Programming Your Network at Run-time for Big Data Applications

Guohui Wang*, T. S. Eugene Ng†, Anees Shaikh*
*IBM T.J. Watson Research Center, †Rice University

ABSTRACT

Recent advances of software defined networking and optical switching technology make it possible to program the network stack all the way from physical topology to flow level traffic control. In this paper, we leverage the combination of SDN controller with optical switching to explore the tight integration of application and network control. We particularly study the run-time network configuration for big data applications to jointly optimize application performance and network utilization. We use Hadoop as an example to discuss the integrated network control architecture, job scheduling, topology and routing configuration mechanisms for Hadoop jobs. Our analysis suggests that such an integrated control has great potential to improve application performance with relatively small configuration overhead. We believe our study shows early promise of achieving the long-term goal of tight network and application integration using SDN.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Software Defined Networking, Optical Circuit Switching, Big Data Applications

1. INTRODUCTION

Network engineers and researchers have long sought effective ways to make networks more "application-aware". A variety of methods for optimizing the network to improve application performance or availability have been considered. Some of these approaches have been edge-based, for example tuning protocol parameters at end-hosts to improve throughput [28], or choosing overlay nodes to direct traffic over application-optimized paths [23]. Examples of network-centric approaches include providing custom instances of routing protocols to applications [10], or even allowing applications to embed code in network devices to perform application processing [24].

HotSDN'12, August 13, 2012, Helsinki, Finland.

While these earlier efforts have met with varying degrees of success and adoption, the recent emergence of the software-defined networking paradigm has created renewed interest in tailoring the network to better meet the needs of applications. By providing a well-defined programming interface to the network (e.g., OpenFlow), SDN provides an opportunity for more dynamic and flexible interaction with the network. Despite this promising vision, the ability of SDN to effectively configure or optimize the network to improve application performance and availability is still nascent.

Recently, two concomitant trends in data center applications and network architecture present a new opportunity to leverage the capabilities of SDN for truly application-aware networking. The first is the growing prominence of big data applications which are used to extract value and insights efficiently from very large volumes of data [1, 2, 20, 29]. Many of these applications process data according to well-defined computation patterns, and also have a centralized management structure which makes it possible to leverage application-level information to optimize the network. The second trend is the growing number of proposals for data center network architectures that leverage optical switches to provide significantly increased point-to-point bandwidth with low cabling complexity and energy consumption [26, 18, 15, 25]. Some of this work has demonstrated how to collect network-level traffic data and intelligently allocate optical circuits between endpoints (e.g., top-of-rack switches) to improve application performance. But without a true application-level view of traffic demands and dependencies, circuit utilization and application performance can be poor [14].

These three trends taken together – software-defined networking, dynamically reconfigurable optical circuits, and structured big data applications – motivate us to explore the design of an SDN controller using a "cross-layer" approach that configures the network based on big data application dynamics at run-time. In this paper, we focus on Hadoop as an example to explore the design of an integrated network control plane, and describe Hadoop job scheduling strategies to accommodate dynamic network configuration. We introduce the topology construction and routing mechanisms for a number of communication patterns, including single aggregation, data shuffling, and partially overlapping aggregation traffic patterns to improve application performance.

The proposed topology configuration algorithms can be implemented using a small number of OpenFlow rules on each switch. Our preliminary estimation suggests that the flow rule installation introduces low overhead compared to the relatively long duration of MapReduce jobs. However, a number of challenges remain which have implications for SDN controller architectures. For example, in contrast to common SDN use cases such as WAN traffic engineering [12] and cloud network provisioning [8], run-time network configuration for big data jobs requires more rapid and frequent

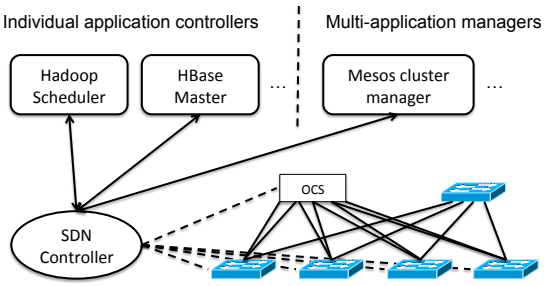

Figure 1: Integrated network control for big data applications

flow table updates. This imposes significant requirements on the scalability of the SDN controller and how fast it can update state across the network. Another challenge is in maintaining consistent network-wide routing updates with low latency, and in coordinating network reconfiguration requests from different applications in multi-tenancy environments.

