Network Configuration and Flow Scheduling for Big Data Applications

Lautaro Dolberg, Jérôme François, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, and Thomas Engel

CONTENTS

INTRODUCTION

Big Data applications play a crucial role in our evolving society. They represent a large proportion of the usage of the cloud [1–3] because the latter offers distributed and online storage and elastic computing services. Indeed, Big Data applications require to scale computing and storage requirements on the fly. With the recent improvements of virtual computing, data centers can thus offer a virtualized infrastructure in order to fit custom requirements. This flexibility has been a decisive enabler for the Big Data application success of the recent years. As an example, many Big Data applications rely, directly or indirectly, on Apache Hadoop [11] which is the most popular implementation of the MapReduce programming

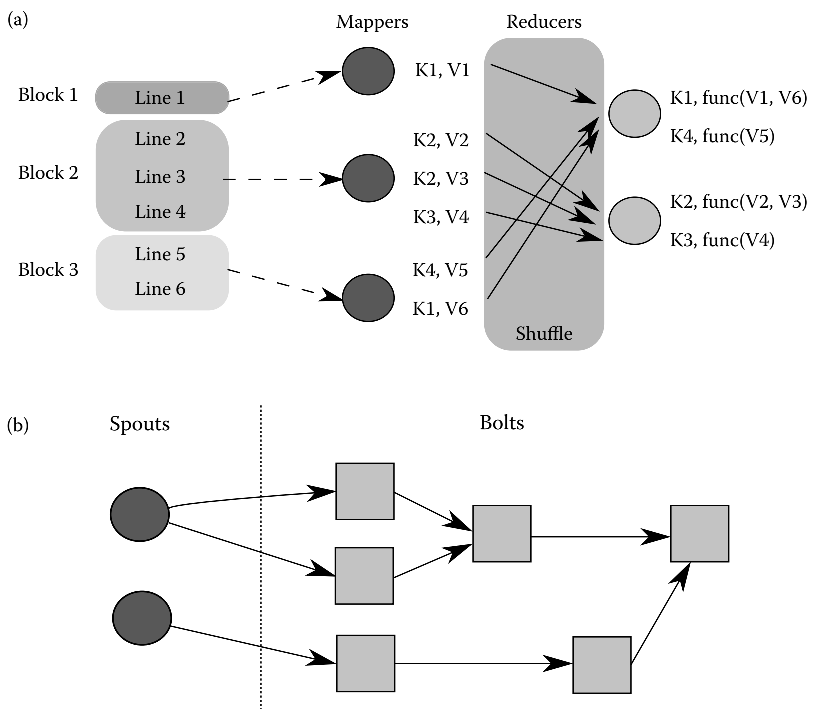

FIGURE 7.1 Big Data computational model and the underlying network traffic as plain arrows: (a) MapReduce with K as key and V as value; (b) Storm.

model [4]. From a general perspective, it consists in distributing computing tasks between mappers and reducers. Mappers produce intermediate results which are aggregated in a second stage by reducers. This process is illustrated in Figure 7.1a, where the mappers send partial results (values) to specific reducers based on some keys. The reducers are then in charge of applying a function (like sum, average, or other aggregation function) to the whole set of values corresponding to a single key. This architectural pattern is fault tolerant and scalable. Another interesting feature of this paradigm is the execution environment of the code. In Hadoop, the code is directly executed near the data it operates on, in order to limit the data transfer within the cluster. However, large chunks of data are still transferred between the mappers and reducers (shuffle phase) which thus necessitate an efficient underlying network infrastructure. It is important to note that the shuffle phase does not wait for the completion of the mappers to start as the latter already emits (key, value) pairs based on partial data it has read from the source (e.g., for each line). Since some failures or bottlenecks can occur, Hadoop tasks are constantly monitored. If one of the components (i.e., mappers or reducers) is not functioning well (i.e., it does not progress as fast as others for example), it can be duplicated into another node for balancing load. In such a case, this leads also to additional data transfers.

Storm [5] is another approach that aims at streaming data analytics, while Hadoop was originally designed for batch processing. Storm consists of spouts and bolts. Spouts read a data source to generate tuples and emit them toward bolts. Bolts are responsible for processing the tuples and eventually emit new tuples toward other bolts. Therefore, a Storm application is generally represented by a graph as shown in Figure 7.1b. The main difference between Storm and MapReduce is that data transfers occur all the time (streaming) and so are not limited to a specific phase (shuffle phase in Hadoop). As a result, among the diversity in Big Data applications, there are common problems, in particular optimizing the data transfer rate between host.

Therefore, while Big Data technological improvements were mainly highlighted by new computing design and approaches, like Hadoop, network optimizations are primordial to guarantee high performances. This chapter reviews existing approaches to configure network and schedule flows in such a context. In the following sections, we will cover the diverse optimization methods grouped according to their intrinsic features and their contributions. In particular, recent network technologies such as Software-Defined Networking (SDN) empowered the programmability of switching devices. Consequently, more complex network scheduling algorithms can be afforded to leverage the performance of MapReduce jobs. That is why this chapter focuses on SDN-based solutions but also introduces common networking approaches which could be applied as well as virtualization techniques. The latter are strongly coupled with the network design. For example, end-hosts in a data center are virtual machines (VMs) which can be assigned to different tasks and so would lead to various traffic types, which can be better handled if the network is adaptive and therefore easily reconfigurable.

This chapter is structured as follows:

1. *Optimization of the VM placement*: even if not dealing with network configuration, it has a significant impact on the same;
2. *Topology design*: it is an important topic as the way the machines are wired have an impact on performance;
3. *Conventional networking, in particular routing and Quality of Service (QoS) scheduling*: these might be customized to support Big Data as well.
4. *SDN*: this highlights recent approaches that leverage a global view of the network to implement efficient traffic management policies.

