Application-Aware Network Design for Hadoop MapReduce Optimization Using Software-Defined Networking

Shuai Zhao, *IEEE Member* and Deep Medhi, *Senior Member*

***Abstract*—The MapReduce (M/R) framework used in Hadoop has become the de facto standard for Big Data analytics. However, the lack of network-awareness of the default MapReduce resource manager in a traditional IP network can cause unbalanced job scheduling and network bottlenecks; such factors can eventually lead to an increase in the Hadoop MapReduce job completion time. In this work, we propose a Software-Defined Network (SDN) approach in an Application-Aware Network (AAN) platform that provides both underlying networks functions as well MapReduce particular forwarding logics. We measure the resources' usage for MapReduce workloads using the HiBench benchmark suite to identify the traffic pattern. We then demonstrate that by using our AAN-SDN framework, which uses an adaptive traffic engineering mechanism, the job completion time can be noticeably improved.**

***Index Terms*—Application-Aware Network, Data Center Network, MapReduce, Software-Defined Networks, OpenFlow, Big Data.**

I. Introduction

The rapid development of cloud services, mobility, Internet of Things (IoT) sensors, and video streaming services not only led to an explosion of network data but is also challenging the existing network management and monitoring system. Users pay the premium and also expect quality on-demand access to those applications, infrastructure, and other IT resources. Handling today's mega datasets requires massive parallel processing that also puts a constant need on flexible capabilities from the underlying network. Network-based applications themselves are increasingly growing and require massively distributed computation.

To handle the ever-increasing data size, Hadoop [1] MapReduce (M/R) is a scalable framework that allows dedicated and a seemingly unbound number of servers to participate in the analytics process. The response time of an analytics request is a major factor for time to gain data insights. Hadoop has been designed as a shared computing and storage platform and supports parallel computing of jobs for multiple users. While the computing and disk I/O requirements can be scaled with the number of servers, scaling the system lead to increased network traffic in the underlying network. Arguably, the communication-heavy phase of M/R contributes significantly to the overall response time. This problem is further aggravated, if communication patterns are heavily skewed, which is common in many MR workloads. Most of this caveat of the MapReduce program is because the default Hadoop resource manager does not take the network condition into consideration for job scheduling.

Shuai Zhao and Deep Medhi are with the Department of Computer Science Electrical Engineering at the University of Missouri–Kansas City, Kansas City, MO, USA. Email: shuai.zhao@mail.umkc.edu, dmedhi@umkc.edu

The emergence of Application-aware Networks (AAN) provides a new approach for managing Hadoop network traffic.The AAN provides the capability of an intelligent network to maintain current information about applications that connect to it and as a result, optimize their functioning as well as that of other applications or systems that they control. The information maintained includes the application state and resource requirements. For example, a port number based application identification is common in a controlled network environment. The traditional IP network provides best-effort services for the applications. Each network device requires operator's particular configuration. It is difficult to write any application controller or other routing protocol in general. As more and more applications are moving toward cloud services, the best-effort service may not provide the quality that users expect to experience.

A software-based solution using the software-defined networking (SDN) is a fine-grained way of controlling individual application and network devices. AAN benefits from SDN in two ways: first, by enabling dynamic control, configuration and giving the ability of AAN to allocate resources at any given moment; second, by running network controls on a separate server from the traffic forward device.

In this work, we focus on Hadoop computation using an AAN framework through SDN when Hadoop nodes are spread out over a network. Such a situation occurs when due to the size of the Hadoop workload, the number of nodes needed cannot be satisfied from just one physical location. For our work, we first examine Hadoop traffic at length for the HiBench benchmark suite and how SDN can be used to optimize through AAN. We then present our AAN-SDN framework and show the gain when this framework is used for Hadoop applications. To our knowledge, such an approach has not been taken by other researchers on network softwarization to benefit Hadoop applications.

A. Motivation

A great deal of consideration must be put in place when managing a Hadoop cluster using resources for running

MapReduce jobs in order to fully understand Hadoop traffic. The key elements are summarized as follows:

1) *The block size and split size*: Hadoop uses blocks and split size to control how many blocks are being divided and used when running MapReduced jobs.
2) *Block replication factor*: Hadoop uses this approach to prevent data loss because of common hardware failures.
3) *The physical configuration of hardware resources*: This includes CPU, memory, hard disk capacity, interconnection network link speed and the number of slave nodes.
4) *Java Virtual Machine(JVM)*: Hadoop uses JVM to complete jobs. The number of resources, mainly CPU and memory, can be assigned to each created JVM. Because the process of creation and killing of such resources takes time, the rule of thumbs when using JVM is that the less mapper is better, which leads to less JVM creation and less killing time. However, this must be accomplished by being given sufficient resources to start with for the submitted jobs.
5) *Hadoop cluster topology*: How slaves deploy across the Hadoop cluster can be critical and depends on the assigned network bandwidth between master nodes and slave/data nodes.
6) *Other Hadoop performance tuning methods*: Additional factors such as the number of files, file size, JVM Reuse and combiners are also important for Hadoop performance tuning.

In a traditional IP network setting, existing Hadoop resource allocation algorithms [2]–[4] have performed well under the assumption that the network is not congested and a job is assumed to be completed under the presumed time range. However, when the network becomes congested during job runs, both the Hadoop master and system administrator have less control over the job run time if no traffic flow optimization is performed in time.

B. Our Contributions

In this paper, we present an AAN environment for Hadoop M/R optimization using the SDN architecture. We introduce tools and traffic reroute mechanisms for Hadoop application optimization. We first focus on understanding resource usage of difference MapReduce jobs and shuffle traffic patterns in order to identify possible network congestion problems that can lead to delay in job completion. Then, we ran real-world MapReduce applications using our proposed AAN environment that uses SDN architecture to show improvement on the overall job completion time.

Within an AAN environment using SDN, the controls can be given by a fine-granularity flow management in an SDN controller with adaptive traffic engineering to a contested network situation. Our contributions are as follows:

1) An application-aware framework with SDN
2) A new ARP (address resolution protocol) flooding avoidance resolver algorithm for our framework
3) A data flow model to improve the data movement efficiency for MapReduce related workloads
4) An adaptive traffic engineering mechanism for the AAN-SDN environment for Hadoop applications

The main benefits of our AAN approach are as follows:

1) It allows a controllable Hadoop cluster management system and a fine-granular application control platform using the SDN architecture.
2) It provides an open programming interface for more intelligent Hadoop resource allocations with consideration of global network traffic information.

For our study, we used the HiBench benchmark suite (with over 300 test cases) in two ways: 1) to obtain an understanding of Hadoop traffic, and 2) to show the gain with our Hadoop-AAN environment.

The rest of the paper is organized as follows. Section II presents a brief background on SDN, Hadoop MapReduce, and the HiBench benchmark suite. In Section III, we further explain the HiBench configuration and report initial measurements for the benchmark suite for a set of small static topology configurations in terms of use and traffic patterns. In Section IV, we present our proposed AAN-SdN framework. Section V explains our network topology on which we conducted an SDN adaptive traffic engineering approach to optimize MapReduce workloads in terms of the overall completion time using the proposed AAN platform and showing the gain with our approach. Section VI describes related work regarding individual MapReduce workload studies and optimization approaches. We then present our concluding remarks in Section VII.

II. Background

We first present a brief background on three important parts for our work: software-defined networking, MapReduce framework, and HiBench benchmark suite.

A. Software-defined Networking

Software-Defined Networking (SDN) [5], [6] provides a dynamic, manageable and cost-effective platform for making it an important platform for the high-bandwidth, dynamic nature of today's network applications. Fig. 1 shows the SDN architecture. It decouples the control and data forwarding layers and provides the programing interface for the underlying forwarding devices as well as upper application layer. The SouthBound and NorthBound APIs are provided as communication channels between the SDN layers.

AAN can be realized using an SDN architecture that includes two main components: an SDN controller and a traffic forwarding protocol using the forwarding devices. An SDN controller is a software application that manages application flows to enable a dynamic and controllable networking environment. The popular SouthBound communication protocols between SDN controllers and forwarding devices are OpenFlow [5]–[7], which allows servers to instruct forwarding devices where to send packets.

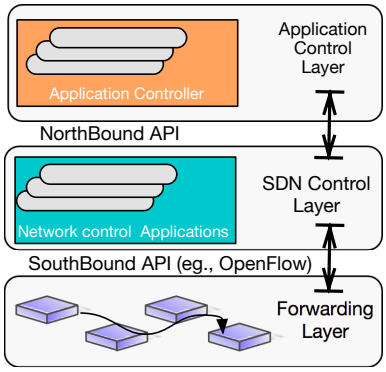


Fig. 1: Software-Defined Network Architecture

B. MapReduce Framework

A Hadoop cluster includes one master node to manage the cluster's metadata, such as the Hadoop File System (HDFS) and a resource manager such as YARN [8] that manages Job and task tracker for each submitted MapReduce job. Designated slave nodes run as computing powers. A Hadoop cluster is normally deployed in a closed and control environment such as in an Enterprise or a Campus datacenter (DC). Hadoop MapReduce [9] is a distributed and parallel computing framework that runs on top of the Hadoop File System (HDFS). A typical MapReduce program is composed of mixed operations among various numbers of mapper and reducer functions.

