Reconfigurable Network Testbed for Evaluation of Datacenter Topologies

William Clay Moody
Computer Science Division,
School of Computing
Clemson University
Clemson, SC
wcm@clemson.edu

Jason Anderson
Computer Science Division,
School of Computing
Clemson University
Clemson, SC
jwa2@clemson.edu

Kuang-Ching Wang
Department of Electrical and
Computer Engineering
Clemson University
Clemson, SC
kwang@clemson.edu

Amy Apon
Computer Science Division,
School of Computing
Clemson University
Clemson, SC
aapon@clemson.edu

ABSTRACT

Software-defined networking combined with distributed and parallel applications has the potential to deliver optimized application performance at runtime. In order to investigate this enhancement and design future implementation, a datacenter with a programmable topology integrated with application state is needed. Towards this goal, we introduce the Flow Optimized Route Configuration Engine (FORCE). The FORCE is an emulated datacenter testbed with a programmable interconnection controlled by an SDN controller. We also utilize Hadoop as a case study of distributed and parallel applications along with a simulated Hadoop shuffle traffic generator. The testbed provides initial experimental evidence of support to our hypothesis for future SDN research. Our experiments on the testbed show a difference in application runtime a factor of over 2.5 times on shuffle traffic for Hadoop MapReduce jobs and the potential for significant speedup in warehouse scale data centers.

Categories and Subject Descriptors

C.2.1 [**Computer-Communications Networks**]: Network Architecture Design—*Network topology*; C.2.4 [**Computer-Communications Networks**]: Distributed Systems—*Distributed Applications*

Keywords

software-defined networking; testbed; topologies; datacenter

DIDC 2014, June 23, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2608020.2608023.

1. INTRODUCTION

Software-defined networking (SDN) technologies such as OpenFlow [22] have emerged in the last few years as a promising new approach to operating computer networks. The seminal feature of SDN is centralized control of packet forwarding via software controllers. The implication is phenomenal. Given centralized knowledge of the complete network topology and traffic demand, optimizing and fault proofing of traffic forwarding can be conveniently implemented and invoked in the SDN controllers. Unlike creation of a new protocol or protocol stack, introduction of SDN is a change of paradigm – as its centrally optimized and proactive control methodology presents a stark difference to today's distributed and reactive Internet architecture. Coupled with high speed networking technologies (100 Gbps in the Internet core and 10 Gbps to the edge), the centralized control methodology makes an even stronger case for achieving extremely high end-to-end networking performance via central optimization. The future is promising, but work is needed to inspect nearly all aspects of networking that we have grown accustomed to today.

In the context of data-intensive computing, the most direct benefit brought by SDN is the ability to flexibly and reliably control network bandwidth amongst compute hosts and storage [18]. The collection of the last two years' research presented at top conferences for cluster, grid, HPC and cloud computing have arrived at the following common conclusions: First, virtual machines running in modern data centers are increasingly the desired vehicle for data intensive computing due to their clean OS isolation, fast configuration, pause, migration, termination, and low setup overhead compared to direct execution over bare metal servers [12]. Secondly, benchmark tests have repeatedly confirmed performance bottlenecks due to low disk I/O or network throughput instead of processing capacity [18]. Finally, enabling virtual machines with dedicated resources, including processor cores, storage, and network bandwidth significantly enhances performance [8].

The research trend clearly suggests that effective resource allocation in a data-intensive distributed environment is a top priority for the community. While allocating processors

for data-intensive computing tasks using batched task schedulers is a well-learned practice, the same is not as well understood for storage and network resources. To date, the majority of data centers provide best-effort network throughput among processors and aggressively over-provision the raw network bandwidth to achieve acceptable performance for users. Where high network throughput is absolutely needed, it is offered as a premium service by placing the tasks on separate carefully admission-controlled hardware with high-speed connectivity. This practice results in a substantial loss of potential computing capacity and, even more crucially, severely limits HPC data centers' ability to host time-sensitive applications. In the case of scientific workflows in such disciplines as genomics, synthetic chemistry, atomic physics and beyond, very large data transfer is the cause for severe bottlenecks. Today, this is addressed by custom planning and over-provisioning of networks among dedicated HPC centers for the exact data flows expected. The approach will not scale beyond a few heavily invested sites, such as those established for the LHC project [10]. To overcome such limitations, methodologies for joint allocation of compute, network, and storage resources must be studied.

Much potential exists in the integration of SDN technologies and performance optimization of distributed and parallel applications within a datacenter. Building entirely new experimental data centers or modifying existing production data centers to study this investigation is costly in time, dollars, and operational output. Historically, researchers have used testbeds or controlled infrastructure to experiment with information technology systems resembling real systems and networks. This approach is advantageous to the SDN and datacenter researcher and can provide significant gains if establish with realistic conditions and instrumented to give constant, measurable results. Our results are the first step to provide a capability that will lead to a roadmap for developing methodologies to study this integration.

In this paper, we introduce the Flow Optimized Route Configuration Engine (FORCE), which is a testbed with an emulated multi-rack datacenter with programmable inter-rack topology. The FORCE provides a cost effective research environment that allows initial exploration into SDN enabled optimizations of parallel and distributed applications. The FORCE allows researchers to get a much earlier understanding the potential impact of network reconfigurations on application run-time at a fraction of the financial and time costs associated with deploying major network and computing reorganization into a production datacenter.