VM PLACEMENT FOR REDUCING ELEPHANT FLOW IMPACT

Very large flows, normally associated to long MapReduce jobs, are often called Elephant Flows [6]. Since any VM can be potentially hosted in any physical server, grouping VM that are involved in large data transfers can reduce the impact on the overall bandwidth usage of the network. This approach is based on the internal routing of Hypervisor systems used in virtualized data centers such as XEN [7], KVM [8], or VMWare [9] solutions. From a more general point of view, VMs can be colocated in a certain region of a network, even if on different physical machines. This is illustrated in Figure 7.2 where in the case of the original allocations (Figure 7.2a), the tasks of the same job are scattered in the network and so the traffic between them has to go through many hops eventually resulting in network

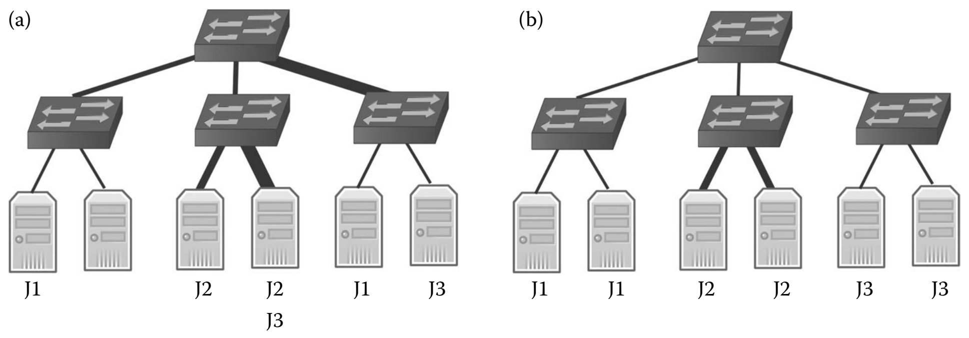

FIGURE 7.2 VM placement with three jobs (each job JI has two tasks). The width of a link represents its load. (a) Initial task allocation, and (b) optimized task allocation.

congestion. In Figure 7.2b, by moving only two tasks (one from J1 and one from J3), each job is isolated in a single rack (under a single switch) and so no congestion occurs at higher level switches while improving the data transfer efficiency between tasks of the same job since these are connected through a single switch.

VM placement is basically related to VM allocation problems, which are optimization problems under certain criteria. One of the criterion should be the usage of network resources. Because this is not the focus of this chapter, we recommend the reader to read Reference 10 for more details about network-aware VM placement.

The downside of existing network-aware VM placement approaches is that they lack the reactiveness. Normally, given the nature of MapReduce phases, it is not possible to exactly match in advance MapReduce jobs and needed network resources (e.g., how large the data transfer will be during the shuffle phase depends on the underlying data and applications). To cope with this practical issue, virtualized data centers may estimate the VM-to-VM traffic matrix but such a method works well with a known batch job only. Another solution is to migrate VMs during their execution, but this might be also resource consuming and negatively impact the finishing time of the Big Data jobs if this occurs too frequently.

TOPOLOGY DESIGN

Data-centers networks are usually organized in a tree topology [11,12] with three defined layers:

- *Core layer*: This layer is the backbone of the network where high-end switches and fibers are deployed. In this layer only L2 forwarding takes place without any packet manipulation. The equipment for this layer is the more expensive among the hierarchical network model.

- *Aggregation or distribution layer*: In this layer most of the L3 routing takes place.

- *Access layer*: This layer provides connectivity to the end nodes and so are located at the top of the racks. They perform the last step of L3 packet routing and packet manipulation. Normally, these are the cheapest devices in the hierarchical network model.

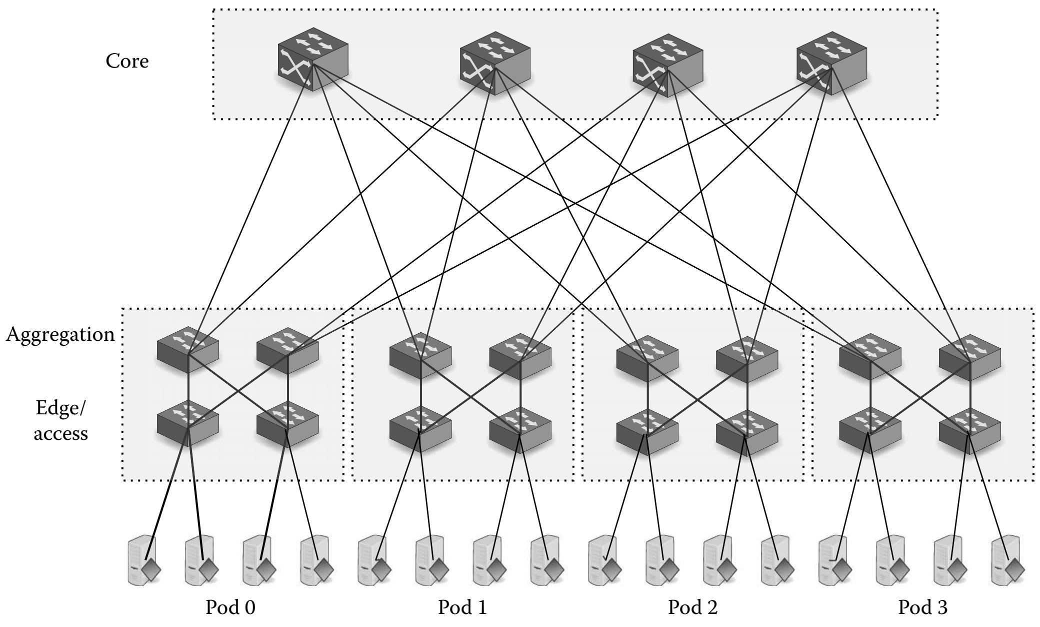

FIGURE 7.3 Example of a hierarchical network model: Multirooted network topology.

Thanks to this hierarchical model, a low latency is achieved for traffic between two nodes in the same rack. This explains why approaches like Hadoop leverage rack awareness to ensure fast replication of data by selecting nodes in the same rack for copying data (but also others out of the rack in order to guarantee data availability under a rack failure). In addition, this type of configuration supports a large number of ports at the access layer. A specific instance of the hierarchical model is the fat tree proposed in Al-Fares et al. [3] and illustrated in Figure 7.3, which enables fault-tolerance by ensuring redundant paths in a deterministic manner. The fat-tree or Clos topology was introduced more than 25 years ago [13] to reduce the cost of telephony-switched networks. The topology layout is organized as k-ary trees, where in every branch of the tree there are k switches, grouped in pods. Actually, a pod consists in $(k/2)^2$ end-hosts and $k/2$ switches. At the edge level, switches must have at least k ports connected as follows: half of the ports are assigned to end nodes and the other half is connected to the upper aggregation layer of switches. In total, the topology supports $(k^2/2)$ k-port switches for connecting host nodes.