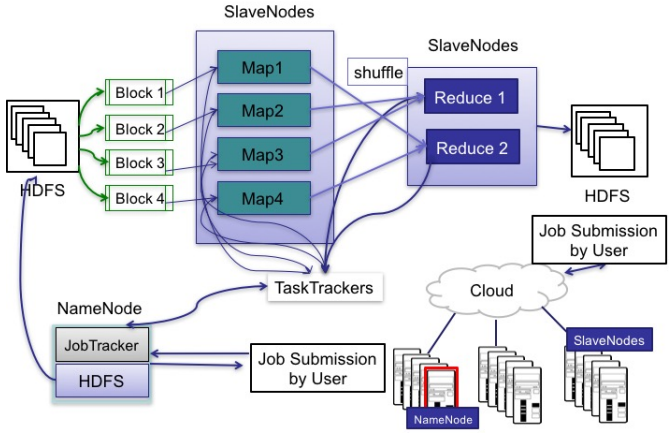


Fig. 2: MapReduce WorkFlow

Fig. 2 shows the major MapReduce workflows and data movements. After the job submission, the input data split into blocks of data. The number of mapper and reducer functions plays a vital role to decide how MapReduce jobs are running on a Hadoop cluster. Based on the design of the MapReduce platform, there is a critical data movement phase when a job is running (called shuffle) that represents the output of a mapper function that is transferred to reduce functions for the final processing. How fast the shuffle phase is completed can affect the overall job completion time. We summarize the possible situations where various traffic patterns might occur in a Hadoop cluster:

- HDFS management such as a cluster health check
- File reads and writes from HDFS such as data replication, MapReduce input and output, and Cluster balancing
- Data shuffle among data nodes
- Interaction between TaskTracker such as data shuffles from mapper to reducer functions and data write back to HDFS as the final output.

C. HiBench: Bigdata Micro Benchmark Suite

HiBench [10] is a big data benchmark suite that helps evaluate different big data frameworks in terms of speed, throughput, and system resource utilization. Hadoop MapReduce workloads in the HiBench benchmark suite include a number of applications such as Sort, WordCount, SQL, and PageRank.

Due to our goal of understanding the resources' usage and shuffle data traffic pattern and to later use this in our AAN-SDN environment, we use the workloads' categories depicted in Fig. 3 based on given hardware resources. The detailed HiBench configuration is explained later in Section III-A.

HiBench Hadoop Related Workloads Categories	
Type	**Explanation**
MicroBench	
Sort	Sorts its *text* input data using RandomTextWriter.
WordCount	Counts the occurrence of each word in the input data using RandomTextWriter.
SQL	
Scan	Performing the typical OLAP queries described in the [36]. Its input is also automatically generated Web data with hyperlinks following the Zipfian distribution.
Join	
Websearch Benchmarks	
PageRank	Benchmarks PageRank algorithm implemented in Spark-MLLib/Hadoop examples, using web data whose hyperlinks follow the Zipfian distribution.

Fig. 3: HiBench Hadoop MapReduce Related Workload Summary

III. HiBench MapReduce: Configuration and Initial Job Run Time Data Collection

Before considering our AAN-SDN framework, we conducted a set of measurements on the Hadoop HiBench benchmark suite while keeping the topology simple and static to understand traffic patterns. We first discuss HiBench configurations used for this work.

A. HiBench Configuration

The main considerations for running a MapReduce job include the job input size, the number of running slave nodes, the number of mappers and reducers, and the block size and location of the slaves. The Hadoop file block size was set to $32MB$ due to our limited hardware resources for the experimental platform and the replication factor is set to 2. Additional Hadoop related configurations are displayed in Table II. One of our goals in this work is to understand the shuffle traffic of Hadoop M/R. The selected configuration parameters are the key control points for our proposed AAN

TABLE I: HiBench MapReduce Workloads Configuration Details

Test Configurations	A	B-1	B-2	B-3	B-4	C-1	C-2	D-1	E-1	E-2	E-3
Data Size (MB)	16	48				240		304	304		
Slave Allocation	TA	TB-1		TB-2		TC		TD-4	TE		
(# of Mapper, # of Reducer)	(1, 1)	(1, 1)	(2, 2)	(1, 1)	(2, 2)	(4, 4)	(8, 8)	(8, 8)	(4, 4)	(8, 8)	(8, 8)
MicroBench Sort											
MicroBench WordCount											
SQL Join and Scan [11]											
Join and Scan Configurations	Universities=88000 pages=6	Universities=265000 pages=6				Universities=1325000 pages=6		Universities=1680000 pages=6	Universities=1680000 pages=6		
WebSearch PageRank											
PageRank Configuration	pages=38200 num_interactions=1 block=0 block_width=16	pages=110000 num_interactions=1 block=0 block_width=16				pages=480000 num_interactions block=0 block_width=16		pages=610000 num_interactions block=0 block_width=16	pages=610000 num_interactions block=0 block_width=16		

platform. For the rest, Hadoop default parameter values were used.

The small files problems [12] are avoided by limiting the total given file size. We setup a HiBench configuration based on a Hadoop MapReduce job type and the network topology locations for slave nodes as shown in Table. I. Consider test configuration "B-1" as an example to explain our setup of HiBench: its input file size is $48MB$. The number of mappers and reducers set to one each. At this configuration setup, we compare the run time difference with other "B-X" configurations as well as other test configurations using different HiBench settings. Slave locations are depicted in Fig. 4. For a TA type, only one slave is active for the submitted job. A TA case sets up a base comparison with other topology configuration types for the same MapReduce job with respect to job CPU, memory consumption and how long does it take to complete the job. We also ran a MapReduce job under different slave locations using a various number of slave nodes to understand if the location of the slave could affect the job completion time. For example in the TB-1 and TB-2 configuration setup, two slaves ran under the same topology location in TB-1 while they were in different locations for TB-2. A TE type uses eight slave nodes that are evenly distributed under four forwarding devices as shown in Fig. 9(a).

Some of the predefined MapReduce jobs in HiBench can be assigned a particular data size for input such as its micro workloads, WordCount and Sort, while others use different input parameters such as for Join and Scan [11] from the SQL workload. For example, a Join workload using two parameters to set up the workloads, "number of Universities" and "pages", instead of specifying the size of the input data. To ensure our input data size is the same, we conducted an initial experiment and derived the following relation based on empirical runs:

HiBench SQL workload: Join, Sort, Aggregation

$$Input\_Size(MB) = \frac{Number\ of\ Universities * 1.8}{10^5}. \quad (1)$$

HiBench WebSearch workload: PageRank

$$Input\_Size(MB) = \frac{Number\ of\ Universities * 2}{6\sqrt{2} * 5000}. \quad (2)$$

We determined that only the parameter, "number of Universities", can change the input data size for workloads such as Join, Scan, and PageRank. By varying the predefined parameters, we keep our test configuration consistent with the same amount of input data size for different workloads.

B. Data Collection cases

Hadoop MapReduce hardware resource data collection is conducted for a set of small static topology configurations. The goal is to understand the run time resource consumption such as CPU, memory, and job completion time for each HiBench workload that is depicted in Table. I. Altogether, there were 330 tests. Because of the size of the test cases, was ran each test three times for each configuration to compute the average value; we also calculated the 95% confidence interval (CI) to determine the oscillatory nature of the MapReduce jobs. Table IV lists the detailed values for reference. Based on our observations that different HiBench jobs behave differently, we first examined the resource usage separately from each other for each HiBench configuration. We will then present a combined analysis.

C. Average CPU Usage Summary

Fig. 5(a) depicts the average CPU usage for various configurations. From the runs, we made the following observations:

- The single slave node uses CPUs most extensively, compared with other configurations. It has the highest CPU usage for a single slave node.
- When the number of mappers and reducers increases, the CPU usage increases slightly using the same input size, such as B-1 vs. B2 and B3 vs. B4. Even though for E-X configurations there is a slight drop for some jobs such as WordCount and Scan, the values are still within the $95\%CI$ range. The reason for such behavior is due to over resource allocations since our testbed had limited hardware.
- When the number of slave nodes increases, the average CPU usage decreases in most our cases; for example, compare between D-1 and E-X.