This is a work-in-progress, and there is clearly more work to be done to realize and fully evaluate the design. Nevertheless, we believe that this work shows early promise for achieving one of the oft-cited goals of software-define networking, that is to tightly integrate applications with the network to improve performance and utilization.

2. INTEGRATED NETWORK CONTROL ARCHITECTURE

In this section, we describe the overall architecture of cross-layer network control for big data applications.

2.1 System Architecture

Figure 1 shows the system architecture of a cross-layer network control plane. We assume a hybrid electrical and optical data center network where OpenFlow-enabled top-of-rack (ToR) switches are connected to two aggregation networks, a multi-root tree with Ethernet switches and a MEMS-based optical circuit switch[1]. Each ToR switch has multiple optical uplinks connected to the optical switch (commodity switches typically have 4 to 6 uplinks). All the switches are controlled by a SDN controller which manages physical connectivity among ToR switches over optical circuits by configuring the optical switch. It can also manage the forwarding at ToR switches using OpenFlow rules.

Many big data applications, such as Hadoop [1], Dryad [20], Spark [29] and HBase [2], have a master node, or application controller, that manages all incoming job requests. To support cross-layer network control, the SDN controller is interfaced to the master node for each individual application, such as the Hadoop scheduler or HBase master. It could also connect to broader coordination frameworks such as Mesos [13] that manage multiple co-existing applications sharing the data center.

Since the SDN controller may be shared among multiple applications, it provides a general interface to configure devices and control forwarding in the network. It also provides a query interface to make authoritative network information available to applications. For big data applications, the SDN controller provides an interface that accepts traffic demand matrices from application controllers in a standard format. The traffic demand matrix describes the volume and policy requirements for traffic exchanged between different source and destination racks.

Using the network configuration interface, application controllers report traffic demands and structure from one or multiple jobs and issue a network configuration command to set up the topology accordingly. They can also use network information provided by the SDN controller, such as topology, to make more informed decisions on job scheduling and placement (further described in Section 3.2). Application controllers implement their own application-specific mechanisms to collect traffic demands from different components, and also define appropriate criteria to correlate and aggregate demands from multiple jobs.

2.2 Traffic Pattern of Big Data Applications

The traffic in big data applications consists of bulk transfer, data aggregation/partitioning, and latency sensitive control messages. The control traffic is typically low data rate and can be handled easily by even a relatively slow Ethernet. In our architecture, control messages are always sent over the packet switched network using default routes that direct traffic over the Ethernet[2].

Data aggregation and partitioning, where data are partitioned or aggregated between one server and a large number of other servers, is a common traffic pattern. For example, in MapReduce, the intermediate results from all the mappers will be aggregated at a reducer for performing the reduce function. The shuffle phase of MapReduce is actually a combination of multiple data aggregation patterns between mappers and reducers. In parallel database systems, most operations require merging and splitting data from different tables. Data aggregation requires high bandwidth to exchange large volumes of data between a potentially large number of servers. In the typical case of oversubscribed data center networks, the data aggregation and shuffling patterns can easily become performance bottlenecks. Our cross-layer network controller is designed primarily to address the challenge of handling a mix of multiple aggregation and data shuffling tasks.

2.3 The Advantage of Application Awareness

For big data applications, an application-aware network controller provides improved performance. By carefully allocating and scheduling high-bandwidth links via optical paths, job completion time can be reduced significantly. Data center operators also benefit from better utilization of the relatively limited set of high-bandwidth optical links.