DCell [14] is a recursively interconnected architecture proposed by Microsoft. Compared to a fat-tree topology, DCell is a fully interconnected graph in order to be largely fault tolerant even under several link failures. In fact, high-level DCell nodes are recursively connected to low level ones, implemented with mini switches to scale out as shown in Figure 7.4.

Experimental results have showed that a 20 nodes network can twice outperform a large data center used for MapReduce. As a downside, DCell requires a full degree of connectivity, making it in practice costly to maintain and deploy. To enhance network connectivity between servers, CamCube [15] is a torus topology where each server is interconnected to

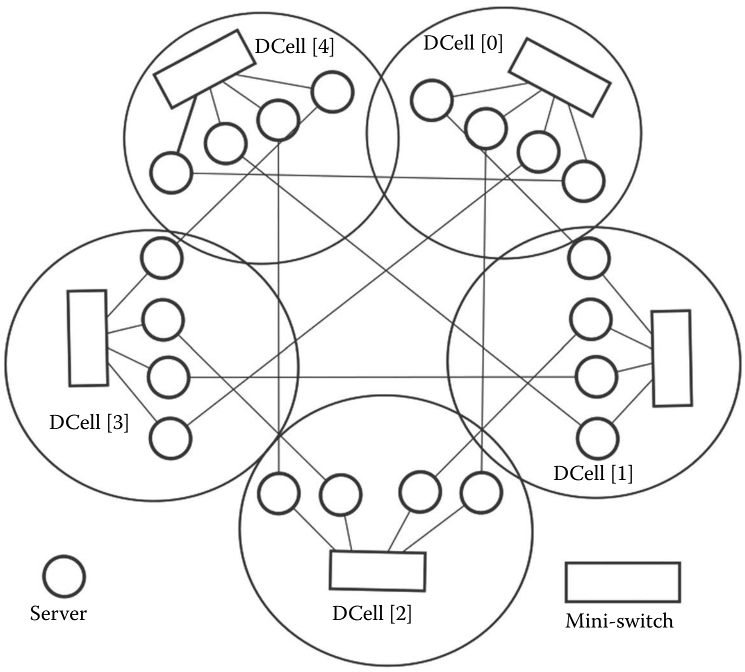

FIGURE 7.4 A DCell topology for five cells of level 0, each containing four servers. (From Guo, C. et al., in *Conference on Data Communication*, SIGCOMM, ACM, 2008.)

other six servers and all communications go through them, without any switch for internal communication. Finally, recent propositions like Singla et al. [16] promote a high flexibility by alleviating the need for a well-defined fixed graph structure, as the fat trees are, and do so by introducing some randomness in the topology bounded by some criteria.

CONVENTIONAL NETWORKING

Routing

Data-center network topologies like fat trees imply a large number of links leading to redundant paths. Therefore, routing algorithms can take that benefit to achieve a higher bandwidth. As an illustrative example in Figure 7.5a, the shortest path is used to route the traffic between the two tasks of the job J1. Unfortunately, it goes through a congested link. Hence, a redundant path can be used (Figure 7.5b) and even multiple of them conjointly (Figure 7.5a). Although these approaches have been proposed for routing in general, they are also used in data-centers to improve the performance of the Big Data applications. This is the reason why this section covers some propositions about how to use these principles in case of Big Data. However, the general issues are (1) to predict the traffic patterns and (2) to be able to rapidly change the configuration of the routing when the traffic suddenly changes, which is the case in a cloud infrastructure.

Nowadays, a major representative of such an approach is the equal cost multipath (ECMP) algorithm [17]. ECMP leverages the opportunity to route flows among multiple paths. Unlike traditional routing algorithms like OSPF which consider a single best path,

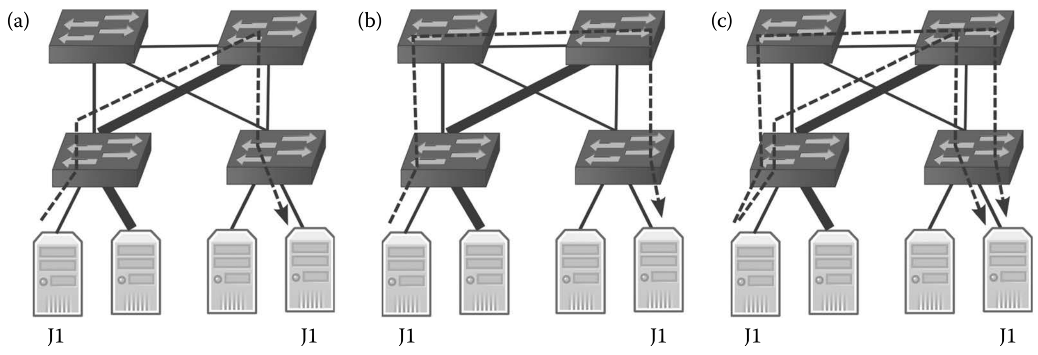

FIGURE 7.5 Routing decisions for one job with two tasks. The width of a link represents its load. (a) Shortest path routing, (b) high throughput, and (c) multipath routing.

ECMP considers all the best multipaths according to any metric (as, e.g., the number of hops) among which a single one is selected for a given flow through a load balancer. The number of multiple paths is dependent on the router implementation but is usually bounded to 16. Hence, this may yield a lower performance than expected for large data-centers. In fact, the amount of entries in the routing tables grows at an exponential rate, increasing the latency of the routing algorithm. Commercial solutions promoting multipath routing include FabricPath by Cisco Systems, BCube, VL2, and Oracle Sun data-center InfiniBand.