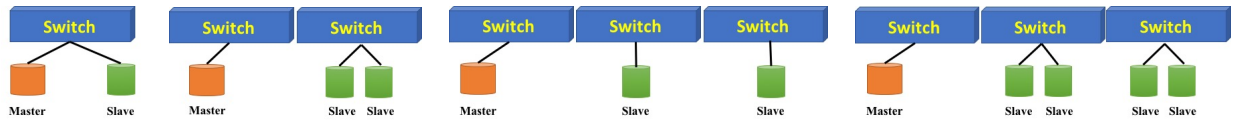


Fig. 4: Topology Type *(a) TA*, *(b) TB-1*, *(c) TB-2*, *(d) TC, TD*

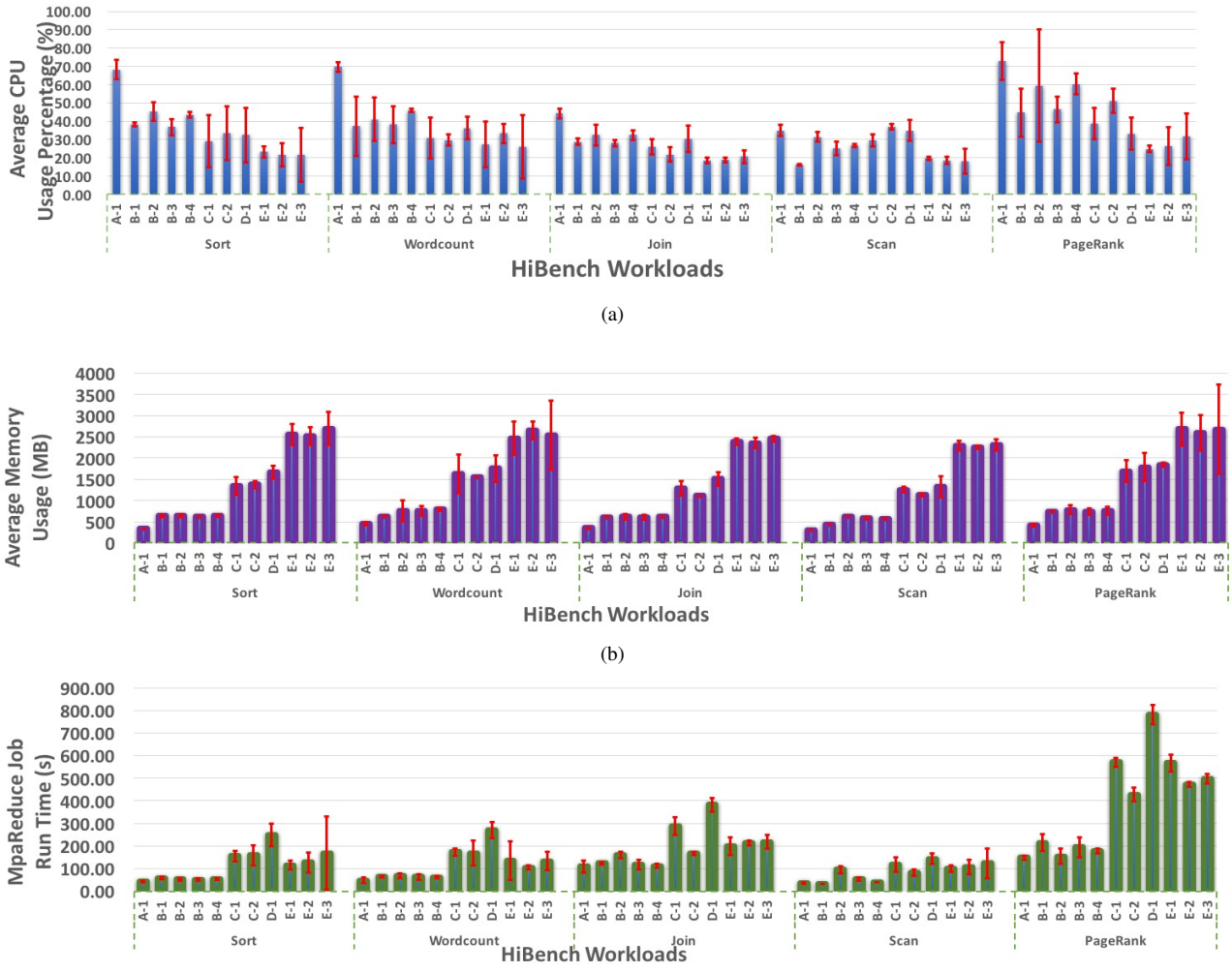


Fig. 5: *(a) CPU Load Summary*, *(b) Memory Load Summary*, *(c) Job Completion time Summary*

- The location of a slave node in our setup does not play a significant role in affecting the overall CPU usage with consistent bandwidth allocation on each interconnected link.
- The overall CPU usage has significant oscillation for the same set of test configurations even with no background processes running on the same system. The main reason is due to the efficiency of the JVM resource allocation and release in the MapReduce platform at the run time environment.
- PageRank has a higher CPU usage compared to other jobs, while others show similar CPU usage trends.

D. Average Memory Usage Summary

Fig. 5(b) depicts the average memory usage for various HiBench configurations. From the runs, we observed the following:

- A single slave had the lowest memory usage even with the lowest input data size when comparing configuration A-1 with others.
- The average memory usage increases dramatically when the data size increase. For example, the average memory usage increased nearly $500\%(\approx 2500/500)$ compared to six times in data growth$(\approx 304/48)$.
- The memory usage was in direct proportion to the number

of allocated mappers and reducers. However, in some cases, we noted opposing results, particularly in the C-2 and E-X cases. Even though they still fell into the calculated $95\%CI$ range, the reason for such behavior was due to over-resource allocations given our hardware's limitations.
- The overall memory usage had less oscillation for the same set of test configurations compared to the CPU usage.
- All of the jobs had the same level of memory usage when compared to the same set of configuration runs.

E. Job Completion Time: Comparison

Fig. 5(c) depicts the average job completion time for various configuration setups. From the runs, we observed the following:

- The input size had the largest impact on the average job completion time. The larger the data size, the longer it took, such as the comparison between C-X and D-1. However, with the same input size, more slave nodes reduced the overall time, such as the comparison between D-1 and E-X.
- The number of mappers and reducers for the same set of HiBench workloads also played an important role in the completion time in most configuration cases except for Scan. However, in some cases, it showed the opposite results such as in PageRank. The slight time increased from E-2 to E-3 was due to over resource allocations on the given hardware's limitation.
- Pagerank has a significant time increase compared with other workloads due to its CPU-intensive nature.

In summary, each HiBench workload behaved differently even with similar configuration setups. Clearly, understanding the different behaviors of various MapReduce jobs was an essential step for us for our SDN-based AAN environment for MapReduce applications.

IV. AAN-SDN Hadoop Architecture and Implementation

In this section, we present our proposed AAN-SDN platform design and implementation. For implementation of SDN, we used Ryu [13] and OpenFlow v1.3. We first introduced our layered SDN network architecture then conducted SDN-assisted MapReduce job completion time optimization. Starting from the bottom to the top layer, our proposed architecture (see Fig. 6) segregated our design into three main components:

1) Core SDN controller layer
2) Network control and monitor layer
3) Application-specific layer

A. Core SDN controller layer

In the core SDN layer, we implemented two network modules, Packet Forwarding, and Traffic Monitoring. In the packet forwarding module, we applied the network primary

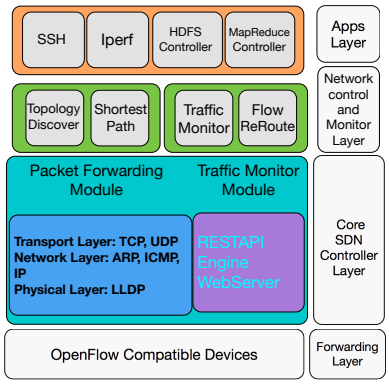


Fig. 6: SDN Hadoop Experimental Architecture

forwarding functions including the link layer discovery protocol (LLDP) in the physical network layer. The implementation was based on OpenFlow-compatible forwarding devices; in our cases, we used the Openvswitch (OVS) [14]. In the network layer, we implemented the forwarding function for the Internet Control Message Protocol (ICMP) messages, which is the key mechanism used to give feedback on network problems that could prevent packet delivery.

