switch equals to 4.

approaches toward this goal are broadly called conservative processing and optimistic processing. ROSS supports both approaches. As shown in 4.2, FatTreeSim reaches the performance peak using the optimistic approach.

CODES [15] is a simulation system based on ROSS. Its goal is to enable the exploration and co-design of exascale storage systems by providing a detailed, accurate, and highly parallel simulation toolkit for exascale storage systems. Besides the two modules illustrated in Fig. 1, CODES also includes two modules CODES-bg and CODES-lsm. As seen in Fig 1, the current CODES-net toolkit includes two important submodules, the torus network model [29], and the dragonfly network model [28]. CODES is capable of simulating complex large-scale systems, e.g. FusionFS [36, 35], a distributed file system for both HPC and Cloud computing. The metadata in FusionFS is managed by ZHT [21], a zero hop distributed hash table service. Researchers from Illinois Institute of Technology have build models for the two systems and evaluated the performance at exascale. Tang et. al build a resource scheduler for multicloud workflows [32]. YARNsim [25] is the Hadoop YARN simulation system that aims to model and simulate extreme-scale Hadoop systems. However, current CODES-net module doesn't include a model for fat-tree network, which is the prevailing network topology used in current data centers. FatTreeSim targets the fat-tree network and is built in the CODES-net module. Therefore, it is compatible with any existing simulation systems built on CODES.

3. SIMULATING FAT-TREE NETWORK

In this section, we present the details related to the design, implementation and simulation of FatTreeSim. Specifically, we discuss the network topology and parameters, the simulated network traffic, routing algorithms and design considerations, and the discrete-event model.

3.1 Fat-Tree Topology

We illustrate a simple topology of the fat-tree network in Fig. 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

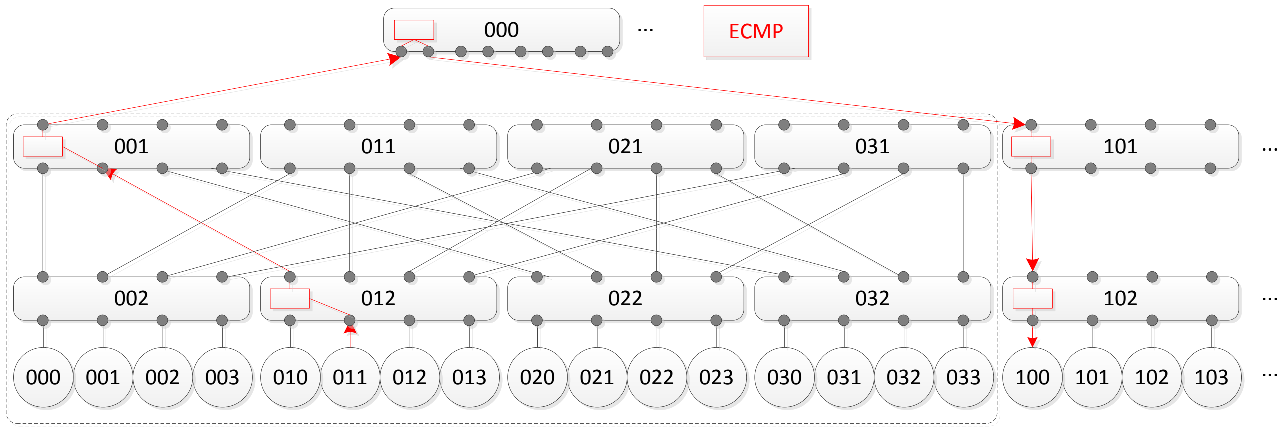


Figure 3: A packet routing in a 128 node fat-tree network. Fat-tree setup: m=8 n=3. The total number of switches is 80. The source node is 011, and the destination node is 100.

and the tree height. In general, a m-port n-tree fat-tree have the following characteristics [22, 11]: 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 4-port 3-tree fat-tree network is given in Fig. 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 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 [24]. In data centers, the network traffic exhibits a high degree of variability and is usually non-Gaussian [27]. In fact, a network that is only 10 percent utilized can see a lot of packet drops when running a web-search application [10].

