Software-Defined Networking Enabled Big Data Tasks Scheduling: A Tabu Search Approach

Mina Soltani Siapoush, Shahram Jamali, and Amin Badirzadeh

***Abstract*—The growth of information technology along with the revolution of the industry and business has led to the generation of an enormous amount of data. This big data needs a platform beyond the traditional data possessing context that relies on some computational servers communicating through a network in its lower layer. One of the most important challenges in data processing is how to transfer the big batches of data between the servers to achieve fast responsiveness. Consequently, the underlying network plays a critical role in the performance of a big data analysis platform. Ideally, this network must use the shortest path that has the lowest amount of load, to transfer the large-scale data. To address this issue, we propose a software-defined networking (SDN) enabled scheduling method that uses the tabu search algorithm to schedule big data tasks. The proposed algorithm not only considers data locality but also uses the network traffic status for efficient scheduling. Our extensive simulative study in the Mininet emulator shows that the proposed scheme gives high performance and minimizes job completion time.**

***Index Terms*—Big data, cross-layer, Hadoop, OpenFlow, software defined networking, task scheduling.**

I. INTRODUCTION

DATA processing is a rapidly growing part of business and industry. These days, big data define databases so large and complicated that traditional data management tools or processing techniques have got some limitations to handle. The amount of data from various sources, for example, the Internet of things (IoT), scientific research, and social networking websites (Twitter, Facebook, Instagram, etc.) is increasing sharply. On the other hand, the latest advances in information technology make it easier to generate data and as a result, an enormous amount of data is generated every day. Hence, big data analysis is a needed service in today's Internet companies. For example, Google processes data over one hundred Petabytes (PB), Facebook generates log data of approximately 10 PB per month, and Baidu, a Chinese company, processes data of tens of PB [1]. To process this massive data efficiently, MapReduce [2] with an open-source implementation called Hadoop as the main solution. It is the common computing framework used by Yahoo!, Amazon, Facebook, etc. Nonetheless, it is important to note that the big data frameworks wouldn't be possible to work without supporting the network underlying due to their extremely large volume of data and computing complexity [3]. To this end, some performance-oriented policies must be implemented in the network. To express the desired high-level network policies, including traffic engineering, QoS, and load balancing, network operators need to configure each individual network device separately using low-level and often commands that are vendor-specific.

To overcome these challenges, software-defined networking (SDN) architecture is proposed. As a new networking paradigm, SDN enables us to improve the performance of the network. In this architecture, the control logic of the network is separated from the devices such as routers and switches forwarding the traffic. Through this separating, the network switches play a forwarding role only and the controlling actions such as applying policies and managing the flows are taken in a controller by (re)configuring the network [4]. In Hadoop frameworks, one of the most challenging issues that impact the performance of data processing is the "minimum makespan" problem. Although many types of research have been done on task scheduling, they can only find a near-optimal solution or have high complexity in their computation due to the limitation of traditional networks to provide a global view of the network. To resolve this NP-complete problem, we have to consider the distance of servers from where data is located. Although to schedule the big data tasks, we prefer to assign the tasks to the local servers, by selecting the local servers frequently, these servers may be overloaded, and the waiting time for tasks to execute increases.

On the other hand, in assigning tasks to the remote servers, the corresponding bandwidth and time are needed for moving data. Therefore, we are going to make a trade off between choosing the remote and local servers to assign the tasks and finding solutions to shorten the job completion time. To address the mentioned problem, we leverage the abstraction of SDN which can provide the network state information to monitor and manage the networks and make the related decisions accurately. In order to reach a solution in this search space, we employ the tabu search algorithm which provides solutions very close to optimal, if not the best. The proposed algorithm not only shortens the completion time of task scheduling in compared the related works by considering the state of the network such as congestion ratio, but it improves the data locality which is vital for performance because the network biSection bandwidth in a large cluster is much lower than the aggregate bandwidth of the disks in the machines.

The rest of the paper is organized as follows: Section II

Manuscript received July 10, 2019 revised December 28, 2019; approved for publication by Martin Reisslein, Division 3 Editor, Januarry 23, 2020.