Algorithm 1: ARP Resolver Algorithm

```
Data: ARP Flows ARP_r
Result: ARP Processing Decision
        (Forwarding/Blocking/NoAction)
ARP cache initialization for each connect switch;
Read incoming ARP packets: arp;
if arp is ARP Broadcast then
    if ARP cache is empty then
        Add this arp entry to ARP Cache;
        Add expire timer to this arp entry;
        Flood this arp packet;
    else
        if arp exists then
            if arp entry timer =< ARP_Timer then
                Renew arp entry timer;
                Do not flood this arp packet;
            else
                Renew arp entry timer;
                Flood this arp packet;
            end
        else
            if arp is coming from a different port from existing
            arp entry then
                Do not flood this arp packet;
            else
                Add this arp entry to ARP Cache;
                Add expire timer to this arp entry;
                Flood this arp packet;
            end
        end
    end
else
    Forward ARP Request/Reply Packet;
end
```

Due to the flexibility provided by the SDN framework, we also addressed a new physical layer flooding avoidance mech-

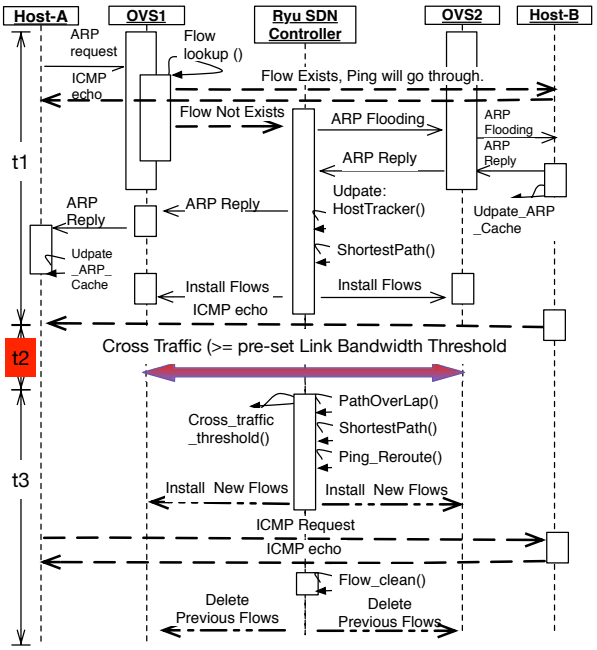


Fig. 7: Traffic reroute workflow example

anism such as for the address resolution protocol (ARP). In a traditional IP network, variations of spanning tree protocols (STP) are widely used to build a loop-free topology. The configuration of such an STP protocol can be cumbersome and complicated based on the used forwarding devices. We designed and implemented an ARP resolver algorithm 1 that offers smooth ARP package flooding, instead of using a costly STP protocol as would be the case in a traditional IP network environment. It also takes care of ARP cache expiration issues by avoiding to send additional ICMP messages to get an updated ARP entry.

Above the network transport layer, we implemented TCP and UDP packet forwarding functions for application-aware networking. Based on the application layer's port number and protocol type, it will forward packets accordingly. In the traffic monitor module, we implemented lightweight REST-API services to proactively fetch global network information such as port traffic for each forwarding device, flow installation/modification, and traffic details in a managed time interval. The REST-APIs are designed to be lightweight without introducing extra overhead for the SDN controller. One Apache web server collects the pulled results from the REST-APIs and aggregates traffic details to provide any traffic alerts and Traffic Engineering (TE) recommendations.

B. Network Control and Monitor Layer, and Adaptive Traffic Engineering

In the network control and monitoring layer, the global network topology was discovered where we took an adaptive traffic engineering approach by feeding into a shortest path algorithm module to calculate a path for each pair of network node/hosts on an on-demand basis. The traffic monitor component, using REST-APIs' services, deployed at the core SDN controller layer proactively pulled network traffic information from the network. If there were any pre-defined traffic priority violations, a traffic reroute using a flow reroute component might happen as explained in Fig. 7 that depicts the primary traffic reroute workflow. From the beginning, the ARP message for a network request such as Ping, SSH, or other applications. It first looks at the flow table and passes the traffic if there is an existing matching flow or checks if there are ARP broadcasting messages, otherwise.

Flows are installed based on the path provided by the shortest path components. Flow reroutes can happen when background traffic (at time $t2$) on the same route has over saturated some of the links along the path through adaptive traffic engineering. For this, the SDN controller recalculates a second shortest path in real time and installs new flows to reroute the application's traffic.

C. Application Layer

In the Application layer, a port number based application recognition feature is implemented (such as port 22 is by default for the remote secure shell (SSH)). In our controlled network, the port number can be managed/changed via a separate configuration file that is read by our SDN controller. With regards to Hadoop applications, HDFS and MapReduce control components are implemented to instruct how to install flows regarding Hadoop file operations and job assignments, accordingly. The modularization of various components provided by different SDN controllers helps the network administrator to control them individually in a manageable way.

In this controlled test environment, multiple applications can run together with separate configuration files. The monitoring and management modules of the SDN controller can control network traffic based on these configuration files. For example, if an application's port number or IP address gets changed when the application is running, the SDN controller can read the real time modification and continue to monitor and control the network flows without any modification delays. Compared to a traditional IP network management system, this method provides a clean and easy method for application monitoring and management.

V. MapReduce Traffic Optimization Using SDN

In this section, we report on experiments conducted using our proposed AAN-SDN platform to optimize MapReduce job running oversaturated network links.

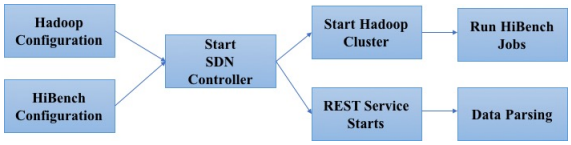
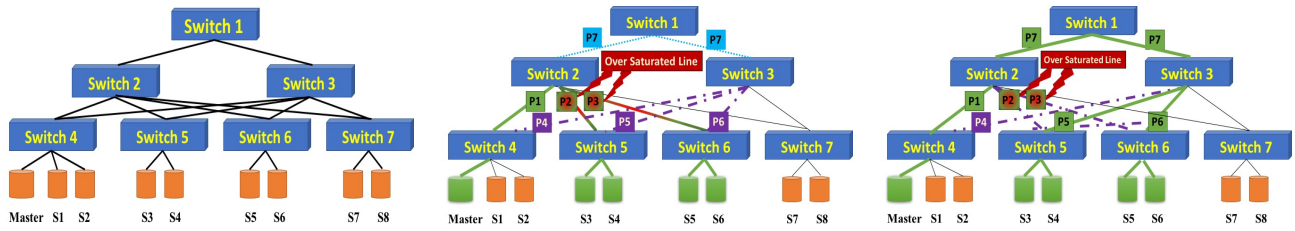


Fig. 8: Experimental Network Setup



(a) (b) (c)

Fig. 9: *(a) Network Topology, (b) Traffic Path Before Reroute , (c) Traffic Path After Reroute*

Fig. 10: Hadoop MapReduce Intermediate Data Transfer

A. System Workflow

Fig. 8 explains the workflow of our running system. After the Hadoop and HiBench configuration are completed, the SDN controller starts to run Hadoop MapReduce jobs. While such jobs are running, the SDN controller will install flows based on application types and collect network traffic information accordingly.

B. Network Topology

We deployed our SDN-based Hadoop and HiBench environment on the GENI testbed [15] platform. We setup a network topology as shown in Fig. 9(a) to emulate a data center network topology with hosts associated with different network switches that correspond to Hadoop nodes. We used Openvswitch (OVS v2.3.1) [13] as our forwarding devices and numbered the DPID in order from '1' to 7'. One Hadoop master node and eight slaves were deployed. The deployed nodes have the same hardware configuration with a single core of Intel(R) Xeon(R) CPU X5650 @ 2.67GHz and 8 GB RAM. Each connected link has 100 $Mbps$ bandwidth allocation.

C. Network Flow Traffic Capture

Network flow traffic is captured when running the HiBench workload using SDN by designed REST-API services, which proactively pulls global network traffic information based on any given port number. Table II shows the related port number regarding our MapReduce benchmark tests. For example, data shuffling traffic is captured during the map-reduce shuffle phase using port number 50010.

Fig. 10 depicts the data shuffling pattern for each workload. Detailed flow traffic is listed in Table V. The major traffic has been captured from port number 50010, the majority of this being from the MapReduce shuffling phase. We summarize our observations as follows:

TABLE II: Monitored port number

Traffic Catagory	Explanation	Port Number
HDFS	Hadoop File System Operation (HDFS)	50010
	Hadoop Datanode Transfer	22
	Master Node Http IPC	54310
YARN	Yarn Resource Scheduler	8030
	Yarn Resource Tracker	8031
	Yarn Resource Manager	8032
Others	Iperf	5001
	Secure Shell (SSH)	22

1) The shuffling data size is in direct proportion to the input data. Except for the WordCount workload, which has minimal increase, the others have significant data flows. This is to show that WordCount has a minimal shuffling data size. The other cases have more output data than the input size:
   a) The Sort workload has roughly 110% of the input data size that gets shuffled.
   b) In the Join workload, around 70% of the input data gets shuffled.
   c) In the Scan, around 120% of the input data gets shuffled.
   d) In the PageRank about 240% of the input data gets shuffled.
2) The number of mappers and reducers has no noticeable effect on the shuffling data sizes, such as the comparison between C-1 and C-2. The total shuffle size is the sum

of individual traffic among possible slave nodes.

To have a deeper understanding of what composes the shuffle traffic, we capture the data transfer for each pair of slave nodes. Due to the nature of the replication factor we set, even though the sum of the total shuffle traffic is within our calculated $95\%CI$ range, the pairs of slave nodes that generate the shuffle traffic are not fixed for each run. However, we list a network trace in Table VI to explain the different behaviors of each workload. By identifying the shuffling pattern, we understand what the traffic size is and which network link it takes to transfer the data. From our test results, we summarize our observations as follows:

1) The number of mappers and reducers plays important roles for shuffling pairs. For examples, in the B-1 and B-2 configuration with one and two mappers/reducers, respectively, there is only one pair of shuffling traffic on case B-1 but there might be two pairs on B-2. It mostly depends on the current system load and FIFO resource allocators.
2) The input data size also contributes to the number of shuffling pairs. For example, there is a significant number of pairs that increases from configuration B-X to E-X.