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. This patterTo 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 [24, 28]. 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 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 paths multiplicity, equal-cost multi-paths routing algorithm (ECMP) is the widely used [11]. ECMP is a load balancing routing protocol based on RFC 2991 [19] that optimizes flows over multiple best paths to a single destination. ECMP applies load balancing routing on flows such as TCP or 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 [12]. A path through the network is called minimal path if no shorter path, less number of hops, exists. Different from other network topology, e.g. dragonfly, a fat-tree has multiple minimal paths. In Fig. 3, we illustrate an exemplar packet routing procedure in a 8-port 3-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 2 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, thus the delay can be described using equation 1.

$$T = H \cdot (D/B + T_p) + T_s \tag{1}$$

```
procedure GT                          ▷ generate packet stream
    $t$ =processing delay
    $\tau = rng(I)$
    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 $\tau$
end procedure
```

Figure 4: Procedure GT

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

Figure 5: Procedure NSP

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

Figure 6: Procedure RFR

Here, H is the number of hops the packet takes in its entire route. In fat-tree network this number usually equals to the number of hops in a minimal path. D is the packet size. B is the link bandwidth. Equation 1 assumes the bandwidth are equal between nodes or switches. In FatTreeSim, the link bandwidth is configurable through a customized configuration file. Thus equation 1 can be slightly modified so as to represent the most accurate cost. T_p is the average propagation delay on links. This parameter is also configurable in FatTreeSim.

3.4 Simulating the Fat-Tree Network

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, e.g. a 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 which 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 to 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 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 [22], 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.

The most important event in an App LP is the packet generation event. We describe this event in Fig. 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 an 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

```
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
```