Current approaches for allocating optical circuits in data centers, such as c-Through [26], Helios [18] and OSA [15], rely on network level statistics to estimate the traffic demand matrix in the data center. While these designs show the potential to benefit applications, recent work has shown that without a true application-level view of traffic demands and dependencies, circuit utilization and application performance can be poor [14]. First, it is difficult to estimate real application traffic demand based only on readings of network level statistics. Without accurate information about application demand, optical circuits may be configured between the wrong locations, or circuit flapping may occur from repeated corrections. Second, blindly optimizing circuit throughput without considering application structure could cause blocking among interdependent applications and poor application performance.

To give an example of these problems, and further motivate our approach, Figure 2 shows an aggregation pattern where data is aggregated from racks $R1$–$R8$ to rack $R0$. Each rack has 200 MB of data to send to $R0$. Rack $R0$ has 3 optical links with 10Gbps band-

[1]We assume MEMS-based optical switch for the simplicity of discussion. The integrated network control plane can also incorporate other optical circuit switching technologies with different ways to configure network topologies, e.g. wavelength selective switch used in OSA [15].

[2]This can be implemented using static default rules on ToR switches in addition to dynamic routes discussed later.

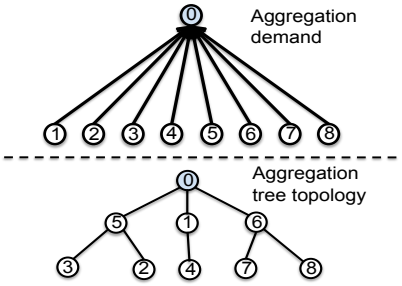

Figure 2: An Example of 8-to-1 Aggregation

width on each link. The aggregation flows are correlated where the application cannot proceed until all the data has been aggregated.

In single-hop circuit allocation systems, such as c-Through [26] and Helios [18], the aggregation can be finished in 3 rounds with circuits on rack $R0$ connecting to 3 other racks in each round. Assuming the minimum circuit reconfiguration interval is set to 1 second, i.e., circuits will be in place for a minimum of 1 second, the aggregation will be finished in 2.16 seconds ($1\text{s} + 1\text{s} + 200\text{MB}/10\text{Gbps}$), even ignoring the reconfiguration overhead of optical circuits (usually on the order of tens of ms). However, as discussed in [14], with competing background traffic, circuits might be allocated to other flows due to the limited view of network-level traffic demand estimation. In this case, the aggregation might take much longer to finish. For example, if the 200 MB traffic on rack $R5$ is mistakenly placed on the congested Ethernet network with, say, 100 Mbps of residual bandwidth, the completion time of the whole aggregation could be as high as 16 seconds.

With a cross-layer view, the SDN controller can collect related traffic demands from application tasks and configure the network for them in an application-aware way. As shown in Figure 2, the SDN controller can set up an aggregation tree topology among the 9 racks and complete the aggregation task in one round without reconfiguring the circuits (i.e., in just 480 ms, a 70% reduction). In this sense, application-aware network configuration based on correlated traffic has the potential to achieve significant performance improvements.

Note that recently proposed all-optical architectures like OSA [15] can create a data center wide multi-hop topology among racks using optical circuits. However, OSA has two important characteristics that make it fundamentally different from our proposal. Firstly, OSA is still application agnostic and the multi-hop topology is constructed based on network-level observations of traffic demand. We and others have already argued that application agnostic designs can lead to poor application performance [14]. Secondly, OSA requires all racks in the data center – potentially thousands of racks in total, with different subsets of racks running unrelated applications – form one connected network topology. This inflexibility coupled with the lack of application awareness makes it hard to optimize for a specific application running over a subset of racks. Consider our data aggregation example under OSA. Due to simultaneous traffic demands from many other tasks, rack $R1$, ..., $R8$ could be placed topologically far away from rack $R0$, resulting in significantly slower data aggregation.

3. NETWORK CONFIGURATION FOR HADOOP JOBS

In this section, we use Hadoop as an example to discuss the specific design of an integrated network control plane.

3.1 Traffic Demand Estimation of Hadoop Jobs

Hadoop uses a centralized architecture to manage jobs and data. Job submission and scheduling are controlled by a job tracker node. A name node manages the meta-data of all the data blocks on Hadoop distributed file system. The centralized architecture makes it easier to collect job and data related information for Hadoop applications. A demand estimation engine can be implemented on the job tracker to collect the traffic demand matrix for selected jobs and request network configuration for them.