Consider workload Join with the E-3 test case for an example to note the randomness of the flow traffic pattern; the detailed traffic direction is listed in Table VI. There are 53 sets of individual shuffle pairs. The uncertainty of MapReduce traffic is not easy to address by a traditional IP network with less individual traffic flow control.

D. ARP Flooding Avoidance using ARP Resolver

Our proposed ARP resolver method (see Algorithm 1) provides smooth ARP cache expiration and packet flooding, instead of using a costly STP protocol as would be the case in a traditional IP network environment. We implemented the default ARP packet defined in RFC 826, which is 28 bytes. We then conducted a measurement on how the proposed ARP can avoid flooding issues using the minimum ARP request packets. If we use slave nodes S1 and S7 as an example, the shortest path is S1→Switch 4→Switch 2→Switch 7→S7. If S1 sends an ARP request, an ARP flooding packet must be sent out if the previous ARP cache has expired. If that is the case, the ARP flooding packet will be forwarded to all network forwarding devices by each other.

Based on our experimental result shown in Fig 11, the forwarding switch's CPU can run almost at 100% utilization without any ARP flooding avoidance methods, which causes the network to stop functioning properly. However, it only takes 20 ARP flooding packets (Each ARP broadcasting packet can only be seen twice by the connecting port) to travel through the network by using our proposed ARP resolver. With the minimum number of flooding packets, the proposed ARP resolver algorithm can minimize the flooding traffic and destination searching time, without overwhelming the forwarding switch's CPU.

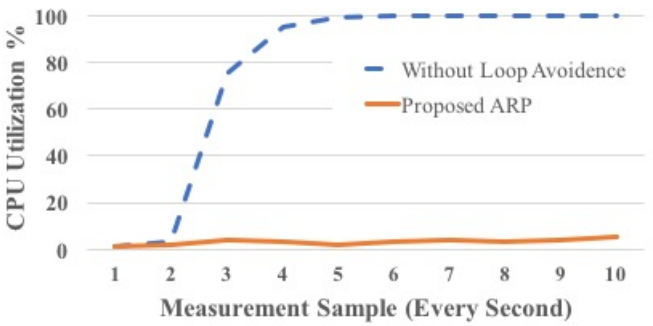


Fig. 11: ARP Broadcasting Packet Cause CPU Utilization High in a Loop Topology

E. SDN Traffic Reroute through Adaptive Traffic Engineering

The behavior of the data shuffle phase for an individual MapReduce workload has shown various data sizes and patterns based on our study. One can assume that if such a workload was running on a busy network link, the increase in delay would be expected. A traditional IP network lacks real-time global traffic information updates and the network administrator has less control over a specific network flow with the minimum cost. With regards to Hadoop MapReduce applications, data could shuffle from any pair of running slave nodes. If any delay happens on any of the shuffle phases, the overall job completion time can be prolonged.

Our goal of running Hadoop MapReduce jobs using the proposed AAN-SDN architecture is to investigate how much SDN can alleviate from a busy Hadoop cluster based on the different sizes of input data and the number of slave nodes. Table III shows experiment scenarios and test results. Two cases are considered. The first is $48MB$ input size with two slaves nodes and the second is $240MB$ input size with four slave nodes. To simulate a busy cluster situation, we induced background traffic using *iperf* on selected links.

Fig. 9(b) depicts the MapReduce job path utilization before reroute. The running Hadoop nodes are marked in green colors. They are the master node and S3, S4, S5, and S6 are the slave nodes. For reroute test case 1 (shown in Table III), slave nodes S3 and S5 ran HiBench workloads. The shortest path module from the SDN platform installed flows along the path [**P1, P2, P3**] at the beginning of the system and ran in order to start the Hadoop cluster and start MapReduce jobs. Meanwhile, path [**P1, P7, P5, P6**] was saved as a backup shortest path between the master node and slave nodes S3 and S5.

Iperf background traffic runs on the path [**P2, P3**] consuming $90Mbps$ bandwidth, which is our predefined threshold for any flow reroute scenario. We first disabled the reroute module and forced the MapReduce jobs to run on the oversaturated links between S3 and S5 to emulate a static environment. When the reroute module is enabled, the SDN controller detects there is data flow over an acceptable threshold of background traffic along the first pair of the shortest path. New flows are installed using the backup path [**P1, P7, P5, P6**] to avoid any potential delays as shown in Fig. 9(c).

TABLE III: HiBench Workloads Traffic Reroute Using SDN

Workloads	Sort		WordCount		Join		Scan		Pagerank	
	ReRoute Test 1	ReRoute Test 2	ReRoute Test 1	ReRoute Test 2	ReRoute Test 1	ReRoute Test 2	ReRoute Test 1	ReRoute Test 2	ReRoute Test 1	ReRoute Test 2
Slaves	S3 S5	S3 S4 S5 S6	S3 S5	S3 S4 S5 S6	S3 S5	S3 S4 S5 S6	S3 S5	S3 S4 S5 S6	S3 S5	S3 S4 S5 S6
Input Data Size (MB)	48	240	48	240	48	240	48	240	48	240
No Background Traffic with path [P1, P2, P3]	49±2	338±56	56±3	220±17	171±12	423±38	116±7	146±19	190±18	713±33
Background Traffic with path[P1, P2, P3]	66±4	483±40	75±2	393±27	216±4	616±63	151±27	208±43	256±56	1022±224
SDN Reroute Path[P1, P5, P6, P7]	55±2	360±62	61±3	234±18	187±10	452±30	130±3	167±34	212±7	745±20
Reroute Time Consumption	7±1	24±10	8±5	14±27	16±13	29±15	14±5	21±14	23±16	31±15
Improvement	20.00%	34.00%	22.00%	68.00%	16.00%	36.00%	16.00%	25.00%	20.00%	337.00%

In summary, Fig. 12 shows that with our adaptive traffic engineering approach through rerouting, the MapReduce Job completion time can be reduced by 16% to 300% depending on test case configurations. The improvement is varied and it depends on the different behaviors of each MapReduce job. Consider the PageRank for example; its shuffle phase has over 200% more data output compared with the input data size. It also has the most run time efficacy improvement in terms of job completion time. Even though the improvements for other jobs are not as significant as PageRank, it still shows an increasing trend when the data input size increases.

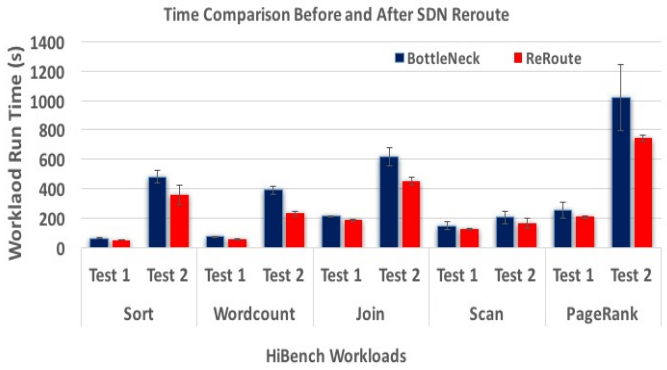


Fig. 12: Hadoop Job Run Time Comparison

As our result shows, our AAN platform for Hadoop MapReduce job optimization offer a significant improvement compared to a static, traditional IP network environment. Our design can be extended to other MapReduce jobs and various network topology without much additional complexity.

VI. Related Work

Hadoop MapReduce [9], [12], [16] as a distributed processing framework has become the dominant approach for processing volumetric traffic in the big data era. Many researchers have studied several options to improve MapReduce's performance. Recent work, using traditional IP networks, can be grouped into two categories within given hardware resources: (1) an advanced Hadoop resource scheduling algorithm design in [2]–[4], [17]–[22] and (2) job optimization with optimized configuration parameters using specific hardware in [23]–[25].

The Existing Hadoop MapReduce resource scheduling algorithm manages to optimize the Hadoop cluster resources such as slave nodes, CPUs, memories, networks, and disks. Those algorithms fall mainly into three categories: FIFO, capacity-based, and fairness schedulers. There are also heuristic designs that focus on data locality with simulations. BAR [2] proposed a heuristic task scheduling based on data locality by simulation, by initially finding an ideal data location for job processing in order to reduce the job running time. However, the assumption of initial job starting and completion time cannot stand in a real network. The proposed wait and random peeking approach in [3] and a fair scheduler [4] improves data locality. However, those methods can be further improved if integrated with network information and by using real-time data traces. SHadoop [26] takes an approach of modifying standard Hadoop's map and reducing execution methods to avoid employing any particular hardware or supporting software. Other similar works, which use adaptive job performance scheduling under various cluster circumstances are in [22], [27].

Another aspect of improving MapReduce job completion time can be achieved using a hardware acceleration approach [24], [28]. Special software and hardware need to be deployed that may not be readily accessible for normal cluster setups. Job specific optimization for MapReduce works is presented in [23], [29]. However, it lacked generalized methods for the overall performance of MapReduce jobs.