Figure 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
```

Figure 8: Procedure ADDRESS

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 evaluate FatTreeSim from three aspects: its accuracy, scalability and usage. To verify that FatTreeSim can accurately model the real-world network traffic, we conducted extensive experiments on Emulab and compare the results against the simulation results. The detailed discussion is provided in section 4.1. To further demonstrate the scalability, we run FatTreeSim on Blue Gene/Q supercomputer with a variety of configurations. The details are presented in 4.2. Lastly, we demonstrate the usage of FatTreeSim through a use case study and its discussion is in section 4.3.

4.1 Fidelity Evaluation on Emulab

Emulab is a network testbed that allows users to flexibly allocate the physical devices as well as virtual devices to build the desired networking experiment environment [16]. Throughout the duration of the experiment, a user has complete control of the devices and thus is capable to configure the system with the desired parameters. We choose Emulab for FatTreeSim accuracy tests because we can configure the fat-tree network in a flexible manner. The maximum number of physical links an Emulab router can have is 4, thus we configured a 4-port 3-tree fat-tree network topology, in which we have full control of the physical nodes and links. We used the MPI Ping-Pong benchmark for the experiments. Here, the message size is set as 1,024 bytes because we want to use and verify the MPI eager protocol. Similar to UDP,

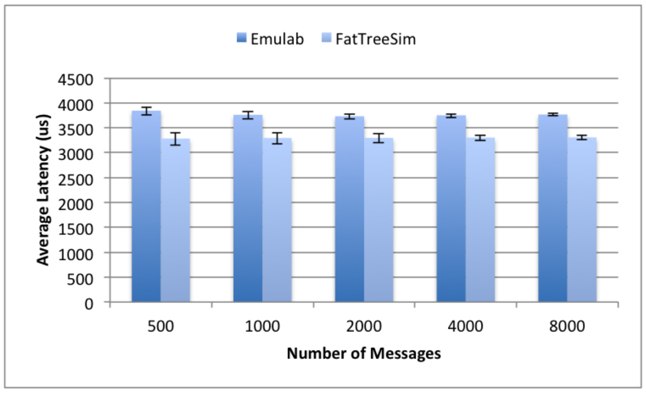


Figure 9: Latency comparison between Emulab measurements and FatTreeSim results using MPI Ping-Pong benchmark. The message size is 1,024 bytes. The total number of nodes is 16, and the total number of switches is 20. The traffic pattern is Nearest Neighbor.

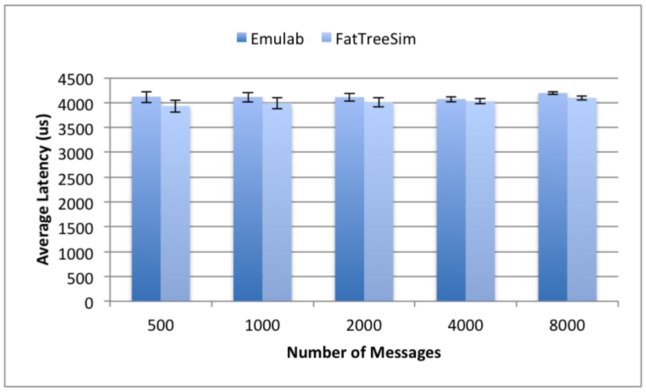
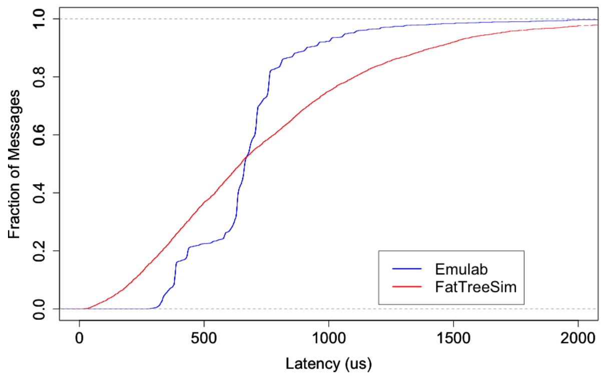


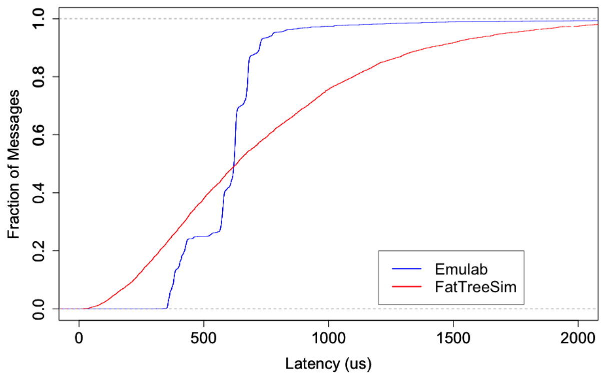
Figure 10: Latency comparison between Emulab measurements and FatTreeSim results using MPI Ping-Pong benchmark. The message size is 1,024 bytes. The total number of nodes is 16, and the total number of switches is 20. The traffic pattern is Random Destination.

The eager protocol features a fire-and-forget communication pattern in which no acknowledgment message is generated. An MPI message smaller than 2,048 bytes will automatically trigger the eager protocol. This experiment setting guarantees that FatTreeSim has the correct configurations for each node, link and switch.

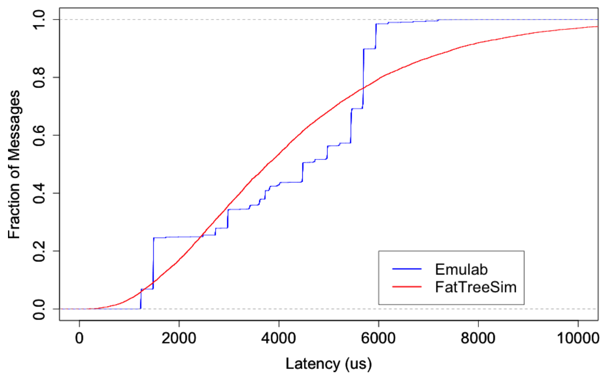
We used two traffic patterns, nearest neighbor and random destination, in the experiments. For each traffic pattern, we vary the number of outgoing messages from 500 to 8,000, and we repeat the test 10 times and calculated the standard deviation. In FatTreeSim, we set up an exact 4-port 3-tree fat-tree network with identical configurations. We repeat the experiments 10 times and calculate the average for each configuration. The experimental results for the nearest neighbor test is reported in Figure 9. As we can see, the standard deviation decreases with the increase of the number of messages 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



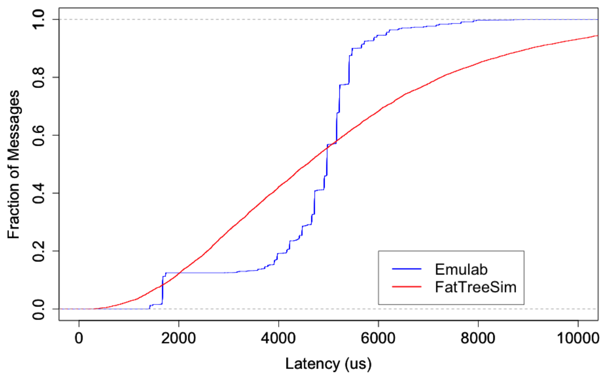
(a) CDF of Message Latency. The number of processing nodes is 8. The number of switches is 6. The traffic pattern is Nearest Neighbor.



(b) CDF of Message Latency. The number of processing nodes is 8. The number of switches is 6. The traffic pattern is Random Destination.



(c) CDF of Message Latency. The number of processing nodes is 16. The number of switches is 20. The traffic pattern is Nearest Neighbor.



(d) CDF of Message Latency. The number of processing nodes is 16. The number of switches is 20. The traffic pattern is Random Destination.

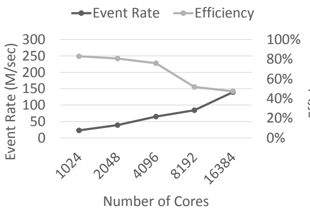
Figure 11: CDF of MPI Ping-Pong Test Message Latency. The message size is 1,024 bytes. The number of messages is 1,000.

experiments on FatTreeSim in which we manually introduce random noise to match the real system. There are, however, around 10-13% error in FatTreeSim for all test cases. In Figure 10, we report the experimental results for random destination traffic. Here, simulation can achieve a better accuracy and the error range is within 3% for all test cases.

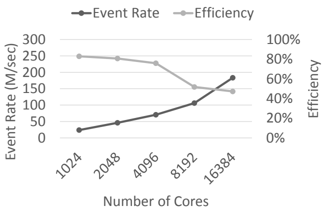