The contributions of this work include the following:

- The design and prototype development of a datacenter testbed with a reprogrammable network topology. The testbed includes a Virtual Topology Engine that builds virtual network topologies over physical links with SDN flows and a Flow Network Evaluation system to generate a network congestion estimation score.
- The design and development of a Hadoop shuffle traffic simulator designed to place realistic load on a datacenter network.
- Experimental results to show that placement of computation racks within a datacenter topology can have significant impact of the Hadoop shuffle traffic completion time.

The remainder of this paper is organized as follows. In section 2, we provide motivations for our research along with background information on software-defined networking. In section 3, we describe our architecture of the FORCE with its hardware, software, and custom integrations. In section 4, we discuss our use case of Hadoop and the shuffle traffic simulator. In section 5 we outline our experiments using the FORCE tool along with analyzing the results. We discuss related works in section 6. We summarize our paper by presenting our conclusion and anticipated future works enabled by the FORCE in section 7.

2. MOTIVATION

In this section, we describe our motivations and goals for the research infrastructure testbed, which includes the enabling capabilities of software-defined networking. Our overall research goal is to investigate technologies and methods for optimizing the performance and energy efficiency of parallel and distributed applications in a cluster or cloud environment. Our goals include the study of how the performance of distributed applications is impacted by the network topology of the datacenter [29]. Analytic and simulation performance models often do not capture all of the characteristics of real applications in a complex distributed environment. At the same time, experimenting with real systems and manipulation of real datacenter topologies can be very expensive in time and dollars. It is not practical to reconfigure an academic production datacenter topology for experimental studies, for example. However, SDN is a technology that can be used to easily and temporarily reconfigure physical and virtual network topologies. Our testbed is a low-cost experimental platform that uses a networked set of single computers and virtualization to emulate the performance of whole racks of machines in a datacenter and their applications. The testbed provides an SDN infrastructure that can be used to study how changes in network topology can impact the performance of the applications.

Several research goals lead us to the integration of SDN and parallel and distributed applications. First, we desire to build a low-cost reconfigurable testbed. We use a modest number of workstations and custom software to emulate the behavior of the entire datacenter. Secondly, we strive to use the testbed to study the effects of network topologies on distributed applications. Our testbed is configured with SDN hardware. Our research goals include using the testbed to gain insight and an early indication of which hypotheses of the use of SDN are worthy of further investigation. Lastly, we would like to understand how execution time modifications of the topology of the datacenter can be used to optimize application performance.

The relationship between network throughput and performance in the Hadoop distributed computing platform has been studied in the literature. Network congestion can delay the movement of data between compute nodes in the shuffle phase of a MapReduce job as reduce tasks fetch data from completed map tasks [29]. These delays can cause overall performance degradation as well as straggler tasks that lengthen the total run time and overuse the available task slots [11]. Network performance also affects operations in the cluster's underlying distributed file system, such as data replication in HDFS [28] and bulk transfers in OrangeFS/PVFS [26].

Congestion in the network can arise for several reasons, such as inadequate provisioning of overall network resources or imbalance in the application workload. Congestion can also occur because of suboptimal decisions or lack of adaptivity in the allocation of the available network resources. In typical cluster configurations, multiple paths exist between any two nodes. Although network switches incorporate forwarding algorithms that attempt to balance traffic across multiple paths, legacy load-balancing algorithms on switching hardware spread traffic based on hashing the source and destination address and protocol. [4] This limits coexistent flows' ability to leverage available capacity in the data center network, causing congestion even when there is available bandwidth in an alternate path. For short flow applications such as MPI the performance can be more sensitive to latency than throughput and the existing mechanisms for network configuration and adaptation may often be adequate. However, in data-intensive applications such as MapReduce and its supporting file systems the workload tends to pass data in very long flows. These types of applications are more tolerant of latency and the predominant limiting factor in performance is throughput. In these cases the performance can be improved by execution time network allocation decisions that are enabled through heightened visibility and maneuverability of the network.

SDN allows a operator-managed controller such as Floodlight [2] with a high level view of the network to remotely program network switches through a protocol such as OpenFlow [22]. The controller is capable of programming its associated switches either reactively or proactively. In the reactive case, a switch receiving a packet for which it has no forwarding rules can query the controller to install rules matching aspects of the packet's header. The controller can also respond to other events, such as a signal of switch failure, to proactively install forwarding rules. Afterward, matching packets are forwarded according to the installed rule.

3. ARCHITECTURE DESCRIPTION

In this section we describe the architecture design of the FORCE testbed. We discuss the hardware, the off-the-shelf software, and custom software designed by the research team. We introduce our virtual topology engine and our flow network evaluation system. A novel design aspect of the FORCE is the use of single workstations to emulate entire datacenter racks full of computing nodes. This emulation allows us to study the inter-rack networking traffic for distributed applications and to understand how topologies impact performance.

3.1 Hardware

The hardware of the FORCE includes one primary server, twelve client workstations, and two SDN-enabled switches. We note that the testbed is extensible and scalable to a very large size. This set of computers used in the testbed is repurposed from upgraded student laboratories and completed research projects and is very low cost. The equipment is installed in a location that allows students to have physical access to the equipment throughout the system building and experimentation.