In addition to promoting the fat-tree topology usage for data-centers, Al-Fares et al. [3] proposed a dedicated routing algorithm based on an approach called Two-Level Routing Tables, where the routing tables are split into two hierarchical tables linked on the prefix length of the network address. A two layer table approach aims at leveraging the routing algorithm speed for establishing a route. This is possible because the authors introduced a private addressing system respecting a pre-established pattern like 8.pod.switch.host assuming a class A network. The first table index entries use a left-handed prefix length (e.g., 8.1.2.0/24, 8.1.1.0/24). The entries of the first table are linked to a smaller secondary table indexed by a right-handed suffix (e.g., 0.0.0.1/4, 0.0.0.4/4). For example, to find the route to the address 8.8.8.8, the algorithm will look up the first table, find the corresponding entry for the first part of the network address 8.8.8.0/24, then jumps to the secondary table and finds the remainder of the route. Since each switch of the aggregation layer in a fat-tree topology has always a $k/2°$ of connectivity to the access layer, Two-Level Routing Tables are bounded in the worst case to $k/2$ entries for suffixes and prefixes. Moreover, flows can be actually classified by duration and size. Then, the proposed algorithm in Al-Fares et al. [3] minimizes the overlap between the paths of voluminous flows. To achieve this, a central scheduler is in charge of keeping track of used links in the network in order to assign a new flow to a nonused path. From this perspective, it falls into the category of centralized networking (see section "Software-Defined Networks"), where a server acts as the controller by informing other ones about the link to use to forward specific packets of a flow.

The flow establishment is also leveraged by the previously described route lookup. In this approach, instead of routing traffic at a packet level, streams of data are grouped into flows and routed as a whole entity. One of the benefits of this approach is a faster route computation as it is reduced in a similar fashion as in circuit switching legacy technology. For example, if a host node requires to transfer a large data file as a part of a Big Data job, the whole stream will follow a pre-established route, reducing the latency of establishing a different route for each packet of the stream.

In order to enhance routing and network speed, hardware plays a core role. Therefore, there have been propositions to replace standard hardware. In particular, Farrington et al. [18] argue for a hybrid optical–electric switch as optical links achieve higher throughput but are not well adapted to bursty traffic. Combining both technologies thus helps in obtaining a good trade-off between accuracy and cost. Moreover, the technological availability of programmable circuits also leads to the possibility of implementing switching devices, especially in the aggregation and core layer using ASIC and FPGA devices. Lu et al. [19] propose an approach for implementing switching cards with a PCI-E interface. A recent proposal [20] addresses dynamic routing by replacing the traditional dynamic host configuration protocol (DHCP) address configuration by an another automated address configuration system. In this approach, the network is automatically blue printed as a graph. Then, by interpreting a set of labels assigned to each computing node, the system tries to find an isomorphism that minimizes the traffic at the aggregation layer. From the preliminary results, this approach has yielded promising results. However, it actually runs only over BCube or DCell because they have a fully connected topology.

Flow Scheduling

Network operators perform various traffic engineering operations in order to provide different network services on a shared network. This consists in classifying the traffic according to the intrinsic characteristics of each service or application using the network. For example, it is possible to define policies to specially treat Big Data applications. Similarly, the IPv6 Traffic Class includes the possibility of injecting information specific to applications in the packet stream. Other types of support for enabling network infrastructure to perform management of traffic are proposed in request for comments (RFCs) [21] and [22]. The first (DiffServ) proposes a protocol for differentiating services and its network behavior. The latter, Resource Reservation protocol (RSVP), specifies also a protocol that enables applications to reserve network resources in advance of initiating a data transfer.

As highlighted in the introduction, Big Data applications include both batch processing and streaming analytics, which are different by nature. In particular, batch processing jobs are more prone to use the network heavily during certain phases while streaming uses the network constantly with various rates. Therefore, the apparition of a batch job (Hadoop) may suddenly impact the network and so the other underlying applications. Dogar et al. [23] have proposed to schedule flows from BigData applications in a data center using a variation of first-in first-out (FIFO) scheduling that allows some level of multiplexing between the flows. The authors propose to schedule flows in the order of arrival with a certain degree of freedom and allow multiplexing over a limited number of flows which in turn allows small flows to be processed alongside large flows. This approach allows the co-execution of batch and streaming Big Data applications.

Limitations

It is worth mentioning that, in traditional data center networks, only aggregation and core layer switches have the capability of scheduling flows. This is a limitation imposed by the hardware. To be able to exploit the full potential of flow scheduling, an additional network function is required. This is often implemented in a central controller, this way allowing core and aggregation switches to be replaced by simple switches. One of the main advantages of using this approach is the reduced cost of switching and forwarding (L2) devices.

Another disadvantage of traditional networking is that the network configuration remains static and so impacts on the maintenance cost of the infrastructure because any modification of the topology must be wired manually by the network administrators. Virtualized networks come into play for coping with the lack of flexibility in traditional networks, and have become popular over the last years, thanks to the emerging virtualization technologies and computing power to support them. As a result, data-center owners offer their clients not only VMs (known as Virtual Private Servers [VPS]) but also virtual network infrastructure. This allows VPS users to create customized topologies. Virtual LANs (VLAN) have been popular in the past decades for splitting large organizational networks into smaller ones. However, this approach fails to segregate application traffic because of the coarse routing granularity inside a VLAN. A possible solution to this issue is to use a dynamic topology that adapts to the specific needs of each application. In such a scope, the section "Software-Defined Networks" covers emerging technologies facilitating dynamic network configuration using a centralized control plane implemented in software.

SOFTWARE-DEFINED NETWORKING

This section covers both theoretical approaches as well as practical implementations. Solutions highlighted in the following paragraphs combine three aspects: computational patterns present in most of Big Data services, data-centers network architectural improvements such as hierarchical topologies (e.g., fat trees) and dynamic routing algorithms leveraged by the adoption of technologies such as SDN. These three aspects combined together allow the adaptation of the network configuration from the core to the aggregation infrastructure layer to better suit Big Data application needs.

Routing and scheduling decisions rely on the traffic matrix. Such a matrix can be observed in real-time at the network level but can also be predicted in order to plan next course of action. The traffic matrix usually reflects the flow's size, duration and frequency for each pair of nodes and eventually application instances or even between multiple tasks of a single job. Alternatively, Big Data applications can interact with a central controller to expose their current usage and needs. These two types of approaches are differentiated in Figures 7.6a and 7.6b. In every cases, there is a Big Data application controller or manager (e.g., the job-tracker or the resource manager in Hadoop), which is in charge of triggering and monitoring the tasks. In Figure 7.6a, a monitoring service gathers traffic from forwarding devices and sends the information to the network controller itself which is in charge of taking routing decisions. The monitoring can even be done by OpenFlow as an OpenFlow controller can request such statistics from OpenFlow switches [24]. In this case, both the monitor and