To further evaluate the accuracy of FatTreeSim, we record the latency for each message from both the Emulab cluster and FatTreeSim and report the results in the CDF plots. We used two different configurations: a 4-port 2-tree and a 4-port 3-tree. The message size is 1,024 bytes and the number of messages is 1,000 per node. In all experiments, we observed that the curve for simulation is much smoother than the curve for Emulab. This is attributed to the fact that we model only one outgoing buffer in each outgoing port. If multiple messages are sent through this port, congestion will occur and this single point queuing effect lead to a unique waiting time for each packet. Another observation is that there is a gap between the two CDF curves in the high latency zone in Figure 11a. We attribute this to the congestion model overuse in FatTreeSim. This gap explains the average latency error in Figure 9. In the 4-port 3-tree test, we observed better CDF curve match. However, FatTreeSim cannot generate the steep increase or plateau observed in the Emulab real system tests. As discussed earlier, one way to model this effect is to introduce the multi-thread and multi-channel model.

4.2 Validation on BG/Q

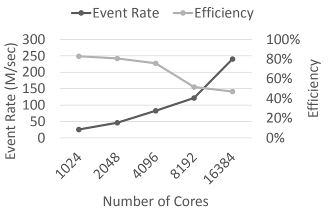
We conduct the strong-scaling experiment of FatTreeSim on Mira, a Blue Gene/Q supercomputing system in Argonne National Laboratory. As of Nov. 2014, Mira ranks 4_{th} in the top 500 lists. Mira consists of a total of 48 racks and 786,432 processors and is capable of 10 quadrillion calculations per second. 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 interconnection network is a 5-D torus which provides a fast collective communication for global reduce operations. This is ideal for ROSS to perform the optimistic synchronization algorithm. FatTreeSim leverages the fast 5-D torus interconnection network and achieves a performance of 305 million events per



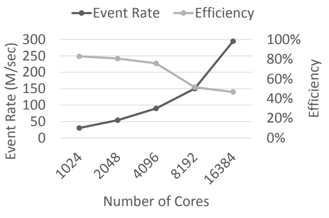
(a) Packet arrival interval equals 200 ns.



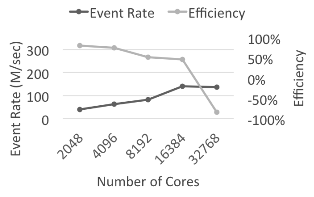
(b) Packet arrival interval equals 400 ns.



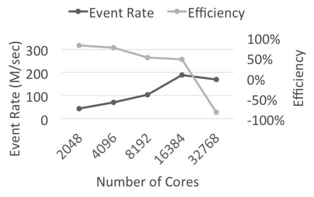
(c) Packet arrival interval equals 800 ns.



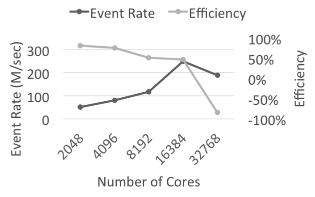
(d) Packet arrival interval equals 1600 ns.



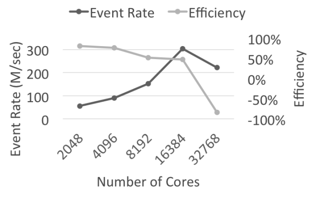
(e) Packet arrival interval equals 200 ns.



(f) Packet arrival interval equals 400 ns.



(g) Packet arrival interval equals 800 ns.



(h) Packet arrival interval equals 1600 ns.

Figure 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 top subfigure, we vary the number of cores from 1,024 to 16,384 through running experiments on c1, c2, c4, c8 and c16 modes. From top-left subfigure to top-right subfigure, we vary the packet arrival interval from 200 ns to 1,600 ns. Experiments on the top subfigures are conducted using 1 Blue Gene/Q rack. In each bottom subfigure, we vary the number of cores from 2,048 to 32,768 through running experiments on c1, c2, c4, c8 and c16 modes. From top-left subfigure to top-right subfigure, we vary the packet arrival interval from 200 ns to 1600 ns. Experiments on the top subfigures are conducted using 2 Blue Gene/Q racks. The traffic pattern is random destination.

second using 32,768 cores.

We use two metrics: the committed event rate and the event efficiency to evaluate the simulation performance of FatTreeSim. In ROSS, the event efficiency determines the amount of useful work performed by the simulation. It is defined in Equation 2 [14]:

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

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 when performing the scaling experiment. Global synchronization can effectively reduce the number of rollbacks, however, this 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 gvt-interval and batch to control the frequency of global virtual time 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 [28].

Specifically, we configured a 128-port 3-tree fat-tree network in FatTreeSim. The total number of processing-node LP and App LP is 524,288 respectively, and total number of switch LP is 20,480. We select random destination traffic as it can generate the worst case scenario for parallel simulation for its large percentage of remote event. Each node continuously generates a packet stream and each packet randomly selects a destination. The time interval between two packets applies to the exponential distribution, thus the packet is a Poison stream. The author in [24] has pointed out that the interval also has an impact on the simulation performance. To perform the strong scaling experiment,we fixed the simulation size through setting the number of packets to 5,000 on each node. The total committed event is 567 billion.