The primary server is a Dell Precision 330 workstation powered with an 2.40 GHz Intel Core2 Quad processor with 4 GB of RAM. An additional dual-port gigabit Ethernet card was added to the server to provide additional network connections. The twelve client workstations are Sun Ultra20-M2 workstations, each with a dual-core AMD Opteron processor with 2 GB of RAM. Two additional dual-port gigabit Ethernet cards were added to these workstations to provide four additional network connections, bringing the total Ethernet ports to six each.

The two network switches are Pica8 Pronto 3290 48-port gigabit Ethernet switches. Each switch is OpenFlow enabled. Two VLANs are established on one SDN-enabled switch for the `control` and `access` networks. The server and the workstations each have one connection to both of these VLANs. The remaining 48-port switch is connected to each of the additional four gigabit Ethernet switches of the twelve workstations. This switch allows each workstation to be connected to a maximum of four other workstations. This switch is the primary target of the reprogrammability of the FORCE testbed.

3.2 Software

The FORCE package contains a number of important standard and custom software suites and tools for managing the testbed and quantifying experimental results.

3.2.1 Open Source Software

The FORCE utilizes several open source software packages without modification, including Ubuntu Linux 12.04.2, Open vSwitch 1.11.0, Floodlight v0.90, and Python 2.7. The server and workstations in the cluster all use Ubuntu Server 12.04.2 as the base operating system.

Since the release of Linux kernel version 3.3, Open vSwitch has been the default bridge system. Open vSwitch is used to provide a virtual top-of-rack switch on each of the 12 workstations. This allows each workstation to emulate one datacenter rack with a configurable number of computational nodes. There is a many-to-many communication network between these virtual switches through the 48-port Pica8 physical switch as described above.

The server node serves as the central management node of the SDN-enabled testbed, using Floodlight v0.90 as the controller software. The server is configured to control the two physical switches along with the 12 virtual switches. When a new topology is deployed, this node issues the OpenFlow commands to install new flows on the switches.

Python 2.7 is core to the operation of the FORCE. Python scripts are used to select topologies, issue OpenFlow commands, start experiments, collect data, and build graphs. The specific software packages are discussed in the remainder of this section and also in section 4.

3.2.2 Virtual Topology Engine

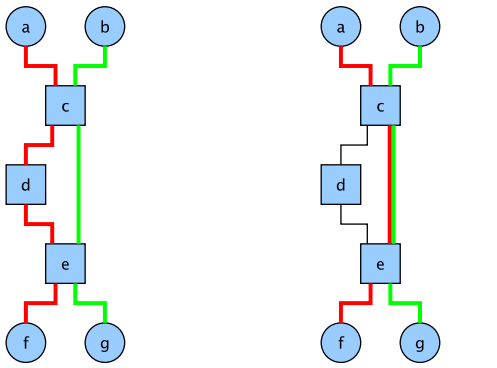
The core of the virtual topology package is our custom developed `force` tool, which implements a virtual network topology by installing forwarding rules on the SDN switches in a cluster. The topology is described in a series of layers using the NetworkX Python package [1]. Each layer maintains a network graph, as well a method for finding a path between any two vertices. The lowest layer describes the physical topology of hosts, switches, interfaces, and links, including characteristics such as hardware addresses and link speeds. The tool then applies subsequent graph layers that abstract each previous layer, building the virtual topology by translating edges into paths on the underlying graph. As a whole, this layered abstraction approach allows us to

map desired connectivity among vertices on the highest level graph to the lowest level flow rules to be installed onto the SDN switches.

As an example, a virtual 2D-torus topology can be implemented in three layers, as shown in Figure 1. In the first layer, the physical network is described. In the case of our experimental cluster, we define a single 48-port switch, 12 virtual switches with 4 links each to the central switch, and 12 hosts with links to the virtual switches. Then, we describe the desired topology, where each virtual switch is connected to its neighbor in a 2D-torus. Each edge in the virtual topology represents a path in the underlying topology, so the edge $vs1, vs2$ in Figure 1(b) translates to the path $(vs1, pronto, vs2)$ in Figure 1(c). The topmost layer describes the hosts that are logically connected, and each edge describes the full path between those hosts: the edge $h1, h6$ translates to the path $(h1, vs1, pronto, vs2, pronto, vs6, h6)$. Rules are then installed on the SDN switches: for this path, rules will be installed to forward traffic on $pronto$, $vs1$, $vs2$, and $vs6$.

3.2.3 Congestion Metric

As our software was written to experimentally evaluate network topologies, we wrote a tool to predict the congestion level of the network given a set of source/sink pairs. This tool approximates the mean of all flows' congested flow potentials divided by their flow potentials when free of congestion, if all concurrent flows in the network were in a state of equilibrium. We refer to this metric as the Link Sharing Score (LSS), where $0 < LSS \leq 1$. An LSS of 1 means that there are no overutilized edges in the flow network, and a score of 0.5 would describe a network where congestion cuts flow rates to 50%, on average.



(a) Source/sink pairs (a,f) and (b,g) with paths (ac, cd, de, ef) and (bc, ce, eg).

(b) Source/sink pairs (a,f) and (b,g) with paths (ac, ce, ef) and (bc, ce, eg).

Figure 2: In **(a)**, the two paths do not share a link, and the LSS algorithm assigns a score of 1. In **(b)**, edge ce is shared, cutting both flows to half capacity and giving a mean congestion score of 0.5.