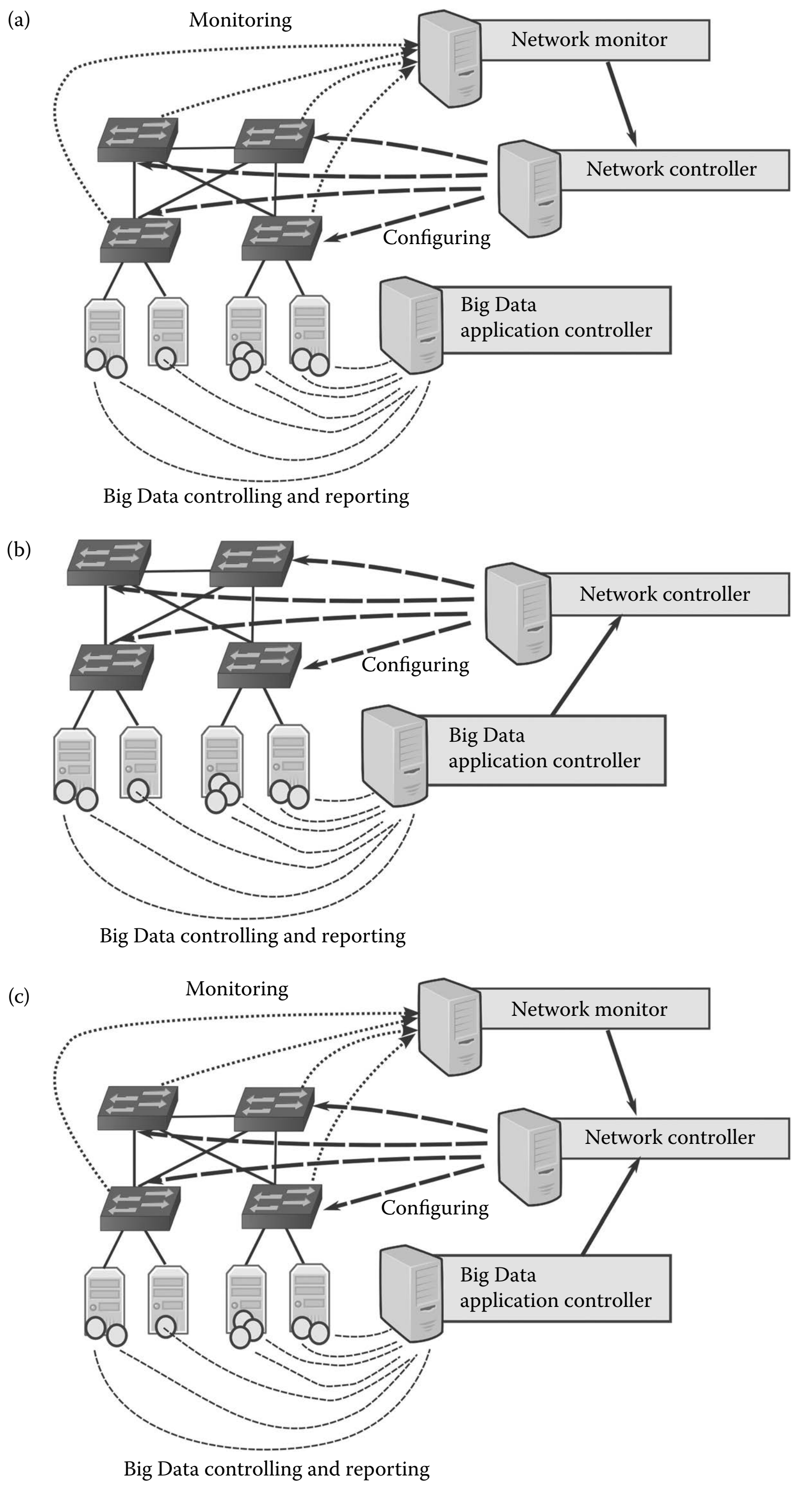
(a)
Monitoring
Network monitor
Network controller
Configuring
Big Data application controller
Big Data controlling and reporting
(b)
Network controller
Configuring
Big Data application controller
Big Data controlling and reporting
(c)
Monitoring
Network monitor
Network controller
Configuring
Big Data application controller
Big Data controlling and reporting

FIGURE 7.6 The different type of *-aware networking (small circles represent a task of a Big Data process). (a) Traffic-aware networking, (b) application-aware networking, and (c) hybrid awareness.

controller are merged in a single entity. In a second scenario (Figure 7.6b), the Big Data controller itself sends information about the running jobs to the network controller which can thus take proper configuration actions. Finally, it is also possible to imagine a hybrid approach (Figure 7.6c) where both types of information are made available to the controller. It might be useful if the level of details from the Big Data controller is coarse-grained.

To summarize, the different methods covered in the following subsections are actually similar to conventional networking (select better paths, minimize congestion, etc.), but they rely on a higher and more dynamic coupling between the network configuration and applications (or the corresponding traffic).

Software-Defined Networks

In recent years, SDN emerged introducing a new layer of abstraction for more flexible network management. Under this approach, switches are just forwarding devices while most of the control (e.g., routing decisions) is performed in a central controller. As a result, a network can be built with merchant silicone and can be programmatically controlled by the central control plane. This eventually results in reduction of both capital expenditures (CAPEX) and operation expenditures (OPEX).

SDN decouples the data and the control plane as shown in Figure 7.7, where

- *Control plane*: The concept of the control plane is to have a dedicated communication channel for exchanging signalization messages among forwarding and management devices. Most of the available products for SDN expose a North Bound application programming interface (API) for applications to subscribe to real-time statistics and service usage.
- *Data plane*: This layer, also referred as the forwarding plane, performs the actual switching/forwarding of the network traffic. The traffic in this plane is accounted and measured but not interpreted by any decisional algorithms.

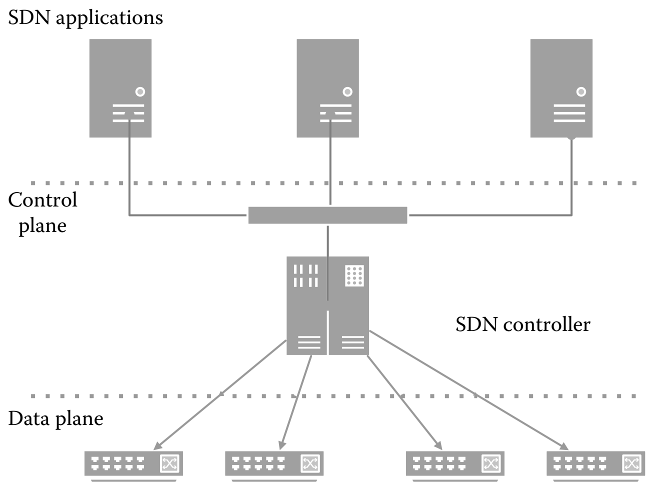

FIGURE 7.7 SDN architecture example.