We perform the experiments on Mira using 1 rack and 2 rack of nodes respectively. Each Blue Gene/Q node has 16 processors and 16 gigabytes of shared memory, the job can run on 6 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 32/64 threads share 16 physical cores. We focus on the first 5 modes because, a parallel simulation is usually memory intensive rather than CPU intensive. Thus, each MPI process can get more memory. We vary the modes and packet arrival intervals and report the performance of FatTreeSim on Mira in details in Figure 12. In the tests that use 1 rack of nodes, FatTreeSim nearly achieves a linear speedup up to 16,384 cores, the peak event rate is 297 million per second. The efficiency decreases as the number of cores increases, this is because of the increase of remote event percentage. The maximum percentage we observed is 37%. Comparing horizontally, we can see that the event rate increases with the enlarged packet arrival interval. This is because the intensive packet arrivals can cause the simulation engine to generate more out of order events which contributes to the total rollbacks. The takeaway here is that the performance of FatTreeSim will decrease on simulating a burst of communication or I/O operations. On the experiments that use 2 racks of nodes, we start to observe negative efficiency when the scale reaches 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; b) increase the problem size of the simulation, e.g. total number of LPs; c) tune the gvt-interval and batch parameter in a fine-grained manner; d) perform a better mapping of LPs to MPI processes so as to better balance the wordload. We have yet to use experiments to corroborate the above assertions. The gvt-interval and batch used in the experiments are 32 and 8 respectively. The peak event observed in this set of experiments is 305 million per second using 16,384 cores.

4.3 Case Study: YARNsim

YARNsim [25] 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 leverage the fast 5-D torus network provided by Blue Gene/Q system for global reduction and synchronization. In [25], the performance of YARNsim is evaluated through comprehensive Hadoop benchmarks and a bioinformatic application study. The details regarding the design, implementation and usage of YARNsim are beyond the scope of this paper. In this experiment, we want to demonstrate the usability of FatTreeSim through running YARNsim using FatTreeSim. We perform the Hadoop application simulation in YARNsim. Here, FatTreeSim serves as the network layer module in CODES and helps YARNsim in evaluating Hadoop benchmarks and application. We record the YARNsim performance and compare it against the results collected from HEC. HEC is a 51-node Sun Fire Linux- based cluster, in which there are one head node and 50 computing nodes. The head node was Sun Fire X4240, equipped with dual 2.7 GHz Opteron quad-core processors, 8GB memory, and 12 500GB 7200RPM SATA-II drives configured as RAID5 disk array. The computing nodes were Sun Fire X2200 servers, each node with dual 2.3GHz Opteron quad-core processors, 8GB memory, and a 250GB 7200RPM SATA hard drive. All 51 nodes are connected through Gigabit Ethernet. We use Hadoop YARN 2.5.0 for all experiments.

We choose Terasort and Wordcount benchmarks for the experiments because they are widely accepted and can represent a class of Hadoop applications. To further analyze the application performance, we decompose each job to three phases, the map phase, shuffle phase and reduce phase, assuming map phase and reduce phase contains the merge-sort operations. In HEC, we use a total of 16 nodes and vary the input data size from 128MB to 16GB. To accurately record the performance of each phase in the real system, we leverage the job history service provided by Hadoop, in which the detailed performance of each phase is reported. We collect and report these numbers and compare them against the numbers collected from the YARNsim system. In YARNsim, we use the same configuration as in HEC for configuring the simulated clusters

We report the results of Terasort benchmark experiment

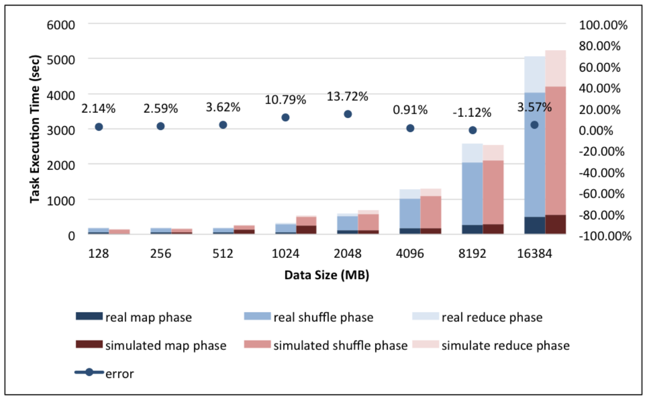


Figure 13: Performance Comparison of Terasort Benchmark between real system and simulation. The simulation uses FatTreeSim as the network module: input data size varies from 128MB to 16GB; number of nodes is 16. Blue stacks are the reported performance of each MapReduce phase on HEC, red stacks are the reported performance of each MapReduce phase on YARNsim.