In Algorithm 1, *topo* is an object containing a graph representation of the computer network and a method that computes a static path between a pair of nodes. *pairs* is a set of source-sink pairs, representing all concurrent flows on the network. In our experiments, this was a list of all mapper-reducer relationships for a set of simulated Hadoop jobs.

Algorithm 1 Link Sharing Score

```
1:  procedure LSS(pairs, topo)
2:      for all (src, dst) in pairs do
3:          path ← topo.path(src, dst)
4:          path_rate ← min_link(path)
5:          for all edge in path do
6:              edge.usage ← edge.usage + path_rate
7:          end for
8:      end for
9:      for all (src, dst) in pairs do
10:         path ← topo.path(src, dst)
11:         path_rate ← min_link(path)
12:         rate ← path_rate
13:         for all edge in path do
14:             scaled_rate ← path_rate × max(1, edge.cap / edge.usage)
15:             rate ← min(rate, scaled_rate)
16:         end for
17:         total ← total + rate / path_rate
18:     end for
19:     return total / len(pairs)
20: end procedure
```

The first loop of the algorithm assigns a usage amount to each edge in *topo* by summing the potential rate of all flows through that edge. Line 3 finds the path from *src* to *dst*, which could be the shortest path or any other walk that has been defined. Line 4 establishes the potential uncongested rate of a flow by finding the lowest capacity edge in the path. In lines 5 and 6, this rate is added to the usage of each edge in the path. This may or may not result in an overutilized edge.

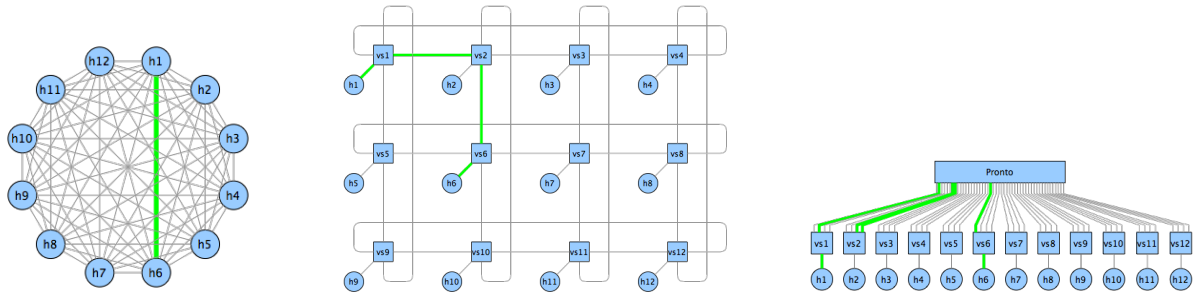
The second loop finds the congested rate of each flow by finding the minimum capacity of the congested edges in its path. Lines 10 and 11 again establish the path and uncongested rate along that path. Lines 12-16 find the lowest shared capacity edge in the path, with Line 14 calculating the shared capacity based on the fractional usage of the edge and Line 15 receiving the congested rate. Lines 17 and 19 compute and return the mean quotient of the congested rates to the uncongested rates.

4. USE CASE: HADOOP

We select Hadoop MapReduce for the representative use case for demonstration of the utility of the FORCE testbed. The literature [29] describes the potential speedup of MapReduce shuffle traffic that is possible by positioning datacenter racks that contain the reduce tasks in close network proximity to datacenter racks that contain the map task from the same MapReduce job. This proposed enhancement requirs the dynamic reallocation of point-to-point connections between top of rack switches in a two-dimensional torus topology. Our testbed is ideally suited for testing this optimization. In this section we describe the use of the FORCE to experimentally test how MapReduce performance is impacted by reallocation of point-to-point connections and to determine if further research and investigation of the method is justified.

4.1 Hadoop Background

Hadoop MapReduce is the most popular MapReduce framwork. It is an open-source project that implements a parallel



(a) The topmost layer, defining all possible source/destination pairs.

(b) Each abstraction layer defines the paths through the previous layer, as well as a routing algorithm between nodes on this layer.

(c) The physical topology layer, consisting of hosts with virtual switches, each connected through 4 interfaces to a central SDN-enabled switch.

Figure 1: A virtual topology is described as a sequence of abstraction layers, with each edge representing a path on the layer below it. In this series of diagrams, $h1, h6$ in **(a)** translates to the path $(h1, vs1, vs2, vs6, h6)$ in **(b)**, which then becomes $(h1, vs1, pronto, vs2, pronto, vs6, h6)$ in **(c)**. Paths on the lowest level become sets of rules to be installed on switches *pronto*, $vs1$, $vs2$, and $vs6$.

programming environment for computation over very large data sets using a cluster of commodity servers and workstations. We focus on the behavior and architecture of Hadoop V1.0 in our testing environment.

Hadoop has a distributed network file system implemented in a client-server model using block level replication across the cluster. The *NameNode* is the metadata server for the Hadoop Distributed File System (HDFS). *DataNodes* store the actual data blocks with a default size of 64 MB. The mechanism of HDFS are not relevant to our work, but future implementations could focus on the data read, write, and replication traffic.