M. Soltani is with the Departmet of Computer Engineering, Islamic Azad University, Ardabil, Iran, email: soltani.netc@gmail.com.

S. Jamali and A. Badirzadeh are with the Department of Computer Engineering, University of Mohaghegh Ardabili, Ardabil, Iran, email: Jamali@uma.ac.ir, aminbadirzadeh@gmail.com.

M. Soltani is the corresponding author.

Digital Object Identifier: 10.23919/JCN.2023.000002

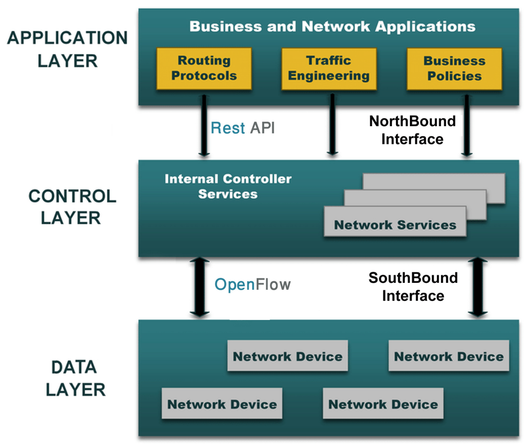

Fig. 1. A simplified view of an SDN architecture [4].

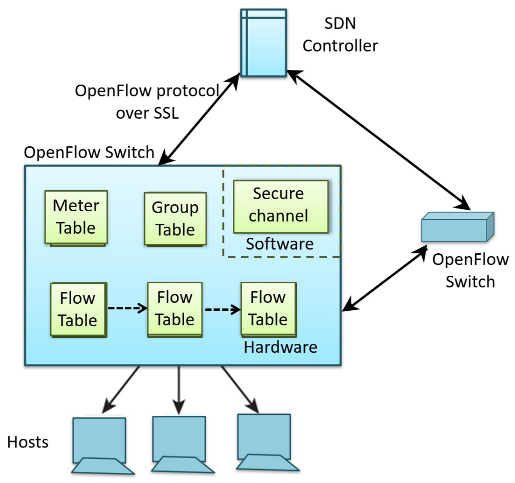

Fig. 2. Main components of an OF switch [9].

presents the background of the issue, Section III introduces the related works. Section IV depicts the problem formulation and Section V brings the proposed algorithm. Experimental evaluation is shown in Section VI and finally, we make a conclusion in Section VII.

II. BACKGROUNDS

In this section, we bring some preliminaries about this research. First, the SDN and big data architectures are explained in detail and then the other related concepts are described.

A. SDN Architecture

SDN architecture consists of three layers (as shown in Fig. 1):

- **Data plane** The bottom layer of SDN is known as the data plane. It consists of packet-forwarding devices such as Ethernet, optical and virtual switches, routers, and access points.
- **Control plane** The middle layer includes the controllers and network services. The controllers are responsible for routing and deciding about traffic transmitting to related destinations.
- **Application plane** The upper layer is placed on top of the central layer. This layer uses a high-level programming language to provide some functions, such as security monitoring, energy-efficient networking, traffic engineering, etc [4].

The interface between the application layer and the control layer is referred to as the northbound interface (NBI), while the interface between the control layer and the data layer is referred to as the southbound interface (SBI). There are a variety of standards for these interfaces, e.g., the OpenFlow (OF) protocol [5] is mostly used for the SBI. For the NBI, representational state transfer (REST) is generally defined as a software architectural style that supports flexibility, interoperability, and scalability [6].

Using the separation of control and data planes, applications achieve their specific purposes such as quality of services, traffic engineering, and monitoring. In addition, the SDN controller manages and controls the switches known as OF switches. In other words, the network can be managed by adding some features to the control plane automatically without any changes in the forwarding devices [5]. To do this, SDN architecture uses a protocol called OF which is introduced as the common standard interface to communicate between the control and the data layer. It is designed by the open networking foundation (ONF) [7] and provides a means to control a switch without vendors that have to reveal any source codes of their peripherals [8]. An OF switch which is shown in Fig. 2 consists of three main items: a) Tables that include flow tables, meter table, and group table, b) the secure channel that is used mostly as an SSL channel among switches and SDN controller, and c) OF protocol that is used to control and manage the switches by communicating with them.