Hadoop job tracker has accurate information about the placement of map and reduce tasks. It also knows which map and reduce tasks belong to the same MapReduce job. So given a set of jobs, the demand estimation engine can compute the source and destination racks of their traffic demand easily by looking at the placement of tasks. However, estimating traffic volume among these tasks requires more detailed analysis. When the job is submitted, the job tracker can compute the input split size for each map task. But the volume of shuffling traffic between a map and a reduce task is decided by the size of intermediate data generated by the mapper, which is not available until the map task is finished. So the job tracker has to monitor the progress of all the tasks and updates the observed traffic volume among them at run-time. By correlating traffic volume observed on different tasks, it can predict the shuffling traffic demand of map tasks before they are finished. For example, we can predict the shuffling traffic demand of a map task using traffic volume observed on previous similar map tasks. Since all the map tasks belong to the same job are running the same map function, they would generate similar amount of intermediate data with the same input split size.

3.2 Network-aware Job Scheduling

The default scheduling discipline in Hadoop is FIFO, where jobs are submitted to a queue and the job tracker simply schedules them one by one in the order of arrival. Hadoop uses different strategies to place map tasks and reduce tasks. Data locality is considered in the placement of map tasks so that the input data of most map tasks is loaded from local disks. Reduce tasks are placed randomly without considering data locality in shuffling phase.

Hadoop job scheduling is still an active research topic, with recent studies showing that task scheduling has significant implications on the performance of Hadoop jobs [30, 7]. With knowledge of network-layer information, Hadoop can make more informed decisions about job placement and scheduling. Although job scheduling is not the focus of this paper, we discuss a few simple strategies to leverage the integrated network control plane in the job placement to make the network configuration easier.

Bin-packing placement: We use rack-based bin-packing placement for reduce tasks to aggregate them onto a minimum number of racks. Since the SDN controller configures topology and routing at the rack level, rack-based bin-packing placement can create opportunities to aggregate traffic on these racks and reduce the number of ToR switches that need to be configured.

Batch processing: The network configuration of map and reduce tasks should be handled in batches, where a group of tasks submitted in a period T will be processed together. Within a batch of tasks, the job tracker selects those with greatest estimated volume and requests the SDN controller to set up the network for them. Tasks in earlier batches have higher priority such that tasks in later batches can be allocated optical circuits only if there are left over circuits from earlier batches. There are two benefits of doing batch processing. First, it helps aggregate traffic from multiple jobs to create long duration traffic that is suitable for circuit switched paths. Second, it ensures that an earlier task does not become starved by later tasks. The batch processing can be implemented as a simple extension to Hadoop job scheduling.

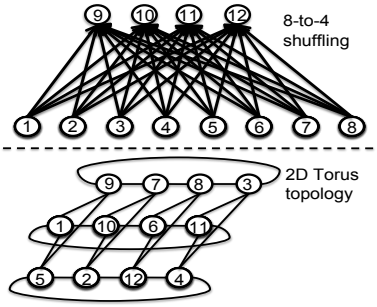

Figure 3: 8-to-4 Shuffling Using Torus Topology

3.3 Topology and Routing for Aggregation Patterns

A major challenge in configuring the network for Hadoop tasks is to handle aggregation traffic among mappers and reducers[3]. Each reducer will collect intermediate results from a set of mappers. With a batch of map and reduce tasks, the traffic among them is a mix of multiple aggregations. Depending on where mappers and reducers are placed, these aggregations could end up as a single aggregation, a many-to-many shuffling pattern, or a partially overlapping aggregation pattern across multiple racks. Below we will discuss how to configure the network for single aggregation and shuffling patterns, and then how to use these configurations as building blocks to handle multiple potentially overlapping aggregations.