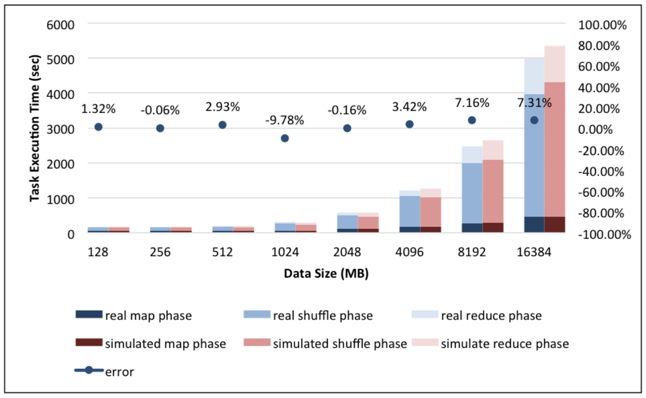


Figure 14: Performance Comparison of Wordcound Benchmark between the real system measurements and simulation results. The Simulation uses FatTreeSim as the network module: the input data size varies from 128MB to 16GB; the number of nodes is 16. The blue stacks are the reported performance of each MapReduce phase on HEC, and the red stacks are the reported performance of each MapReduce phase on YARNsim.

in Figure 13. Here we compare the performance results from both HEC and YARNsim. In most test cases the error of the accumulated performance 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 14, we report the Hadoop Wordcount benchmark performance results on both HEC and YARNsim. As we can see, YARNsim can also achieve a good accuracy in modeling Wordcount benchmark and the Hadoop system with the FatTreeSim network module deployed. The observed error rate is within 10% for all test cases.

In the field of bioinformatics, large dataset clustering is a

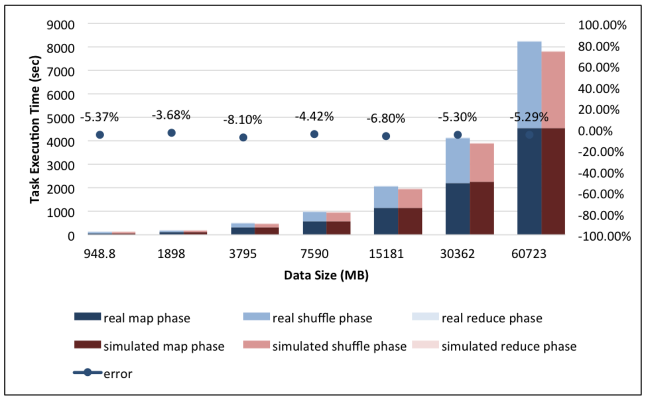


Figure 15: Performance Comparison of Bio-application between real system measurements and simulation results. The Simulation uses FatTreeSim as the network module: the input data size varies from 128MB to 16GB; the number of nodes is 16. The blue stacks are the reported performance of each MapReduce phase on HEC, and the red stacks are the reported performance of each MapReduce phase on YARNsim.

challenging problem. Many biological scientists resort to Hadoop MapReduce for large scale and parallel processing solutions. In [34], researchers from the University of Delaware have developed an octree based clustering algorithm for classifying protein-ligand binding geometries. The proposed method is implemented in Hadoop MapReduce and is divided into a two-phase MapReduce job. The geometry reduction and key generation are the first phase MapReduce job where large datasets are read by the map tasks. The output of the first phase is the input of the second phase MapReduce job. Here, iterative octree based clustering algorithm is implemented as a chain of MapReduce jobs representing 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. Thus the MapReduce job spends most of the time on the map and the shuffle phase. To effectively model this application, we identify the sizes and locations of all data blocks in each phase and use them as input to the modeled MapReduce jobs. We vary the input file of protein geometry data from 948MB to 60GB and run the experiments on HEC using 16 nodes. We also build a model for this clustering application and run it on YARNsim with different configuration. The performance on HEC and YARNsim are reported in Fig 15. Here, FatTreeSim also serves as the underlying network topology. The performance results show that the error is within 10% for all test cases. With FatTreeSim, we are capable of building a large-scale model of the data centers where the Hadoop system is deployed. Thus, it is potentially possible to evaluate and optimize a large-scale Hadoop application from a simulation perspective. It is our future work to continue on 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, Perumallas $\mu\pi$ [30] 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 μsilk 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 [37], 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, neither of these systems performs packet-level simulations of the underlying network at scale. Instead, the focus of their research is application computation performance with the hardwares abstracted away.

Wang et al. have built the Hadoop simulator MRperf [33]. MRperf use NS2 [20] as the network module. NS2 is a well established network simulator in the community due its rich set of functionalities, however, NS2 cannot run in parallel and thus is constrained in terms of modeling large-scale network. The new NS3 [6] project uses the conservative parallel simulation and its performance on large-scale has yet to be evaluated.