Like the distributed file system, the MapReduce computation model also implements a single central management node, *JobTracker* and multiple computation nodes called *TaskTrackers*. DataNodes can reside on the same machine as TaskTrackers, thus allowing the architecture to take care of the adage "moving the data to the computation." The JobTracker is responsible for delegating the specific map and reduce tasks for a submitted MapReduce job to a subset of the TaskTrackers in the cluster. The JobTracker receives heartbeat updates of the status of the TaskTrackers providing a timely completion of the job. A single TaskTracker can be assigned both map and reduce tasks within the same job. TaskTrackers for Hadoop clusters can be spread across multiple computing racks in a datacenter.

A MapReduce job is made up of multiple map tasks and multiple reduce tasks. A mapper takes an input set of data in a key-value pair (K, V) and outputs an intermediate key-value pair (K', V'). The input to the reducer is an intermediate key and the set of intermediate values for that key $(K', \{V'_1, V'_2, V'_3, ...\})$ from all the mappers across the cluster. The reducer performs some computation on the set of intermediate values and produces the final output value for each key it was responsible for reducing. The entire possible set of intermediate key values are partitioned into a single partion per reducer. During the shuffle phase between map and reduce, a reducer pulls its respective partition from each mapper before beginning the reduce computation. This bulk data transfer has been shown to be a network bottleneck in MapReduce clusters.

The JobTracker for a MapReduce cluster, in concert with the NameNode, is data locality aware for assigning map tasks; however, it does not consider data locality for assigning reduce tasks by default. Data locality awareness means that the JobTracker assigns map tasks to the TaskTracker which is the home to the input data in HDFS. This allows the computation to be executed on the data with no network transfer. As such, TaskTrackers can begin executing as soon as they are assigned a map task. In our test environment reducers receive data from all the mappers and wait for all maps to complete before calculating their output. TaskTrackers are assigned reduce tasks and specific partitions on a first come first serve basis with no consideration to map output data location.

The division of possible intermediate key values is customizable by the user on a job-by-job basis in a MapReduce cluster. This functionality is provided by a partitioner. The default partitioner for MapReduce is the HashPartitioner, which applies a hash function to the key value and then computes the remainder when the hash value is divided by the number of reducers. This modulo math function provides the index of the R reducers that are responsible for reducing the set of values associated with this key. The partitioner expects a uniform distribution of keys across the set of reducers, in theory assigning each reducer $\frac{1}{R}$ of the intermediate keys. This partitioner does not consider the sizes of the intermediate values that are associated with each intermediate key. The difference between the actual measure of the partition size and the theoretical size for each reducer is called the reducer partition skew. This skew is a function of the nature of the input values, the intermediate key set, and associated size of the intermediate values, and is a factor in the amount of network traffic between the mappers and reducers.

4.2 Hadoop Shuffle Simulator

The nodes comprising the testbed are not robust enough to run a real workload consisting of multiple Hadoop jobs or multiple virtual Hadoop nodes. To solve this problem, and to ensure that the testbed supports a realistic shuffle traffic network load that corresponds to the traffic between datacenter racks, we implemented a Hadoop shuffle traffic emu-

lator within the FORCE. We implemented a centrally controlled software suite that synchronizes bulk network data transfers from map tasks to reduce tasks within the cluster. These data transfers are the same as the movement of map tasks outputs to reducers across the cluster as seen in the shuffle phase of a real MapReduce job.

The Hadoop shuffle emulator reads configurations from a set of files that describe the pseudo Hadoop cluster within the FORCE. The cluster configuration file defines the number of emulated nodes within in each virtual rack, the number of TaskTrackers within each emulated node, the number of map and reduce slots present per TaskTracker, and the default HDFS block size. The workload configuration file defines the number of jobs to be placed on the cluster during an experiment, the input size of the data to be processed by the job, and the type of MapReduce job for each job. With the configuration variables retrieved from these files, various workloads and workload execution scenarios can be emulated within the FORCE without the need for robust servers running multiple Hadoop virtual machines.

Given the set of configurations and experiments, the system is able to deploy the emulated MapReduce jobs across the cluster that perform transfer of shuffle traffic. Based on the size of the data to be processed and the HDFS block size, the system is able to determine the number of maps task required for each job. The number of reduce tasks is determined by a default global parameter or specific argument on a per job basis. The type of job to be run determines the ratio of map task output relative to the block size. After the system is configured with the number of map and reduce tasks per job and the size of the data transfer between all the map tasks and the set of reduce tasks in each job, the transfer of data from map tasks to reduce tasks begins. Since we are only interested in studying the inter-rack traffic, any map and reduce tasks that reside within the same virtual datacenter rack do not transfer data.

5. EXPERIMENT DESIGN

In this section we describe the design of the experiments using the FORCE and our Hadoop shuffle simulator that study the impact of placement of reduce and map racks within a datacenter. We describe the cluster configuration, the workload placed on the environment, the data collected, and the method of collection.

We initialized our cluster with the full set of 12 racks and 16 compute nodes per rack, with 8 map slots and 4 reduce slots per node. This configuration emulates a cluster with 1536 map slots and 768 reduce slots. Using the method of bin-packing placement [29], we ran three simulated MapReduce jobs with reducers exclusively on three separate computing racks. Each reducer rack had its own set of three feeding map racks. Thus we had three distinct sets of mapper and reducer racks divided across our datacenter.