By using the Hadoop cluster under the traditional IP network, MapReduce's performance can be substantially degraded due to (1) the inherent characteristic of intensive data shuffle frameworks to transfer a large amount of intermediate data among slave nodes, and (2) default resources' allocation methods that lack the global view of real-time network traffic information. TCP related optimization work for MapReduce workloads is invested in [30]–[32], but the overall operation still bears low-performance improvement.

New network frameworks have been studied to identify new approaches to achieve a better MapReduce performance, such as in MROrchestrator [33], Coflow [34] and Orchestra [35] with much more sophisticated application integration designs. Pythia [36] has similarities with our approach but lacks a clear and comprehensive SDN system design with respect to MapReduce and a related application-aware approach. A preliminary idea on our approach was presented in [37]. Our application-Aware network design on top of SDN provides a common API interface, which can provide a full range of capabilities for network management and monitoring for different applications. We also measure the SDN control latency cost in our test cases. Based on the test results, our AAN realization can be better utilized for applications such as Hadoop M/R,

which does not rely on a low lantency requirement.

VII. Conclusion and Future Work

Our approach in this work was to develop an Application-aware networking (AAN) environment with modularized components and fine-grained control network behavior to improve MapReduce job performance without changing the underlying design of Hadoop MapReduce itself. We presented a software-defined network (SDN) based system architecture and over-the-top (OTT) software applications for MapReduce data flow control. For example, we proposed a general application interface regarding a traffic flow alternation mechanism. We also identified major traffic patterns of various MapReduce workloads based on HiBench benchmark suites using different configurations, which is the key to understanding the problem caused by data shuffle. A primary goal of our work was to demonstrate a concept of AAN using SDN implementation and MapReduce performance improvement without alternating the existing Hadoop configuration.

Due to the system limitation of the GENI platform used for our work, we were limited in our ability to use very large scale big data analytics test cases that use MapReduce. Nevertheless, we considered 330 test cases to present a robust assessment of the traffic patterns, and demonstrated how our AAN-SDN network softwarization approach with adaptive traffic engineering can lead to a noticeable reduction in the job completion time. In the future, we plan to explore additional traffic engineering (TE) methods as well as larger platforms for our approach.

References

[1] D. Borthakur, "The Hadoop distributed file system: Architecture and design," 2007, the Apache Software Foundation.

[2] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "Bar: An efficient data locality driven task scheduling algorithm for cloud computing," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2011, pp. 295–304.

[3] J. Tan, X. Meng, and L. Zhang, "Coupling task progress for MapReduce resource-aware scheduling," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 1618–1626.

[4] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 265–278.

[5] D. Kreutz, F. M. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[6] M. Pióro and D. Medhi, *Routing, flow, and capacity design in communication and computer networks*. Elsevier, 2004.

[7] 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*, vol. 38, no. 2, pp. 69–74, 2008.

[8] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, "Apache Hadoop YARN: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[9] T. White, "Hadoop: The definition guide," 2009.

[10] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 2010, pp. 41–51.

[11] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 165–178.

[12] A. Holmes, *Hadoop in practice*. Manning Publications Co., 2012.

[13] RYU, "Osrg framework community: Ryu sdn controller." [Online]. Available: https://osrg.github.io/ryu/

[14] OpenvSwitch, "Open vswitch, an open virtual switch." [Online]. Available: http://openvswitch.org/

[15] C. Elliott, "Geni-global environment for network innovations." in *LCN*, 2008, p. 8.

[16] C. Lam, *Hadoop in action*. Manning Publications Co., 2010.

[17] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments." in *Osdi*, vol. 8, no. 4, 2008, p. 7.

[18] H.-H. You, C.-C. Yang, and J.-L. Huang, "A load-aware scheduler for MapReduce framework in heterogeneous cloud environments," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 127–132.

[19] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–9.

[20] R. Vernica, A. Balmin, K. S. Beyer, and V. Ercegovac, "Adaptive MapReduce using situation-aware mappers," in *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, 2012, pp. 420–431.

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

[22] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new MapReduce scheduling technique," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 40–47.

[23] B. Li, E. Mazur, Y. Diao, A. McGregor, and P. Shenoy, "A platform for scalable one-pass analytics using MapReduce," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 985–996.

[24] M. Xin and H. Li, "An implementation of GPU accelerated MapReduce: Using Hadoop with OpenCL for data-and compute-intensive jobs," in *Service Sciences (IJCSS), 2012 International Joint Conference on*. IEEE, 2012, pp. 6–11.

[25] S. Babu, "Towards automatic optimization of MapReduce programs," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 137–142.

[26] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, and Y. Huang, "SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters," *Journal of parallel and distributed computing*, vol. 74, no. 3, pp. 2166–2179, 2014.

[27] H. Mao, S. Hu, Z. Zhang, L. Xiao, and L. Ruan, "A load-driven task scheduler with adaptive DSC for MapReduce," in *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*. IEEE, 2011, pp. 28–33.

[28] Y. Becerra, V. Beltran, D. Carrera, M. González, J. Torres, and E. Ayguadé, "Speeding up distributed MapReduce applications using hardware accelerators," in *Parallel Processing, 2009. ICPP'09. International Conference on*. IEEE, 2009, pp. 42–49.

[29] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng, "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–8.

[30] Y. Chen, R. Griffit, D. Zats, and R. H. Katz, "Understanding tcp incast and its implications for big data workloads," *University of California at Berkeley, Tech. Rep*, 2012.

[31] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 3–3.

[32] W. Yu, Y. Wang, and X. Que, "Design and evaluation of network-levitated merge for Hadoop acceleration," *ieee transactions on parallel and distributed systems*, vol. 25, no. 3, pp. 602–611, 2014.

[33] B. Sharma, R. Prabhakar, S.-H. Lim, M. T. Kandemir, and C. R. Das, "Mrorchestrator: A fine-grained resource orchestration framework for

MapReduce clusters," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 1–8.

[34] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 31–36.

[35] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 98–109.

[36] M. V. Neves, C. A. De Rose, K. Katrinis, and H. Franke, "Pythia: Faster big data in motion through predictive software-defined network optimization at runtime," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 82–90.

[37] S. Zhao, A. Sydney, and D. Medhi, "Building application-aware network environments using SDN for optimizing Hadoop applications," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference (poster paper)*. ACM, 2016, pp. 583–584.

Shuai Zhao is a Ph.D. Candidate in the Department of Computer Science & Electrical Engineering at the University of Missouri Kansas City, USA. He received B.Sc in Applied Mathematics from Heze University, China, M.S. in Computer Science from the University of Missouri–Kansas City, USA. He currently works as a senior engineer at Mediatek USA Inc. He is an IEEE member and serves (or served) as a Technical Committee Member for IEEE NOMS, ICNC, NFV-SDN, ICIIP, Globecom, ANTS, NetCom, NoF, ICCC, WPMC, ICUMT, CyberNeticsCom, IPCCC, CNSM and designated reviewer for IEEE ICC, VCIP and ACM MMSys. He interned with Lenovo (2013), FutureWei (2014), Raython BBN (2015) and Mediatek Inc USA (2016). He has published 12 papers. His research interests are Application-Awareness Networking, Software-Defined Networking, network design, big data, deep learning, MPEG-DASH and MPEG-I Standards, and VR/360 video streaming.

Deep Medhi is Curators' Distinguished Professor in the Department of Computer Science Electrical Engineering at the University of Missouri-Kansas City, USA, and an honorary professor in the Department of Computer Science & Engineering at the Indian Institute of Technology–Guwahati, India. He received B.Sc. in Mathematics from Cotton College, Gauhati University, India, M.Sc. in Mathematics from the University of Delhi, India, and his Ph.D. in Computer Sciences from the University of Wisconsin-Madison, USA. Prior to joining UMKC in 1989, he was a member of the technical staff at AT&T Bell Laboratories. He served as an invited visiting professor at the Technical University of Denmark, a visiting research fellow at Lund Institute of Technology, Sweden, and State University of Campinas, Brazil. As a Fulbright Senior Specialist, he was a visitor at Bilkent University, Turkey, and Kurukshetra University, India. He is the Editor-in-Chief of Springer's *Journal of Network and Systems Management*, and serves (or served) on the editorial board of *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Network and Service Management*, *IEEE Communications Surveys & Tutorials*, *Telecommunications Systems*, *Computer Networks*, and *IEEE Communications Magazine*. He has published over 150 papers, and is co-author of the books, *Routing, Flow, and Capacity Design in Communication and Computer Networks* (2004) and *Network Routing: Algorithms, Protocols, and Architectures* (1st edition in 2007, and 2nd edition in 2018), published by Morgan Kaufmann Publishers, an imprint of Elsevier Science. His research interests are: multi-layer networking; network virtualization; data center optimization; network routing, design, and survivability; and video quality-of-experience. His research has been funded by NSF and DARPA.