Single aggregation pattern: We start from the single aggregation pattern, where reducers on one rack need to collect data from mappers on other racks. Using multiple optical circuits available on each rack, we can construct a tree topology among all the racks to support the aggregation traffic in an application-aware network configuration. As shown in earlier in Figure 2, using 3 optical uplinks, we can construct a 2-hop aggregation tree to finish an 8-to-1 aggregation pattern without reconfiguring the network. All the racks can send to the aggregator rack through multi-hop optical paths, and we can use OpenFlow flow rules on the ToR switches to setup the routes.

Neighbor selection and routing: Two issues need to be addressed when constructing these multi-hop topologies, neighbor selection and routing. To reduce the traffic sending over multi-hop optical paths, we want to place racks with higher traffic demand closer to the aggregator in the tree. One simple strategy to achieve this is to rank all the racks based on the traffic demand, and add racks into the tree topology in the order of high demand to low demand. The aggregation traffic can be easily routed along the tree towards the aggregator target.

Data shuffling pattern: When reduce tasks are placed on multiple racks, the traffic among reducers and mappers creates the common cross-rack data shuffling pattern in many MapReduce jobs. In general, we can describe a $N - to - M$ data shuffling as follows: a $N - to - M$ data shuffling is a traffic pattern from source set S to destination set D, $|S| = N, |D| = M$, where data will be aggregated from all the nodes in S to each of the nodes in D. When $S = D$, we have an all-to-all shuffling pattern.

We can build more densely connected topologies using optical links to support shuffling traffic. Recently proposed server-based data center network architectures, such as BCube [19] and CamCube [5], leverage Hypercube and Torus topologies originally developed in the HPC community to build network with high path redundancy. These topologies are designed with balanced structures to avoid hotspots or single point bottlenecks, which makes them suitable to exploit multi-pathing for shuffling traffic. In our case, we can build a Hypercube or Torus-like topology among racks to support data shuffling. For example, Figure 3 shows a 2-D Torus topology we can construct to support a 8-to-4 rack data shuffling using 4 optical uplinks on each rack.

Neighbor selection and routing: The construction and routing of Torus topology is more complicated. We want to place racks with high traffic demand close to each other to reduce the amount of data over multi-hop paths. Finding an optimal Torus topology for a shuffling pattern is difficult due to the large search space in the topology construction. However, a greedy heuristic algorithm can be used to construct the Torus network. Building a 2-D Torus topology essentially places racks into a 2-D coordinate space and connects each row and each column into rings. Given an $N - to - M$ shuffling pattern with R unique racks involved, we can build a $X \times Y$ Torus topology with X racks in each row and Y racks in each column, where $X = \lceil(\sqrt{R})$ and $Y = \lceil(\frac{R}{X})$. The Torus network is constructed as follows:

1. Find four neighbors for each rack based on the traffic demand and rank all the racks based on the overall traffic demand to its neighbors.

2. Start constructing the Torus from the highest ranked rack S. Connect two rings around rack S with X and Y racks in the rings respectively. Similarly, racks with higher traffic demand to rack S will be placed closer to S in the ring. These two rings will be the "framework" for the Torus topology, which map to coordinates $(0,0), ..., (0, X-1)$ and $(0,0), ..., (Y-1, 0)$ in the 2-D Torus space.

3. Select racks for row 2 to row Y one by one based on the coordinates. Given a coordinate $\{(x,y), x > 0, y > 0\}$, select the rack with the highest overall demand to neighboring racks with coordinates $\{(x-1,y), (x,y-1), ((x+1) \bmod X, y), (x, (y+1) \bmod Y)\}$. If a neighbor rack has not been placed, the demand will be ignored.

Previous work has proposed different routing schemes over a Torus topology to explore multiple paths in the topology [5]. A routing scheme well-suited for data shuffling traffic is per-destination rack spanning tree, which is similar to the source-based spanning tree scheme for multicast traffic. The idea is to build a spanning tree rooted at each aggregator rack. Traffic sent to each aggregator will be routed over its spanning tree. When an optical link is selected for a spanning tree, we can increase its link weight to favor other links for other spanning trees. By doing that we can exploit all the available optical links in the Torus topology to achieve better load balancing and multi-pathing among multiple spanning trees. In our ongoing work, we plan to evaluate the performance implications of this routing scheme on Hadoop performance. In addition to the routing scheme, recent studies have also explored in-network partial aggregation on server-based Torus topologies [5, 11]. A similar in-network partial aggregation service can also be explored on the rack level using a Torus topology constructed with optical circuits.