Through the OF channel, the SDN controller manages the OF switches and receives the network status such as network traffic and bandwidth. The OF protocol supports three message types, controller-to-switch, asynchronous, and symmetric messages. The controller-to-switch messages are used to manage or inspect the state of the switch. The asynchronous messages are sent by the switches and used to update the controller about its state, and finally, the symmetric messages are initiated by either the switch or the controller and sent without solicitation [8]. By transmitting these messages the controller is able to have a global view of the network [9]. When the first table match is found, a set of actions associated with that table entry is executed. They may direct the packet to a particular switch port in a particular queue, send it to the controller, or drop the packet [8]. It is generally intended that when the existing flow tables do not know how to handle a packet, it is sent to the controller, which may respond by installing rules on the switch to properly process similar packets [9].

B. Big Data

There are different types of definitions for big data in the literature [10]. In the comparative definition, big data can be defined as ‘data sets which are unstructured or time-sensitive or very large that cannot be processed by relational database engines and is beyond the ability of typical database tools to capture, store, manage, and analyze.'

1) Big data characteristics: Big data has some parameters which make it distinct from other structured data like relational database systems. The main characteristics are *volume, variety, velocity, value,* and *veracity*.

a) The *volume* of data is the major feature of big data because the volume of the data is in terabytes and petabytes [11].
b) *Variety* refers to the different formats of data such as images, audio, video, document, social media messages, etc [12].
c) The basic concern for *velocity* is the completeness and consistency of data streams and getting the demanded result matching on a specific time.
d) Another attribute is the *value* which is the results that come out from analyzing data and defining how will we get a better result from the data stored.
e) *Veracity* describes the quality of data and specifications such as noiselessness, completeness, and accuracy are concerned.

These 5Vs have been specified as the main features of big data which make it different from traditional data. Other features that play important roles in how to capture and analyze data are as follows: *Viscosity, variability, volatility, viability*, and *validity*.

2) Big data framework: Hadoop is an open-source platform that can process vast amounts of data and store them in distributed systems. It usually consists of two main components: HDFS and MapReduce.

The Hadoop distributed file system (HDFS) is a file system that creates multiple replicas of data blocks for reliability and places them on computational nodes in the cluster. HDFS divides each file into 64 MB blocks, and stores several copies of each block on different systems (by default 3 copies) [13].

MapReduce is a computing engine developed at Google and runs in two phases: Map and reduce phases (Fig. 3). First of all, every block including some number of records is processed and each record contains a key/value pair. In map phase, a specified node named the master node takes the input, divides it into the data splits, and assigns them to slave nodes [14]. The reduce phase then begins with sorting and shuffling the partitions that are produced by the map phase.

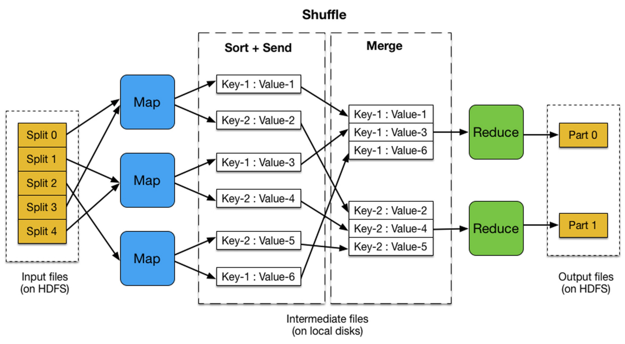

Fig. 3. MapReduce workflow.

III. Related Works

Big data job scheduling is one of the most vital elements in big data analytics, which has drawn much more attention in recent years. The important issue in all scheduling algorithms is minimizing the makespan which needs to meet solutions that shorten the job completion time and have a significant impact on the performance of Hadoop systems. The default scheduling method in Hadoop is FIFO which jobs are located in a queue and a job tracker schedules them one after another based on arrival time. This algorithm is known as Hadoop default scheduler (HDS). Later on, setting the priority of a job was added. Facebook and Character contributed meaningfully apply in processing schedulers i.e., fair scheduler [15] and capacity scheduler [16] respectively which after free to Hadoop dominion.