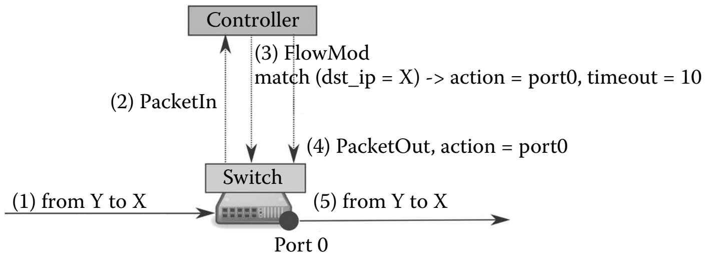
Controller
(3) FlowMod
match (dst_ip = X) -> action = port0, timeout = 10
(2) PacketIn
(4) PacketOut, action = port0
Switch
(1) from Y to X
(5) from Y to X
Port 0

FIGURE 7.8 SDN with open flow rules.

Additionally, the application layer is composed of custom-made applications. The latter subscribe to the North Bound API of the SDN controller to enable extra functionality not provided by the out of the box controller. For example, these applications might be security oriented [25] or for routing purposes [26]. OpenFlow [27] is adopted as de facto standard control protocol. OpenFlow acts as the communication protocol between switches and controllers (e.g., NOX, Floodlight, POX). An OpenFlow rule consists of two parts: a match field, which filters packet headers, and instructions, indicating what actions to take with the matched packets. Upon arrival of a packet at a switch, the controller decides on the route of the packet and sends the corresponding rule to the switch. This event is known as FlowMod. Finally, the packet is sent (PacketOut). Figure 7.8 illustrates an example where a routing action is taken upon arrival of a packet with destination X and source Y. Additionally, a controller can provision switches with flow tables entries in advance. Hence, a PacketIn message is not required to emit an event FlowMod. The rules also have soft (last seen packet) and hard (maximum absolute value) timeouts, and after expiration of these timeouts the rule is removed.

While originally proposed for campus networks, the modification proposed by Curtis et al. [28] consists of reducing the overhead induced by OpenFlow to enable a more efficient flow management for Big Data analytics applications networking through the extensive use of wildcard rules within the switches to avoid invoking the OpenFlow controller for each new flow. However, the extensive use of wildcards on OpenFlow might cause loss of granularity in the statistics derived from the counters on the controller and evidently on routing and scheduling decisions. As mentioned in Curtis et al. [28], DevoFlow aims to devolve control by cloning rules whenever a flow is created using wildcards. The cloned rule will replace the wildcard fields using the clone's specific information. Additionally, DevoFlow enriches OpenFlow rules by including local routing actions (without relying on the OpenFlow controller), such as multipath routing. This last feature allows to rapidly reconfigure the route for a given flow leveraging the flow scheduling.

Traffic-Aware Networking

The Topology Switching approach [29] proposes to expose several adaptive logical topologies on top of a single physical one. It is similar to the allocations problem in VM placement introduced in section "VM Placement for Reducing Elephant Flow Impact" by trying to assign every individual flow to a specific path to optimize an objective. The optimization objectives can be multiple in case of Big Data applications, the most important one is the total capacity, that is, trying to use the available bandwidth as much as possible in order to reduce the job completion time. For example, considering a fat-tree topology as showed in Figure 7.3, every MapReduce typical bisection traffic is considered as a separate routing task. Thus, each task runs an instance of a particular routing system. For every routing system, a pre-allocated bandwidth is established in the physical topology to maximize the bandwidth. Topology Switching is implemented in a central topology server, responsible for allocating resources but also for subtracting unused resources and collecting metrics. The two metrics used in this approach are the bisection bandwidth and the all-to-all transfer. Bisection bandwidth is used to measure the topology ability to handle concurrent transfers at the physical layer. The all-to-all metric is used to evaluate how the logical topologies react under a worst case scenario. Based on both metrics, the Topology Switching approach runs an adaptive algorithm for readjusting the logical configurations for the virtual networks. Topology Switching offers an alternative to "one-size fit all" data-center design, providing a good trade-off between performance and isolation.

Hedera [30] scheduler assigns the flows to nonconflicting paths similarly to Al-Fares et al. [3], especially by aiming at not allocating more than one flow on routes that cannot satisfy its network requirements in terms of aggregate bandwidth of all flows. Hedera works by collecting flow information from the aggregation layer switches, then computing nonconflicting paths, and reprogramming the aggregation layer to accommodate the network topology in order to fulfill the MapReduce jobs requirements. More especially, bottlenecks can be predicted based on a global overview of path states and traffic bisection requirements in order to change the network configuration.

Application-Aware Networking

The methods described in this section improve the network performance by scheduling flows according to application-level inputs and requirements. At the transport layer, flows are not distinguishable from each other but groups of computing nodes in Big Data Application usually expose an application semantic. For example, an application can be composed of several shuffle phases and each of them corresponds to a specific set of flows. Furthermore, a Big Data application can evaluate its current stage. For instance, in a MapReduce task, the mapper status (completion time) is computed from the proportion of the data, from the source, which has been read and such a completion time can approximate the remaining data to transfer. Therefore, a mapper having read 50% of its data source and having already sent 1GB of data should send approximately another 1GB. This is an approximation and it cannot be guaranteed that the mapper will send as much information for the remaining data it has to read. For example, a usual example where a mapper sends a <key,value> pair for each read line can also apply some filtering and so may emit nothing based on the input data.

Therefore, some methods build a semantic model reflecting Big Data application needs. The semantic model used for these approaches associates the network traffic to be managed with the characteristics and the current state of the application it originates from. This model might differ among the different proposed works but generally aims at assessing the state of the Big Data applications and their related flows.