TABLE IV: Hibench Workload: Hardware Related Data Collection

Configurations	A		B-1		B-2		B-3		B-4		C-1		C-2		D-1		E-1		E-2		E-3	
	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI	Avg	95% CI
Sort																						
Time	42.00	**4.30**	58.67	**6.25**	52.67	**6.25**	51.67	**3.79**	52.33	**3.79**	155.00	**22.77**	158.67	**45.83**	248.33	**49.33**	115.67	**18.97**	127.00	**43.53**	169.00	**160.09**
CPU	68.17	**5.17**	38.27	**1.23**	45.33	**5.09**	36.80	**4.35**	43.70	**1.51**	29.10	**14.33**	33.53	**14.62**	32.37	**14.90**	23.17	**3.11**	21.67	**6.48**	21.67	**14.69**
Memory (MB)	342.43	**7.40**	638.60	**31.18**	641.13	**44.53**	620.70	**10.82**	641.13	**13.53**	1338.67	**205.58**	1380.97	**85.03**	1661.17	**153.65**	2556.47	**254.89**	2517.37	**205.55**	2694.87	**387.48**
WordCount																						
Time	47.64	**12.40**	64.72	**4.29**	69.12	**10.93**	62.90	**13.81**	62.20	**6.44**	173.07	**17.08**	167.54	**55.09**	269.42	**34.56**	135.14	**85.04**	104.03	**8.61**	133.07	**40.71**
CPU	69.67	**2.73**	37.17	**16.37**	41.03	**11.80**	38.13	**10.13**	45.87	**0.94**	30.93	**11.18**	29.63	**3.11**	36.27	**6.02**	27.20	**12.44**	33.30	**5.19**	26.07	**17.27**
Memory (MB)	457.13	**33.30**	631.57	**3.19**	761.13	**249.26**	760.87	**110.73**	797.63	**38.83**	1623.07	**454.34**	1552.73	**7.50**	1753.57	**309.09**	2466.47	**390.76**	2659.37	**209.67**	2533.60	**812.82**
Join																						
Time	108.97	25.89	123.22	4.38	160.40	14.21	117.85	21.37	111.69	9.53	288.49	38.53	167.75	8.13	383.02	30.65	200.23	38.76	213.25	10.19	217.93	31.11
CPU	44.20	2.59	28.70	1.72	32.47	5.72	28.03	1.80	32.37	2.74	26.00	4.24	21.80	3.76	30.33	7.35	18.33	1.52	18.87	1.37	20.67	3.49
Memocy (MB)	352.03	6.34	613.20	10.97	615.73	64.34	599.00	56.28	616.27	11.14	1289	162.99	1115.30	8.41	1510.03	166.02	2393.27	74.33	2357.73	122.34	2454.17	61.67
Scan																						
Time	37.06	3.01	33.03	1.06	94.95	15.89	53.89	6.62	38.77	0.91	117.22	32.34	81.25	13.52	143.82	22.66	99.77	14.34	106.57	31.39	123.03	66.02
CPU	34.97	3.28	16.30	0.43	31.30	2.62	25.00	3.82	26.67	1.03	29.43	3.43	36.97	1.46	34.8	5.68	19.63	0.87	18.67	1.93	18.20	6.76
Memocy (MB)	298.73	4.83	432.53	6.25	623.40	5.39	592.13	18.42	574.83	5.96	1258.20	70.75	1144.07	38.60	1322.47	243.37	2293.23	110.99	2256.8	42.53	2311.87	124.70
Pagerank																						
Time	148.55	9.55	214.96	36.21	153.65	33.36	194.66	44.4	177.75	9.23	571.13	19.67	427.82	31.55	781.25	41.6	567.69	37.02	473.73	11.02	496.78	20.52
CPU	72.80	10.24	44.80	13.17	59.37	30.66	46.37	6.85	60.27	5.74	38.70	8.35	51.17	6.39	33.17	8.68	24.70	1.72	26.40	10.25	31.83	12.51
Memocy (MB)	424.97	24.46	742.13	8.32	784.13	97.23	731.07	76.85	756.87	88.44	1693.47	261.40	1785.07	333.00	1841.30	46.89	2681.37	390.61	2595.93	415.55	2677.80	1044.33

TABLE V: Mapreduce Workload Traffic Flow Data Summary

Configurations		A		B-1		B-2		B-3		B-4		C-1		C-2		D-1		E-1		E-2		E-3	
Traffic Catagories	**Port**	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI	Avg	95%CI
Sort																							
Slaves To Master	54310	0.06	0.051	0.098	0.087	0.074	0.024	0.097	0.151	0.091	0.092	0.154	0.056	0.251	0.137	0.236	0.525	0.288	0.418	0.216	0.062	0.358	0.197
	8031	0.037	0.043	0.093	0.121	0.049	0.011	0.071	0.183	0.053	0.103	0.154	0.105	0.317	0.246	0.304	0.667	0.341	0.742	0.239	0.176	0.416	0.15
	8030	0.01	0.006	0.016	0.012	0.012	0.002	0.014	0.012	0.015	0.009	0.025	0.038	0.033	0.008	0.043	0.093	0.036	0.021	0.037	0.018	0.044	0.028
Slaves To Slaves	50010	0	0	50.634	0.242	50.642	0.135	50.578	0.666	51.23	0.639	253.237	3.877	256.354	4.173	355.036	10.221	314.606	7.945	318.393	7.668	319.712	14.65
Others	22	0.006	0.003	0.011	0.003	0	0	0.003	0.014	0.004	0.017	0.011	0.049	0	0	0	0	0.027	0.117	0	0	0	0
	8032	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WordCount																							
Slaves To Master	54310	0.060	0.054	0.092	0.099	0.096	0.046	0.082	0.119	0.079	0.018	0.161	0.042	0.191	0.141	0.298	0.104	0.266	0.222	0.245	0.236	0.311	0.182
	8031	0.036	0.049	0.072	0.074	0.064	0.037	0.053	0.002	0.05	0.046	0.155	0.164	0.173	0.132	0.359	0.11	0.248	0.253	0.255	0.441	0.278	0.329
	8030	0.011	0.007	0.016	0.011	0.014	0.001	0.015	0.009	0.013	0.001	0.03	0.01	0.032	0.019	0.054	0.019	0.044	0.014	0.042	0.036	0.037	0.003
Slaves To Slaves	50010	0	0	0.913	0.035	0.686	0.038	0.665	0.001	0.68	0.002	1.561	0.4	1.363	0.351	2.373	0.522	3.063	0.847	3.488	0.511	3.706	0.885
Others	22	0.005	0.002	0.002	0.008	0.016	0.018	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8032	0.019	0.083	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Join																							
Slaves To Master	54310	0.137	0.138	0.179	0.261	0.150	0.059	0.189	0.230	0.137	0.014	0.401	0.315	0.297	0.141	0.381	0.195	0.461	0.737	0.478	0.292	0.442	0.348
	8031	0.081	0.066	0.113	0.19	0.129	0.031	0.148	0.125	0.119	0.035	0.395	0.069	0.271	0.219	0.461	0.02	0.517	1.326	0.532	0.567	0.646	0.387
	8030	0.024	0.026	0.032	0.033	0.022	0.001	0.03	0.029	0.022	0	0.058	0.029	0.06	0.026	0.066	0.02	0.061	0.036	0.049	0.017	0.056	0.03
Slaves To Slaves	50010	0	0	35.886	2.624	34.774	1.765	36.278	3.007	35.182	1.745	85.577	7.607	105.285	1.128	101.337	19.675	242.593	5.054	242.128	8.268	238.858	20.447
Others	22	0.003	0.014	0.006	0.026	0	0	0	0	0	0	0.012	0.05	0	0	0	0	0.028	0.12	0	0	0	0
	8032	0	0	0	0	0	0	0.006	0.027	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Scan																							
Slaves To Master	54310	0.077	0.151	0.151	0.248	0.062	0.006	0.125	0.241	0.066	0.011	0.311	0.410	0.200	0.097	0.363	0.458	0.353	0.228	0.474	0.430	0.269	0.126
	8031	0.052	0.102	0.12	0.193	0.049	0.05	0.112	0.194	0.076	0.013	0.279	0.324	0.197	0.086	0.32	0.303	0.281	0.266	0.385	0.545	0.236	0.077
	8030	0.012	0.025	0.019	0.03	0.008	0.001	0.018	0.03	0.008	0.001	0.035	0.048	0.02	0.003	0.043	0.039	0.041	0.038	0.039	0.038	0.027	0.003
	50010	0	0	59.08	1.69	59.448	0.277	59.768	0.032	59.768	0.108	289.482	1.568	298.707	26.64	439.974	24.981	439.016	35.091	465.416	28.026	451.201	17.702
Others	22	0	0	0	0	0	0	0	0	0	0	0	0	103.318	444.542	0	0	0	0	0	0	0	0
	8032	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pagerank																							
Slaves To Master	54310	0.138	0.132	0.216	0.204	0.186	0.088	0.232	0.133	0.238	0.291	0.587	0.440	0.471	0.081	0.841	0.214	0.662	0.811	0.683	0.161	0.748	0.197
	8031	0.09	0.059	0.198	0.182	0.169	0.12	0.199	0.076	0.202	0.188	0.522	0.061	0.533	0.157	0.817	0.132	0.615	0.955	0.81	0.2	0.926	0.172
	8030	0.021	0.045	0.056	0.021	0.049	0.011	0.057	0.028	0.063	0.07	0.121	0.023	0.1	0.016	0.195	0.07	0.134	0.048	0.123	0.025	0.114	0.069
	50010	0	0	127.878	4.058	129.949	5.251	130.879	2.618	127.304	6.722	579.303	18.069	571.967	6.101	757.69	9.565	718.863	16.586	732.344	39.643	738.479	5.218
Others	22	0.003	0.014	0.006	0.027	0	0	0.005	0.022	0	0	0.012	0.053	0	0	0	0	0.063	0.271	0	0	0	0
	8032	0.015	0.065	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE VI: Example of MapReducde Shuffle Traffic For Each Flow