Researchers have focused on modeling large-scale network model for topologies like torus [24] and dragonfly [28]. These network models also leverage the functionalities provided by ROSS platform and have already been ported to the CODES [15] platform. Currently, the network models can support a wide range of application models that run on CODES. For example, Tang et al. [32] build a data-aware resource scheduler on CODES. YARNsim [25] 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 network, which has already been widely used in the distributed computing community and is being considered as a network candidate for the next generation HPC system.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we present FatTreeSim, a fat-tree network simulation system built on ROSS and CODES. We discuss the design and implementation of FatTreeSim and validate its accuracy, scalability and usability through extensive experiments. Specifically, we run FatTreeSim on the Argonne Leadership Computing Facility's Blue Gene/Q system, Mira, and demonstrates a close-to-linear scalability up to 32,768 cores. We also configured a 4-port 3-tree fat-tree network in EmuLab and compared with FatTreeSim results using the MPI Ping-Pong benchmark. The experimental results show that the error rate of average latency is within 10%. Finally, we run YARNsim, a Hadoop YARN simulation system with FatTreeSim to test the Terasort and Wordcount benchmarks and a bio-application. The experimental results show that FatTreeSim can help YARNsim to accurately model Hadoop benchmarks as well as real system applications.

Fat-tree is an important network topology that has been widely used in the community of parallel and distributed computing. Nowadays, fat-tree networks face new challenges with the advent of the era of extreme-scale computing, when 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. We plan to focus on the following issues in the future: a) To increase the accuracy of FatTreeSim, we plan to augment the existing system with a multi-channel model and a buffer management mechanism; b) to conduct extensive experiments on a Blue Gene supercomputer at even larger scale to find the optimal system configuration to maximize the model scalability; c) to test YARNsim on FatTreeSim with large-scale Hadoop applications, whose results can be used for large-scale Hadoop system optimization.

Acknowledgements

The authors would like to thank Dr. Misbah Mubarak, Dr. Jonathan Jenkins, and Dr. Robert Ross from Argonne National Laboratory for their valuable suggestions and help throughout this work. This material is based upon work supported byt the Maryland Procurement Office under Contract NO. H98230-14-C-0141. This also 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 DE-AC02-06CH11357[1].

7. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org. [Last accessed May 2015].

[2] Cisco Global Cloud Index: Forecast and Methodology, 2013-2018. http://cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html. [Last accessed November 2014].

[3] Gartner Report. http://www.gartner.com/newsroom/id/2313915. [Last accessed May 2015].

[4] IDC: Amount of World Data Centers to Start Declining in 2017. http://www.datacenterknowledge.com/archives/2014/11/11/idc-amount-of-worlds-data-centers/-to-start-declining-in-2017/. [Last accessed November 2014].

[5] Mumak: Map-Reduce Simulator. https://issues.apache.org/jira/browse/MAPREDUCE-728. [Last accessed May 2015].

[6] ns-3. https://www.nsnam.org/. [Last accessed May 2015].

[7] Real Cost Comparison of Fat-tree and Torus Networks | ClusterDesign.org. http://clusterdesign.org/2013/01/real-cost-comparison-of-fat-tree-and-torus-networks/. [Last accessed May 2015].

[8] Summit. Scale new heights. Discover new solutions. http://www.olcf.ornl.gov/summit/. [Last accessed May 2015].

[1]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.

[9] Yarn Scheduler Load Simulator (SLS). http://hadoop.apache.org/docs/r2.4.1/hadoop-sls/SchedulerLoadSimulator.html. [Last accessed May 2015].

[10] D. Abts and B. Felderman. A guided tour through data-center networking. *Queue*, 10(5):10:10–10:23, May 2012.

[11] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, Oct. 2008.

[12] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, Apr. 2010. USENIX Association.

[13] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model&Mdash;One Step Closer Towards a Realistic Model for Parallel Computation. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '95, pages 95–105, New York, NY, USA, July 1995. ACM.

[14] C. Carothers, D. Bauer, and S. Pearce. ROSS: a high-performance, low memory, modular time warp system. In *Fourteenth Workshop on Parallel and Distributed Simulation, 2000. PADS 2000. Proceedings*, pages 53–60, Bologna, Italy, May 2000.

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