In our first experiment, each experimental run consisted of using the FORCE to build and record a random 4x3 two dimensional torus topology, beginning continuous shuffle traffic from each map rack to its destination reduce rack, and recording the total bytes transferred from each virtual top-of-rack switch at each map rack. We ran 1000 experimental runs and recorded the amount of data transferred at each map rack at equally spaced times. These measurements allowed us to compute the average transfer rate at each map.

Table 1: Mean Throughputs on Various Node Placements in a 2D-Torus

Node Placement	Trials	$\bar{x}$	s
same	500	620.7 Mb/s	2.27 Mb/s
random	1000	563.6 Mb/s	79.78 Mb/s

In our second experiment, each run consisted of again randomly placing the rack in different positions in the 4x3 torus topology. We then began the flow of data between mappers and reducers in deterministic sizes. We recorded the time required to complete each of these data flows. This is different from the first experiment since less congested paths allow the transfer to finish sooner, permitting straggler flows to continue on a network free of congestion.

In both experiments, we established a baseline for comparison with randomized topologies by measuring the results with 500 runs using a fixed network topology with no particular distinguishing characteristics.

5.1 Performance Evaluation

In this section we report evaluation results are on the network performance and for the Hadoop MapReduce application.

5.1.1 Network Performance Results

After sampling the mean transfer rate of 1000 random topologies, we found that the mean and variance were significantly different than obtained with 500 runs using the baseline topology, as shown in Table 1. We also found that the mean throughput for the baseline topology is 620.7Mb/s while the mean throughput for the randomly set of topologies is 563.6Mb/s. The variance of measured throughputs of the baseline is small, only 2.27Mb/s, whereas the variance of measured throughputs of the set of 1000 randomly generated topologies is much larger, at 79.78Mb/s. This indicates that there are topologies with better and/or worse congestion characteristics. We also found that the baseline topology had a mean transfer rate that was higher than that of random topologies. These results provide the motivation for future investigation into optimal topologies for the MapReduce workloads.

Next, we used the described LSS congestion scoring algorithm to calculate the theoretical mean congestion of each topology. These scores correlate reasonably well ($r^2 = 0.677$) with the mean throughput of each topology, as shown in Figure 3. This tendency can also be seen in Figure 3, which supports our conjecture that certain topologies have less congestion than others, given a certain set of flow patterns.

5.1.2 Hadoop Performance Results

The results from the second experiment of simulated Hadoop jobs showed similar results. A histogram of simulated job run times is shown in Figure 5. As shown in the chart, some run times are over 2.5 times higher than base case times. These worst case configurations ideally would be the topologies an intelligent system would strive to avoid. Additionally, Figure 4 shows that the LSS congestion score correlates with the completion time of jobs, which is to be expected as the job completion time affected by the time spent transferring intermediate data. This correlation shows a congestion score is potentially an early indication of shuffle time per-

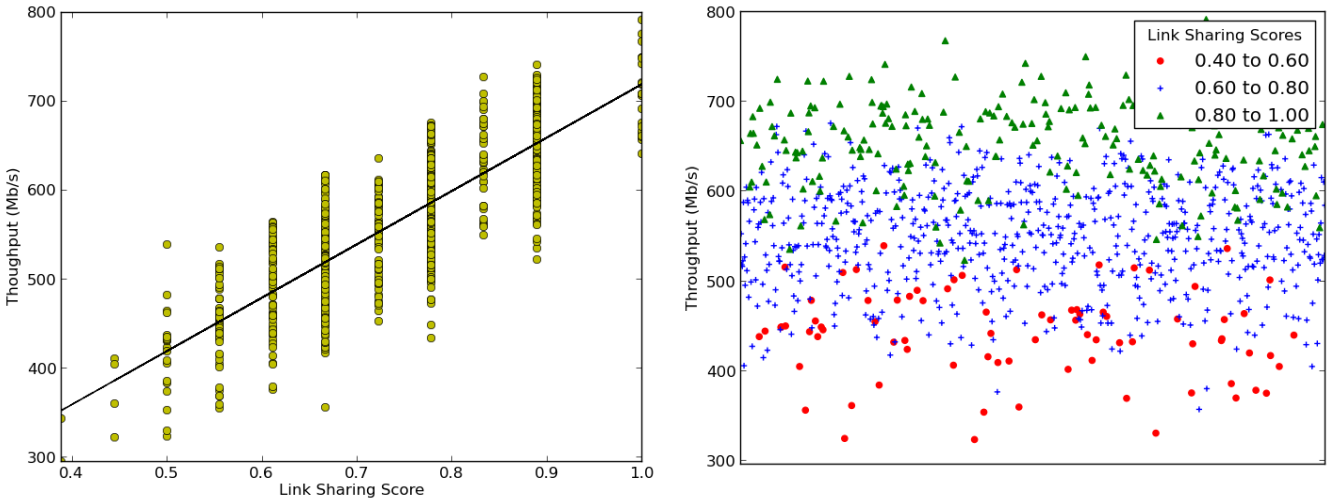


Figure 3: The mean throughput of a set of 9 flows is shown, with each point representing a different random placement of nodes within a 2D-torus. The congestion score, with 1.0 representing no shared links, correlates well with the observed throughput.

formance and can be used to intelligently select a topology given a specific network flow pattern.