Partially overlapping aggregations: Data shuffling can be treated as multiple aggregations sharing the same sources, which is a special case of overlapping aggregation patterns. In the general case, aggregation patterns may have partially overlapping sources and aggregators. For these patterns, the traffic demand among racks could be sparse. If we build a big Torus network among these racks, many optical links may not be highly utilized. We discuss how to

[3]Traffic due to data loading and result writing is mostly point to point bulk transfer, which can be handled relatively easily with point-to-point optical circuits.

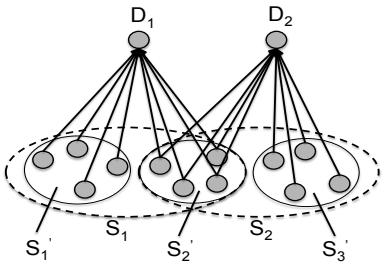

Figure 4: Scheduling two partial-overlapping aggregations

divide general aggregation patterns into multiple single aggregation and shuffling patterns and schedule circuits among them.

As shown in Figure 4, given two aggregation patterns $T_1 = \{S_1 \rightarrow D_1\}$ and $T_2 = \{S_2 \rightarrow D_2\}$ where S_i is the set of source racks and D_i is the aggregator racks. We have: $S_1' = S_1 \setminus S_2$, $S_2' = S_1 \cap S_2$, $S_3' = S_2 \setminus S_1$. T_1 and T_2 can be divided into four sub-patterns: $T_1' = \{S_1' \rightarrow D_1\}$, $T_2' = \{S_2' \rightarrow D_1\}$, $T_3' = \{S_2' \rightarrow D_2\}$, $T_4' = \{S_3' \rightarrow D_2\}$. Among the four patterns, T_1' and T_4' have different source sets and different aggregators. They can be scheduled as two independent aggregations. T_2' and T_3' have the same source set and different aggregators, which is essentially a $N - to - 2$ shuffling pattern. Therefore, we can divide general aggregation tasks T_1 and T_2 into two independent aggregations $\{T_1', T_4'\}$ and one shuffling pattern $\{T_2', T_3'\}$.

We can use the configuration algorithms discussed before to schedule the network for each pattern. Depending on the number of available optical links on aggregator racks, the two groups of sub-patterns can be scheduled concurrently or one after another. Using this mechanism, we can iteratively organize multiple general aggregation patterns into multiple groups of independent aggregations and shuffling patterns and scheduling them accordingly. The re-organization of aggregation patterns allows us to explore path sharing among them and improve the utilization of optical circuits.

4. IMPLEMENTATION AND OVERHEADS

To implement the topology construction and routing schemes discussed above, we need to install OpenFlow rules on ToR switches and issue commands to reconfigure optical switches. Commercially available MEMS switches take less than 10 ms to set up a new configuration [26, 18, 15]. However, run-time routing configuration for big data jobs over a dynamic network topology requires rapid and frequent flow table updates over a potentially large number of switches. With high arrival rate of jobs, the routing configuration has to be done within a short period of time. This imposes significant challenges on the SDN controller in terms of its scalability and how fast it can update network-wide routing state.

We want to use as few rules as possible to implement the routing configuration, not only because of limited flow table size on switches, but also in order to reduce delays in reconfiguring the network. Since the routing setup happens at rack level, one way to reduce the required number of rules is to tag packets based on their destination rack. We can use the VLAN field to tag the packets, with each rack assigned to one VLAN id[4]. Packets sent to a destination rack will be tagged with the same VLAN id. We can then set up forwarding rules at ToR switches based on the VLAN id in packets. Packet tagging can be implemented in the server's kernel as in c-Through [26] or using OpenFlow rules on a hypervisor virtual switch like Open vSwitch [4]. Servers can look up the VLAN tag in a centralized repository based on the destination address of packets. Packet tagging is set up on demand when the traffic is first sent to a destination server, so it is not necessary to install all the rules when the network is reconfigured for dynamic tasks.