[16] C. Cutler, M. Hibler, E. Eide, and R. Ricci. Trusted disk loading in the emulab network testbed. In *Proceedings of the 3rd International Conference on Cyber Security Experimentation and Test*, CSET'10, pages 1–8, Berkeley, CA, USA, Aug. 2010. USENIX Association.

[17] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008.

[18] S. Hammoud, M. Li, Y. Liu, N. Alham, and Z. Liu. MRSim: A discrete event based MapReduce simulator. In *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 6, pages 2993–2997, Yantai, China, Aug. 2010.

[19] C. E. Hopps and D. Thaler. Multipath Issues in Unicast and Multicast Next-Hop Selection. https://tools.ietf.org/html/rfc2991. [Last accessed May 2015].

[20] T. Issariyakul and E. Hossain. *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[21] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang, and I. Raicu. ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table. IPDPS '13, pages 775–787, Boston, MA, May 2013.

[22] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang. A multiple

LID routing scheme for fat-tree-based InfiniBand networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 11–, Santa Fe, New Mexico, Apr. 2004.

[23] N. Liu, C. Carothers, J. Cope, P. Carns, and R. Ross. Model and simulation of exascale communication networks. *Journal of Simulation*, 6(4):227–236, Nov. 2012.

[24] N. Liu and C. D. Carothers. 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, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.

[25] N. Liu, X. Yang, X.-H. Sun, J. Jenkins, and R. Ross. Yarnsim: Simulating hadoop yarn. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '15, Shenzhen, China, May 2015.

[26] Y. Liu, M. Li, N. K. Alham, and S. Hammoud. HSim: A MapReduce Simulator in Enabling Cloud Computing. *Future Generation Computer Systems*, 29(1):300–308, Jan. 2013.

[27] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying elephant flows through periodically sampled packets. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 115–120, New York, NY, USA, Nov. 2004. ACM.

[28] M. Mubarak, C. Carothers, R. Ross, and P. Carns. Modeling a Million-Node Dragonfly Network Using Massively Parallel Discrete-Event Simulation. In *2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*, pages 366–376, Washington, DC, USA, Nov. 2012.

[29] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns. A case study in using massively parallel simulation for extreme-scale torus network codesign. In *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '14, pages 27–38, New York, NY, USA, 2014. ACM.

[30] K. S. Perumalla and A. J. Park. Simulating billion-task parallel programs. In *Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2014), International Symposium on*, pages 585–592, Monterey, CA, USA, July 2014.

[31] S. Snyder, P. Carns, J. Jenkins, K. Harms, R. Ross, M. Mubarak, and C. Carothers. 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 (PMBS14).*, pages 237–248, New Orleans, LA, USA, Nov. 2014. Springer International Publishing.

[32] W. Tang, J. Jenkins, F. Meyer, R. B. Ross, R. Kettimuthu, L. Winkler, X. Yang, T. Lehman, and N. L. Desai. 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)*, pages 887–892, Singapore, Dec. 2014.

[33] G. Wang, A. Butt, P. Pandey, and K. Gupta. A simulation approach to evaluating design decisions in MapReduce setups. In *IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*, MASCOTS '09, pages 1–11, London, UK, Sept. 2009.

[34] B. Zhang, D. T. Yehdego, K. L. Johnson, M.-Y. Leung, and M. Taufer. Enhancement of accuracy and efficiency for RNA secondary structure prediction by sequence segmentation and MapReduce. *BMC Structural Biology*, 13(Suppl 1):S3, Nov. 2013.

[35] D. Zhao, D. Zhang, K. Wang, and I. Raicu. Exploring reliability of exascale systems through simulations. In *Proceedings of the High Performance Computing Symposium*, HPC '13, pages 1:1–1:9, San Diego, CA, USA, 2013. Society for Computer Simulation International.

[36] D. Zhao, Z. Zhang, X. Zhou, T. Li, K. Wang, D. Kimpe, P. Carns, R. Ross, and I. Raicu. Fusionfs: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In *2014 IEEE International Conference on Big Data*, pages 61–70, Washington, DC, Oct 2014.

[37] G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale. Simulating Large Scale Parallel Applications using Statistical Models for Sequential Execution Blocks. In *Proceedings of the 16th International Conference on Parallel and Distributed Systems*,