In this context, Ferguson et al. [31] propose to optimize network performance by arranging QoS policies according to application requests. Host nodes running Big Data applications can exchange messages within their proposed framework called PANE to submit QoS policies similarly to what can be done with conventional networks (see section "Flow Scheduling"). Naturally, this approach will lead to traffic oversubscription under high traffic demand circumstances. To solve this issue, users have also to provide conflict resolution rules for each QoS rule they submit into the system. Also, this approach can be employed for implementing security policies such as denial of service prevention by setting a top hierarchy policy triggered at the SDN controller.

OFScheduler [32] is a scheduler which assesses the network traffic while executing MapReduce jobs and then load-balances the traffic among the links in order to decrease the finishing time of jobs based on the estimated demand matrix of MapReduce jobs. OFScheduler assumes that MapReduce flows can be marked (e.g., by Hadoop itself) to distinguish those related to the shuffle from those related to the load balancing (when a task is duplicated). The scheduling first searches for heavily loaded links and then selects flows to be offloaded by giving the preference to (1) load-balancing flows, and (2) larger flows in order to limit the impact on performance (cost of the offloading due to OpenFlow rule installation). The reason for (1) is that it corresponds to a duplicated task the original of which may finish somewhere else in the data-center unlike the others. The rationale behind (2) is to minimize the global cost of offloading and so by moving big flows, there are more chances to remedy the problem of the link load without rescheduling additional ones.

Assuming optical links, Wang et al. [33] describe an application-aware SDN controller that configures optical switches in real time based on the traffic demand of Big Data applications. By enabling the Hadoop Job Scheduler to interact with the SDN controller, they propose an aggregation methodology to optimize the use of optical links by leveraging intermediate nodes in the aggregation. In the simplest case, when a single aggregate has to gather data through *N* switches whereas the number of optical links is lower, it has to go through multiple rounds (optical switching) in order to complete the job. The other switches only using a single connection to the aggregating switch can also be connected together to act as intermediate nodes to form a spanning tree rooted in the aggregator and so to avoid the multiple rounds. Such a principle (many to one) is extended toward general case with any to many jobs or when multiple single aggregation overlaps (e.g., different sources overlap their aggregators). This requires more complex topologies such as torus. Other data center network topologies discussed in this chapter such as DCell or CamCube also make use of high redundancy to build similar shaped topologies. Building a torus topology is more complicated than a tree because the search space for suitable neighbors is larger, a greedy heuristic is used to support the traffic demand as much as possible. The routing algorithm within the torus topology is meant to exploit all possible optical paths. Authors also propose to assign weights to the optical links for load-balancing purposes on the torus topology.

FlowComb [34] is a combination of proactive and reactive methods for flow scheduling. It allows the Hadoop controller to specify requirements but also promotes the use of a statistic-based method that predicts based on the network load of previous runs. Hence, this approach lies between application-aware and traffic-aware. Based on that, any routing or scheduling approach described in section "Traffic-aware Networking" could be applied, especially Hedera [30] which has been chosen by the authors. The central decision engine gathers all the job pertinent data and creates a set of Open Flow rules to be installed temporarily and erased after job completion. However, the main drawback of the proactive method using estimation is that about 30% of jobs are detected after they start, and 56% before they finish.

Coflow [35] proposes a full reactive method, which only after receiving the Hadoop Job Scheduler network requirements is able to yield results. Its implementation exposes an API for declaring flows at application level. This API can be used, for example, from the Hadoop Job Scheduler as it is mentioned by the authors to express on demand bandwidth requirements at the different phases of a MapReduce job. Actually, CoFlow introduced an abstraction layer to model all dependencies between flows in order to schedule an entire application, that is, a set of flows, and not only a single flow.

In contrast with the methods described previously, Dixit et al. [36] propose an approach for routing on a packet basis by splitting the flows in chunks similarly to TCP. These chunks are distributed to the available ports of a switch using different strategies: random, round robin, and counter based. However, the main limitation of this approach is the necessity to reorder the chunks.

CONCLUSIONS

Big Data applications are a major representative in today's cloud services, which have also guided the network design and configuration for performance purposes. For example, the fat-tree network topology is a popular choice among data-centers hosting Big Data applications. Also, the usage of ECMP as a routing algorithm leverages the notion of flow routing for a better efficiency in redundant-linked networks. Complementary to the fat-tree approach, the DCell and BCube design patterns propose a high degree or almost full connectivity between the nodes of the data-center. The usage of these kind of topologies is tightly related to the type of applications running over the network. Therefore, one size (network architecture/topology) does not fit all applications and some will experience degraded performance. To cope with this situation, alternatives in the field of dynamic routing and flow scheduling have been proposed.

The network topology can be adapted dynamically to meet the application bandwidth needs in terms of data transfer but also to reduce the latency and improve the Big Data job's finishing time. Many of the solutions proposed in this field consist in regrouping application nodes (VMs) that concentrate a high volume of data to be transferred.

Programmable networks are more flexible in having a central controller that can take a lead role in flow scheduling. Many Big Data applications have an observable traffic pattern which is exploited by several works to propose specific scheduling to make more efficient network usage (e.g., load balancing, traffic management, and resources allocation).

In this direction, several authors have highlighted the notion of "network awareness". In general, two kinds of application state-full controllers and network architectures have been proposed: Passive application controllers (traffic-awareness) are those that take the traffic matrix as input; on the active controllers, there is an interface that allows the application, for instance the Hadoop Job Scheduler, to interact with the network controller about the job status.

Furthermore, applications can also leverage network awareness such that they adapt themselves to network conditions like for instance bandwidth usage and topology. This has been demonstrated in Chowdhury et al. [37] for different types of applications including Big Data ones.

In summary, network awareness seems to be a very promising direction for Big Data applications and its early adoption has already shown improvements. Programmable networks are a fundamental enabler for leveraging the statefulness of the controllers, and accordingly provide customized support for Big Data applications.