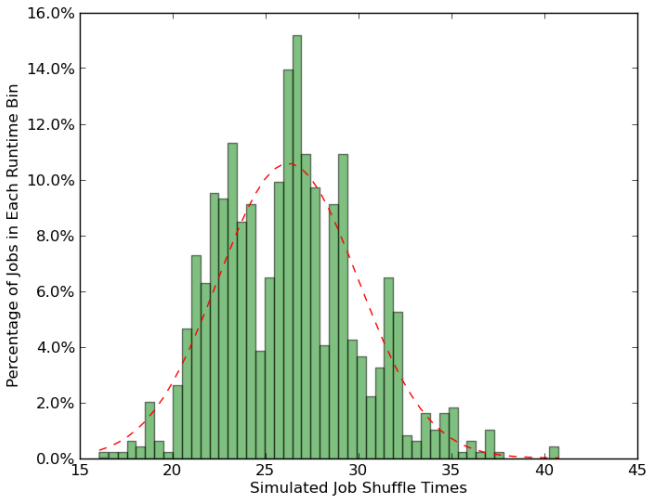


Figure 5: Histogram of simulated Hadoop shuffle times with random placement of racks in two-dimensional torus topology.

5.2 Implementation Evaluation

The combination of the FORCE testbed along with the Hadoop shuffle traffic simulator has allowed us to obtain encouraging experimental results to support the hypothesis about topological impact on MapReduce shuffle traffic. Without the use of the testbed and traffic simulator, a much more significant investment of research dollars and time would have been needed to obtain experimental results to support our hypothesis.

The experimental results suggest that further investigation is warranted. We are encouraged by the results and can now begin further study of flow optimization under different topologies. Our hypothesis that topology can impact the completion time of MapReduce shuffle traffic is reinforced through experimental results.

The use of the FORCE as an emulated testbed for studying the effect of network topology on application performance in a datacenter has proven beneficial. Continued use of this research methodology could prove financially advantageous to academic and research organizations.

6. RELATED WORKS

There is a breadth of research in many of the components that compose the FORCE system. Researchers have focused on software-defined networking, Hadoop shuffle traffic, and data center network topologies. We highlight some work in each of these categories in this section.

Pioneering SDN research efforts have explored use cases for network virtualization [27], service insertion [3, 20, 25], load provisioning [17, 30], network debugging [16], and programming methods. Our work seeks to harness the promise and potential of those works in the area of datacenter topology optimizations.

Many solutions have been proposed for minimizing the time and size of Hadoop shuffle traffic. A theoretical model that inspired our initial testbed hypothesis suggests runtime reconfiguring of topologies can optimize big-data processing [29]. This work though lacks implementation and experimental results. Our work is the first step in expanding on some of their ideas. Other research has focused on the reduce task scheduling [14, 15] and the partitioning function [19]. As these works look to schedule tasks and partitions to improve shuffle traffic performance, we look to schedule the network to optimize application performance. Many works have looked at OpenFlow enhancements for Hadoop traffic in general [21, 24, 31]. These works use software defined network to create prioritized flows and logical paths. Our work is focused on reconfiguring physical network topology with on-demand point to point links for top of the rack switches.

There is increasing interest in the topology of data centers with researchers focusing on decreasing cost, improving application performance, and lowering power consumption.

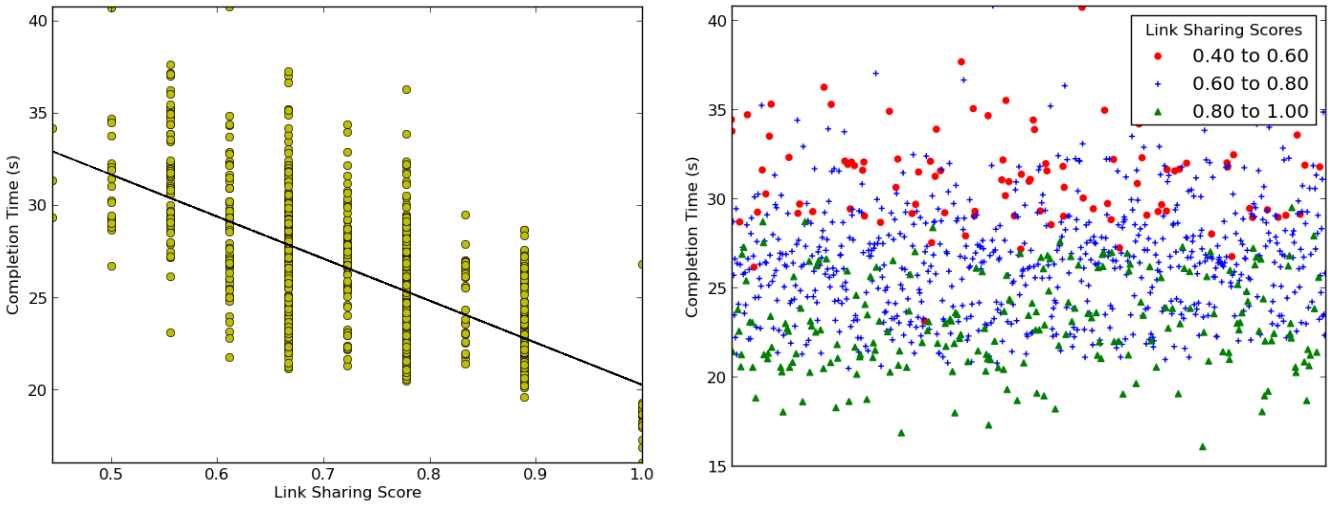


Figure 4: Simulated Hadoop job completion time also correlates with network congestion. The difference between optimal and suboptimal configurations can have significant effect on the overall time taken.