Considering data locality, some methods are proposed such as state-of-the-art balances reduce scheduler (BAR) [17]. Though, this approach either disregards available bandwidth as the base for assigning or does not allocate MapReduce tasks in a global view which results in missing optimal opportunities for scheduling. OFScheduler [18] is another approach that identifies MapReduce traffic load in the network and then balances the load among the links to shorten the completion time of jobs. It first looks for overloaded links and then selects flows to reduce the link's load by granting priority to load-balancing flows.

When the switches consider packet scheduling priorities based on the bandwidth provisioning table from the SDN controller, the network's resources are allocated efficiently, and the power consumption is also reduced for different big data applications [19]. The use of SDN is applied to solve this problem in Hadoop [20]. Specifically, a scheduler known as bandwidth-aware scheduling with SDN (BASS) schedules Hadoop tasks using the SDN and assures the providing data locality in the overall network. It first utilizes the SDN to manage the network bandwidth and allocates it in a time-slot manner. In this regard, an application-aware networking setup based on the SDN controller [21], propose the network configuration which is made it easier by using the integrated control plane in job placement. Due to application awareness of the network, there is an improvement in the big data performance, as well as the completion time of the job that is minimized drastically. As another work, a virtual network of tenants [22] represents the possible case where all the batches of a task can be mapped on the same physical server. In this method, a batch of tasks is split into two cases based on latency and bandwidth such that the given batch cannot be mapped into the same server.

IV. Problem Formulation

As mentioned above, the underlying network has a great impact on big task scheduling. Obviously, if the scheduling process has a global view of the network, it can calculate better scheduling solutions regarding the communication and

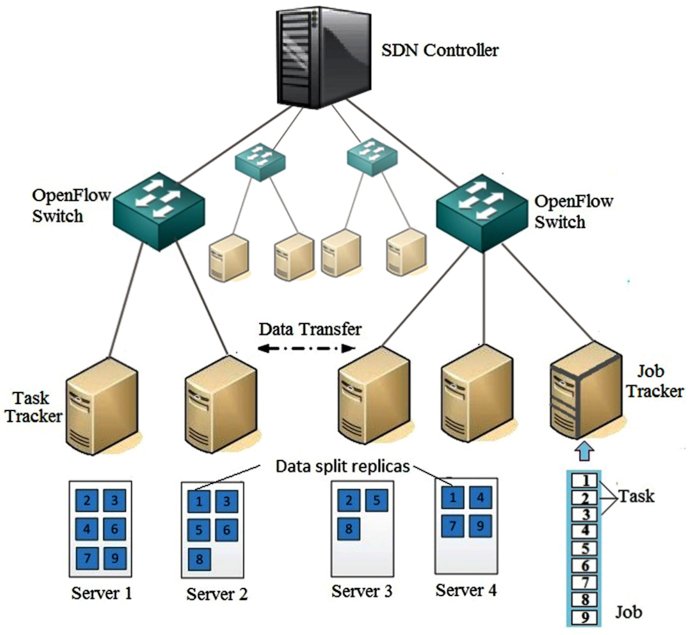

Fig. 4. Architecture of Hadoop cluster enabled with SDN.

computation resources of the network. In SDN-enabled task scheduling, upon receiving the request, the proposed algorithm contacts the network information manager in the controller to get the current state of the network. Based on this information, the algorithm determines where and when this task will be executed. For a better illustration of this model, let us consider the architecture of the SDN-powered Hadoop cluster. As shown in Fig. 4, an OF-controlled Hadoop cluster is composed of the same servers or task trackers, an OF controller, and a job tracker/scheduler. The SDN controller is connected to these servers by OF switches.

A Hadoop MapReduce schema (MR-schema) is a pair (T, S), where T is a set of tasks and S is a set of servers. Let $m = |T|$ be the number of tasks and $n = |S|$ be the number of servers in the cluster. The main entities of this framework are described as follows;

Task: Each task of a Hadoop job is denoted by t_i and can be defined as given in (1):

$$t_i = \ (\sigma_i, PT_i) \tag{1}$$

σ_i: Input data size of the ith task
PT_i: Processing time of the ith task.