To realize the per-destination rack spanning tree, we need one flow rule for each destination rack. For a topology with N racks, we need at most N rules on each ToR switch to realize the routing configuration. Recent analysis of large production data center traces [17, 21] shows that most MapReduce jobs last for tens of seconds or longer, and many data intensive jobs run for hours. The largest jobs run on a few hundred servers[5]. Commodity OpenFlow switches can install a flow rule in a few ms. For example, commercially available 10Gbps OpenFlow switches can install more than 700 new rules in a second depending on the load on the switch and how many rules are batched together. For a large Hadoop job running on 20 racks, it takes at most tens of ms to install 20 forwarding rules on each ToR switch. Given the duration and size of typical MapReduce jobs in production data centers, we believe the latency to install rules is relatively small. Also note that the network configuration can overlap with task startup process.

Nevertheless, challenges remain in addressing the consistency issue when we reconfigure the physical connectivity and refresh all the routing state on tens of switches simultaneously. We will have to control the timing and order of topology reconfiguration and route state updates on different switches to avoid potential transient errors and forwarding loops. Recent work [22] has proposed solutions for the consistent update issues in the SDN context, but the solution requires a significant number of extra rules on each switches. It remains to be seen how much extra delay this approach adds to achieve state consistency during the topology and routing updates required by our system. In our future work, we plan to investigate schemes to ensure network-wide consistent updates while adhering to the low latency requirement.

5. DISCUSSION AND FUTURE WORK

Our discussion has been focused on configuring physical topology and routing for big data applications. Several other issues remain to be explored in future work to realize the full capability of the integrated network control.

Fairness, priority and fault tolerance: On the fault tolerance aspect, the SDN controllers, such as Floodlight [3], have built in mechanisms to handle network device and link failures. Failed devices and links will be updated in the topology by the SDN controller. Most big data applications have also been designed to handle failures. For example, Hadoop job tracker monitors the progression of all the jobs and failed tasks will be rescheduled onto different servers. Therefore, in the integrated system, the failure handling mechanisms can remain untouched with application managers and the SDN controller handling failures at different levels.

Since network configuration is performed over batches of application tasks, task-level fairness and priority scheduling within an application can be done by the application job scheduler. However, the data center network infrastructure is normally shared by multiple applications. Application managers could request network configuration concurrently for their tasks. The fairness and priority among different application requests must be handled by the SDN controller. The scheduling policy of SDN controller for different applications is an open question to be explored.

Traffic engineering for big data applications: Using OpenFlow protocol, SDN controller can perform flow-level traffic engineering for big data applications. There are multiple routes on the rack level going through different optical paths and electrical

[4]MPLS labels can also be used with support from the new version of OpenFlow protocol v 1.2.

[5]Assuming 20-40 servers per racks, it amounts to roughly tens of racks.

paths. Accurate traffic demand and structural pattern from applications can allow SDN controller to split or re-route management and data flows on different routes (as discussed in Hedera [6], MicroTE [9]). Although it remains to be seen how effectively the flow-level traffic engineering can optimize the performance of big data applications, these schemes could be useful if optical switches are not available in production data centers. Implementing flow level traffic engineering requires installing a rule for each selected flow on ToR switches, which imposes additional overhead to the network configuration. In future work, we will explore flow level traffic engineering mechanisms for big data applications and the efficient implementation of them using SDN controller.

6. RELATED WORK

In addition to aforementioned work using optical switches to reconfigure the data center network topology, Schares et al. have discussed the use of optical switches for stream processing system [16]. Several recent studies explore the use of OpenFlow to adjust the routing for different applications. In [12], Das et al. propose to use OpenFlow to aggregation traffic for different services over dynamic links on converged packet-circuit network. This work is focused on wide-area network services. Topology switching [27] is a recent proposal to isolate applications and use different routing mechanisms for them in fat-tree based data centers. Our work explores more tight integration between applications and network control. We focus on the run-time network reconfiguration for big data applications and the dynamic interaction between application components and the SDN controller in data centers. We study the programmability on every layer of network from physical topology to routing and flow level traffic engineering.