REFERENCES

1. Armbrust, M., Fox, A., Griffith, et al. A view of cloud computing. *Communications of the ACM* 53 (4), 2010, 50–58.
2. Kavulya, S., Tan, J., Gandhi, R., and Narasimhan, P. In *International Conference on Cluster, Cloud and Grid Computing*, CCGrid, IEEE/ACM, Illinois, USA.
3. Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. *SIGCOMM Computer Communication Review* 2008, vol. 38, ACM, New York, NY, USA, pp. 63–74.
4. Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y. D., and Moon, B. Parallel data processing with mapreduce: A survey. *SIGMOD Record* 40 (4), 2012, 11–20.
5. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J. et al. Storm@twitter. In *SIGMOD International Conference on Management of Data* 2014, ACM, Utah, USA, p. 18.
6. Pandey, S., Wu, L., Guru, S. M., and Buyya, R. A particle swarm optimizationbased heuristic for scheduling workflow applications in cloud computing environments. In *International Conference on Advanced Information Networking and Applications* 2010, AINA, IEEE, Perth, Australia, pp. 400–407.
7. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. XEN and the art of virtualization. *ACM SIGOPS Operating Systems Review* 37 (5), 2003, 164–177.
8. Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. KVM: The Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, Ottawa, Canada, 2007, vol. 1, pp. 225–230.
9. Rosenblum, M. VMWare's virtual platform. *In Proceedings of Hot Chips*, Palo Alto, CA, USA 1999, 185–196.
10. Yao, Y., Cao, J., and Li, M. A network-aware virtual machine allocation in cloud datacenter. In *Network and Parallel Computing, vol. 8147 of Lecture Notes in Computer Science*. Springer, New York, USA, 2013.
11. Cisco Data Center Infrastructure 2.5 Design Guide. 2008. http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5a_book.html.
12. Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Conference on Data Communication* 2009, SIGCOMM, ACM, Barcelona, Spain, pp. 39–50.

13. Leiserson, C. E. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE* C-34 (10), 1985, 892–901.
14. Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., and Lu, S. DCell: A scalable and fault-tolerant network structure for data centers. In *Conference on Data Communication* 2008, SIGCOMM, ACM, Seattle, WA, USA.
15. Abu-Libdeh, H., Costa, P., Rowstron, A., O'Shea, G., and Donnelly, A. Symbiotic routing in future data centers. *Computer Communication Review* 40 (4), 2010, 51–62.
16. Singla, A., Hong, C.-Y., Popa, L., and Godfrey, P. B. Jellyfish: Networking data centers randomly. In *Conference on Networked Systems Design and Implementation* 2012, NSDI, USENIX Association, San Jose, California.
17. Iselt, A., Kirstadter, A., Pardigon, A., and Schwabe, T. Resilient routing using MPLS and ECMP. In *Workshop on High Performance Switching and Routing* 2004, HPSR, IEEE, Arizona, USA, pp. 345–349.
18. Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., Papen, G., and Vahdat, A. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *SIGCOMM* 2010, ACM, New Delhi, India, pp. 339–350.
19. Lu, G., Guo, C., Li, Y., Zhou, Z., Yuan, T., Wu, H., Xiong, Y., Gao, R., and Zhang, Y. Serverswitch: A programmable and high performance platform for data center networks. In *Conference on Networked Systems Design and Implementation* 2011, vol. 11 of NSDI, USENIX, San Jose, California, pp. 15–28.
20. Chen, K., Guo, C., Wu, H., Yuan, J., Feng, Z., Chen, Y., Lu, S., and Wu, W. Dac: Generic and automatic address configuration for data center networks. *Transactions on Networking* 20 (1), 2012, 84–99.
21. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. RFC 2475: An Architecture for Differentiated Service, IETF, California, USA, 1998.
22. Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S. RFC 2205: Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification, September 1997.
23. Dogar, F. R., Karagiannis, T., Ballani, H., and Rowstron, A. Decentralized task-aware scheduling for data center networks. In *SIGCOMM* New York, NY, USA, 2014, ACM.
24. Chowdhury, S. R., Bari, M. F., Ahmed, R., and Boutaba, R. PayLess: A low cost network monitoring framework for software defined networks. In *Network Operations and Management Symposium* 2014, NOMS, IEEE/IFIP, Krakow, Poland, p. 16.
25. Roschke, S., Cheng, F., and Meinel, C. Intrusion detection in the cloud. In Dependable, Autonomic and Secure Computing, 2009. DASC'09. *Eighth IEEE International Conference on* 2009, IEEE, Chengdu, China, pp. 729–734.
26. Dinh, H. T., Lee, C., Niyato, D., and Wang, P. A survey of mobile cloud computing: Architecture, applications, and approaches. *Wireless Communications and Mobile Computing* 13 (18), 2013, 1587–1611.
27. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review* 38 (2), 2008, 69–74.
28. Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. Devoflow: Scaling flow management for high-performance networks. In *Computer Communication Review* 2011, vol. 41, ACM SIGCOMM, Toronto, ON, Canada, pp. 254–265.
29. Webb, K. C., Snoeren, A. C., and Yocum, K. Topology switching for data center networks. In *Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services* 2011, Hot-ICE, USENIX, Boston, MA.
30. Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. In *Symposium on Networked Systems Design and Implementation*, NSDI, USENIX, San Jose, California.

31. Ferguson, A. D., Guha, A., Liang, C., Fonseca, R., and Krishnamurthi, S. Participatory networking: An api for application control of SDNS. In *SIGCOMM* 2013, ACM, Hong Kong, China, pp. 327–338.
32. Li, Z., Shen, Y., Yao, B., and Guo, M. *Ofscheduler: A Dynamic Network Optimizer for Mapreduce in Heterogeneous Cluster*. Springer, New York, USA, pp. 1–17.
33. Wang, G., Ng, T. E., and Shaikh, A. Programming your network at run-time for big data applications. In *First Workshop on Hot Topics in Software Defined Networks* 2012, HotSDN, ACM, Helsinki, Finland, pp. 103–108.
34. Das, A., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G., and Yu, C. Transparent and flexible network management for big data processing in the cloud. In *Workshop on Hot Topics in Cloud Computing*, Berkeley, CA, 2013, USENIX.
35. Chowdhury, M. and Stoica, I. Coflow: A networking abstraction for cluster applications. In *Workshop on Hot Topics in Networks*, 2012, HotNets, ACM, Redmond, WA, USA, pp. 31–36.
36. Dixit, A., Prakash, P., and Kompella, R. R. On the efficacy of fine-grained traffic splitting protocolsin data center networks. In *SIGCOMM* 2011, ACM, Toronto, ON, Canada.
37. Chowdhury, M., Zaharia, M., Ma, J., and Jordan. Managing data transfers in computer clusters with orchestra. In *SIGCOMM Computer Communication Review* 2011, vol. 41, ACM, Toronto, ON, Canada, pp. 98–109.