New data center topologies have been proposed [5, 6, 13, 23] while others have proposed SDN enhancements to the flows within current topologies [32]. Finally, research is even further growing in understanding network characteristics and optimization in production data centers [7, 9]. These advantages and system gains of these works could be deployed within a reprogrammable topology when certain conditions exist. Our work leads to a future mechanism to provides this features when at the optimal time.

7. CONCLUSION

We have presented the Flow Optimized Route Configuration Engine (FORCE), a datacenter testbed emulator with a programmable interconnection controlled by an SDN controller. The FORCE allows researchers to get an early indication of the worthiness of data center topology hypothesis. These experimental results come without the cost in time or funding of building production level data centers. Additionally, the system features a Virtual Topology Engine, a Flow Network Evaluation System, and a Hadoop shuffle traffic simulator. We have presented initial experimental results to suggest that datacenter topology, specifically placement within a 4x3 2-D torus network, can impact the time to shuffle intermediate results from a MapReduce job.

In the future, we plan to build a complete Hadoop traffic simulator, upgrade the emulated rack workstations, and develop a system that will provide execution time adaptivity and maneuverability of datacenter topology to steer away from worst case scenarios. We also plan to deploy and validate our hypotheses in production data centers with SDN capabilities.

8. ACKNOWLEDGMENTS

We would like to acknowledge Prof. Brian Dean and Prof. Ilya Safro for advising on existing work on measure network and graph congestion. We also acknowledge the assistance of Dylan Wagenseller in acquiring and installing the hardware needed to build our testbed. This work was supported in part by the National Science Foundation Grants #1212680 and #1228312.

9. REFERENCES

[1] NetworkX - High-productivity software for complex networks. http://networkx.github.io/.

[2] Project Floodlight. http://www.projectfloodlight.org/.

[3] The OpenStack Project. http://www.openstack.org.

[4] Understanding ECMP Flow-Based Forwarding. http://www.juniper.net/techpubs/en_US/junos13.3/topics/concept/routing-policy-security-ecmp-flow-based-forwarding-understanding.html.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74, 2008.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[7] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[8] T. Benson, A. Akella, S. Sahu, and A. Shaikh. Epic: Platform-as-a- service model for cloud networking. Technical Report 1686, CS Department, University of Wisconsin, Madison, 2011.

[9] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, Jan. 2010.

[10] CERN. The grid: A system of tiers. http://home.web.cern.ch/about/computing/grid-system-tiers.

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

[12] R. J. O. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A case for grid computing on virtual machine. In *Proceedings of the International Conference on Distributed Computing SYstems*, 2003.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[14] M. Hammoud, M. Rehman, and M. Sakr. Center-of-gravity reduce task scheduling to lower MapReduce network traffic. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 49–58, 2012.

[15] M. Hammoud and M. Sakr. Locality-aware reduce task scheduling for MapReduce. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 570–576, 2011.

[16] B. Heller, C. Scott, N. McKeown, S. Shenker, A. Wundsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, K. Zarifis, and P. Kazemian. Leveraging sdn layering to systematically troubleshoot networks. In *Proceedings of ACM Workshop on Hot Topics in Software Defined Network*, 2013.

[17] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[18] M. Hillenbrand. Towards virtual infiniband clusters with network and performance isolation, 2011.

[19] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi. LEEN: Locality/Fairness-Aware key partitioning for MapReduce in the cloud. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 17–24, 2010.

[20] J. W. Lee, R. Francescangeli, J. Janak, S. Srinivasan, S. A. Baset, H. Schulzrinne, Z. Despotovic, and W. Kellerer. Netserv: Active networking 2.0, 2011.

[21] Z. Li, Y. Shen, B. Yao, and M. Guo. OFScheduler: a dynamic network optimizer for MapReduce in heterogeneous cluster. *International Journal of Parallel Programming*, pages 1–17.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38, 2008.

[23] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies. In *NSDI*, pages 265–280, 2010.

[24] S. Narayan, S. Bailey, and A. Daga. Hadoop acceleration in an OpenFlow-Based cluster. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 535–538, 2012.

[25] A. Rosen and K.-C. Wang. Steroid openflow service: Seamless network service delivery in software defined networks. In *Proceedings of the First GENI Research and Educational Experiment Workshop*, 2012.

[26] R. B. Ross and R. Thakur. PVFS: a parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 391–430, 2000.

[27] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flowvisor: A network virtualization layer. OpenFlow Switch Consortium Technical Report, 2009.

[28] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. K. Panda. Can high-performance interconnects benefit hadoop distributed file system. In *Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO*, 2010.

[29] G. Wang, T. E. Ng, and A. Shaikh. Programming your network at run-time for big data applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 103–108, New York, NY, USA, 2012. ACM.

[30] R. Wang, D. Butnariu, and J. Rexford. Openflow-based server load balancing gone wild. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[31] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal. Hadoop acceleration through network levitated merge. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 57, 2011.

[32] K. C. Webb, A. C. Snoeren, and K. Yocum. Topology switching for data center networks. In *Hot-ICE Workshop*, 2011.