7. CONCLUSION

In this paper, we explore an integrated network control architecture to program the network at run-time for big data applications using optical circuits with an SDN controller. Using Hadoop as an example, we discuss the integrated network control architecture, job scheduling, topology and routing configuration for Hadoop jobs. Our preliminary analysis suggests the great promise of integrated network control for Hadoop with relatively small configuration overhead. Although our discussion has been focused on Hadoop, the integrated control architecture can be applied to any big data applications with a centralized or logically centralized master. Since data aggregation is common in big data applications, the network configuration for aggregation patterns can be generally applied to other applications too. We believe our study serves as a step towards tight and dynamic interaction between applications and network using SDN.

8. REFERENCES

[1] Apache Hadoop, http://hadoop.apache.org.

[2] Apache HBase, http://hbase.apache.org.

[3] Floodlight openflow controller. http://floodlight.openflowhub.org/.

[4] Open vswitch. http://openvswitch.org/.

[5] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly. Symbiotic routing in future data centers. In *SIGCOMM'10*, August 2010.

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

[7] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *USENIX OSDI'10*, December 2010.

[8] T. Benson, A. Akella, A. Shaikh, and S. Sahu. Cloudnaas: A cloud networking platform for enterprise applications. In *ACM SOCC'11*, October 2011.

[9] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: The case for fine-grained traffic engineering in data centers. In *ACM CoNEXT'11*, December 2011.

[10] P. Chandra, A. Fisher, C. Kosak, T. S. E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Resource management for value-added customizable network service. In *IEEE ICNP'98*, October 1998.

[11] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea. Camdoop: Exploiting in-network aggregation for big data applications. In *USENIX NSDI'12*, April 2012.

[12] S. Das, Y. Yiakoumis, G. Parulkar, P. Singh, D. Getachew, P. D. Desai, and N. McKeown. Application-aware aggregation and traffic engineering in a converged packet-circuit network. In *OFC'11*, March 2011.

[13] B. Hindman et al. Mesos: A platform for fine-grained resource sharing in the data center. In *USENIX NSDI'11*, March 2011.

[14] H. Bazzaz et al. Switching the optial divide: Fundamental challenges for hybrid electrical/optical data center networks. In *ACM SOCC'11*, October 2011.

[15] K. Chen et al. OSA: An optical switching architecture for data center networks with unprecedented flexibility. In *NSDI'12*, April 2012.

[16] L. Schares et al. A reconfigurable interconnect fabric with optical circuit switch and software optimizer for stream computing systems. In *OFC'09*, March 2009.

[17] Y. Chen et al. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *ACM EuroSys'12*, April 2012.

[18] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *ACM SIGCOMM*, August 2010.

[19] 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. In *ACM SIGCOMM'09*, August 2009.

[20] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *ACM EurySys'07*, March 2007.

[21] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production mapreduce cluster. In *CMU Technical Report*, December 2009.

[22] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *ACM SIGCOMM'12*, August 2012.

[23] J. Seedorf and E. Burger. Application-layer traffic optimization (alto) problem statement. In *RFC-5693*, 2009.

[24] D. L. Tennenhouse, J. M. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. In *IEEE Communications Magazine*, January 1997.

[25] A. Vahdat, H. Liu, X. Zhao, and C. Johnson. The emerging optical data center. In *OFC'11*, March 2011.

[26] G. Wang, D. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan. c-Through: Part-time optics in data centers. In *ACM SIGCOMM*, August 2010.

[27] K. Webb, A. Snoeren, and K. Yocum. Topology switching for data center networks. In *USENIX Hot-ICE'11*, March 2011.

[28] E. Weigle and W. Feng. A comparison of tcp automatic tuning techniques for distributed computing. In *IEEE HPCS'02*, July 2002.

[29] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *USENIX HotCloud'10*, June 2010.

[30] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *USENIX OSDI'08*, December 2008.