Workloads	Configuration	Number of Shuffle Pairs	Total Data Size (MB)	% Of Flow#1	% Of Flow#2	% Of Flow#3	% Of Flow#4	% Of Flow#5	% Of Flow#6	% Of Flow#7	% Of Flow#8	% Of Flow#9	Others
Sort	A-1	0	-	-	-	-	-	-	-	-	-	-	-
	B-1	1	50.54	98.75%	1.25%	-	-	-	-	-	-	-	-
	B-2	2	50.63	50.50%	49.50%	-	-	-	-	-	-	-	-
	B-3	2	50.70	99.48%	0.52%	-	-	-	-	-	-	-	-
	B-4	2	50.99	100.00%	0.00%	-	-	-	-	-	-	-	-
	C-1	9	252.20	24.92%	24.92%	13.06%	12.98%	12.01%	11.93%	0.10%	0.08%	0.00%	0.00%
	C-2	12	257.53	24.88%	12.93%	12.51%	12.45%	12.34%	12.32%	12.31%	0.10%	0.10%	0.06%
	D-1	12	353.21	29.87%	20.69%	13.24%	11.36%	9.38%	9.37%	4.04%	1.97%	0.08%	0.00%
	E-1	33	302.05	10.85%	10.82%	10.82%	10.81%	10.77%	10.75%	9.09%	6.40%	4.82%	14.87%
	E-2	28	321.83	14.63%	14.43%	10.29%	10.28%	10.19%	10.18%	10.18%	10.17%	4.56%	5.08%
	E-3	30	324.83	20.27%	12.49%	12.29%	10.31%	10.24%	10.01%	9.98%	4.61%	2.28%	7.52%
	Traffic Node Pairs For E-3			(S3→S8)	(S7→S2)	(S8→S4)	(S7→S1)	(S8→S5)	(S6→S8)	(S5→S6)	(S3→S8)	(S6→S7)	-
Wordcount	A-1	0	-	-	-	-	-	-	-	-	-	-	-
	B-1	2	0.93	68.60%	31.40%	-	-	-	-	-	-	-	-
	B-2	2	0.70	86.05%	13.95%	-	-	-	-	-	-	-	-
	B-3	2	0.66	56.84%	43.16%	-	-	-	-	-	-	-	-
	B-4	2	0.68	59.47%	40.53%	-	-	-	-	-	-	-	-
	C-1	10	1.74	35.21%	21.13%	18.09%	15.61%	7.63%	1.01%	0.43%	0.35%	0.10%	0.43%
	C-2	10	1.25	27.30%	27.06%	25.48%	11.61%	7.04%	0.92%	0.28%	0.16%	0.13%	0.04%
	D-1	11	2.15	21.57%	19.03%	14.41%	12.35%	12.22%	7.29%	4.59%	4.37%	3.84%	0.34%
	E-1	31	2.70	15.98%	9.92%	9.83%	9.76%	9.72%	9.53%	6.08%	5.88%	4.53%	18.78%
	E-2	27	3.58	36.31%	10.48%	9.41%	7.65%	7.56%	7.37%	5.20%	4.27%	3.76%	7.99%
	E-3	33	3.88	20.95%	14.40%	13.33%	9.27%	7.19%	6.86%	6.81%	4.10%	4.06%	13.02%
	Traffic Node Pairs For E-3			(S5→S3)	(S5→S7)	(S5→S4)	(S3→S7)	(S6→S3)	(S5→S3)	(S7→S8)	(S4→S7)	(S8→S4)	-
Join	A-1	0	-	-	-	-	-	-	-	-	-	-	-
	B-1	2	35.28	98.88%	1.12%	-	-	-	-	-	-	-	-
	B-2	2	35.53	95.84%	4.16%	-	-	-	-	-	-	-	-
	B-3	2	35.60	98.09%	1.91%	-	-	-	-	-	-	-	-
	B-4	2	35.57	97.68%	2.32%	-	-	-	-	-	-	-	-
	C-1	12	87.79	21.11%	19.67%	19.58%	19.52%	19.21%	0.34%	0.25%	0.22%	0.07%	0.04%
	C-2	12	105.27	32.20%	16.41%	16.38%	16.29%	16.03%	1.56%	0.40%	0.24%	0.17%	0.31%
	D-1	12	105.91	32.25%	16.55%	16.28%	16.09%	15.90%	1.79%	0.56%	0.19%	0.19%	0.20%
	E-1	39	243.42	14.07%	7.58%	7.06%	7.04%	7.03%	7.02%	6.99%	6.98%	6.97%	29.26%
	E-2	37	243.45	7.25%	7.25%	7.09%	7.07%	7.05%	7.03%	7.01%	7.01%	6.99%	36.26%
	E-3	53	243.88	14.05%	7.09%	7.07%	7.04%	7.04%	7.01%	7.00%	6.99%	6.99%	29.72%
	Traffic Node Pairs For E-3			(S4→S8)	(S2→S6)	(S7→S1)	(S7→S3)	(S1→S2)	(S5→S2)	(S3→S4)	(S6→S7)	(S2→S5)	-
Scan	A-1	0	-	-	-	-	-	-	-	-	-	-	-
	B-1	2	59	99.99%	0.01%	-	-	-	-	-	-	-	-
	B-2	2	59	64.99%	35.01%	-	-	-	-	-	-	-	-
	B-3	2	59.76	100.00%	0.00%	-	-	-	-	-	-	-	-
	B-4	2	59.66	64.17%	35.83%	-	-	-	-	-	-	-	-
	C-1	11	289.02	25.42%	23.45%	21.67%	17.85%	5.90%	5.50%	0.14%	0.06%	0.00%	0.00%
	C-2	11	297.57	25.32%	15.74%	15.66%	10.70%	9.97%	7.26%	6.42%	5.73%	3.17%	0.03%
	D-1	9	451.50	29.20%	17.09%	15.85%	12.93%	12.32%	7.81%	4.80%	0.00%	0.00%	0.00%
	E-1	39	445.73	14.65%	13.32%	11.14%	9.75%	7.86%	6.77%	5.45%	5.29%	4.27%	21.50%
	E-2	33	458.39	14.81%	11.54%	11.00%	7.70%	7.25%	7.14%	6.72%	5.85%	5.24%	22.76%
	E-3	37	447.17	17.03%	11.25%	7.80%	7.40%	6.74%	5.98%	5.10%	4.94%	4.40%	29.36%
	Traffic Node Pairs For E-3			(S7→S3)	(S4→S7)	(S5→S8)	(S4→S8)	(S5→S2)	(S4→S1)	(S8→S6)	(S8→S4)	(S7→S2)	-
PageRank	A-1	0	-	-	-	-	-	-	-	-	-	-	-
	B-1	2	126.15	99.49%	0.51%	-	-	-	-	-	-	-	-
	B-2	2	128.15	50.41%	49.59%	-	-	-	-	-	-	-	-
	B-3	2	130.24	97.05%	2.95%	-	-	-	-	-	-	-	-
	B-4	2	130.28	51.32%	48.68%	-	-	-	-	-	-	-	-
	C-1	12	587	14.34%	12.34%	11.26%	11.20%	11.12%	9.24%	6.42%	6.30%	5.71%	12.07%
	C-2	12	574.44	18.43%	18.18%	13.93%	12.56%	11.73%	6.48%	6.42%	6.38%	5.72%	0.17%
	D-1	12	757.74	27.41%	18.65%	14.42%	14.31%	12.97%	4.36%	3.13%	3.03%	0.60%	1.13%
	E-1	47	726.34	9.09%	9.04%	8.92%	7.61%	5.91%	4.61%	4.58%	4.56%	4.54%	41.14%
	E-2	40	738.04	13.23%	8.87%	8.86%	8.82%	8.81%	5.82%	4.54%	4.52%	4.48%	32.06%
	E-3	39	737.60	16.28%	11.78%	8.91%	8.87%	7.66%	7.37%	7.35%	7.34%	7.34%	17.11%
	Traffic Node Pairs For E-3			(S8→S2)	(S8→S7)	(S3→S8)	(S6→S7)	(S1→S8)	(S4→S8)	(S6→S3)	(S8→S5)	(S7→S4)	-