Server: Servers are computational resources of a Hadoop cluster. In assigning map tasks, a critical consideration is to keep the computation as close to the data as possible. But to avoid the long waiting time for local servers, assigning the tasks to the remote servers is an alternative solution as shown in Fig. 5. In other words, for task t_i in T, the goal is to find an available server that can yield the earliest completion time among all n servers of the cluster. Table I gives a brief description of each parameter used in this section

Waiting time: As shown in Fig. 5, when task t_i is allocated to server j, either local or remote, it may face some queued

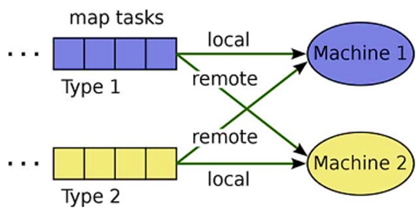

Fig. 5. Task scheduling with data locality.

TABLE I
THE INVOLVED SPECIFICATIONS IN PROPOSED ALGORITHM.

Notation	Description
σ_i	Input data size of task i
BW_{ej}	Bandwidth of jth edge (or link)
$Q_j(ts)$	Queue length of server j at the current time slot
θ_j	Waiting time until server j will be idle, until $Q_j(t)$ be free
$A_{i,j}$	Allocation of task t_i to server j at the current time slot
$TT_{i,j}$	Data transfer time of task i to the remote server j
PT_i	Processing time of task i
$WT_{i,j}$	Waiting time of task i on the server j
$ET_{i,j}$	Execution time of task i on server j
p_k	kth routing path between master and slave servers
V_{p_k}	The switches are involved in kth path
E_{p_k}	The links are involved in kth path
MC_{v_i}	Meter counter of ith switch
MB_{v_i}	Meter band of ith switch
CT_T	The completion time of all tasks T

tasks at the server. server j starts to process this task immediately, if its queue length is zero, otherwise, the task waits until server j becomes idle. Hence, the waiting time can be defined as in (2):

$$WT_{i,j} = \begin{cases} \theta_j & if \ \ Q_j > 0 \\ 0 & if \ \ Q_j = 0 \end{cases} \tag{2}$$

θ_i: The time it takes to serve all previously queued tasks.

Data transfer time: It is clear that assigning a task to a remote server dictates some overhead to the controller and it includes data transfer time. But, sometimes the local servers may be heavily congested, and hence, the remote server may be a better option. In other words, we need to strike the right balance between data locality and load balancing in MapReduce to minimize the makespan. To study the data transfer time, consider the data plane of the underlying network. From the data plane perspective, the topology of the network can be modeled by $G = (V, E)$, where $V = \{v_1, \cdots, v_n\}$ is the set of switches and $E = \{e_1, \cdots, e_m\}$ is the set of links that connect switches each other. Note that each path that is denoted by $P = \{p_1, \cdots, p_K\}$ is a route consists of the set of the switches and links between the source and the destination nodes to send a flow. We use p_k as a path that is generated by the controller. This path includes some switches and links, $V_{p_k} = \{v_i | v_i \in P_k\}$ is the set of switches in the kth path, and $E_{p_k} = \{e_j | e_j \in p_k\}$ is the set of edges/links in the kth path.

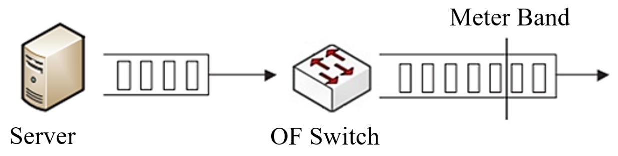

Fig. 6. Meter table of OF switch.

Regarding these definitions, to compute the data transfer time, we need to consider the data size of task i (σ_i) and the status of the links and switches, which data is transmitted as shown below:

$$TT_{i,j} = \sum_{e_j \in E_{p_k}} \frac{\sigma_i}{BW_{e_j}} \times (CR_{p_k} + 1), \tag{3}$$

where BW_{e_j} is the bandwidth of link j of p_k and CR_{p_k} or congestion ratio of p_k, calculated through the meter table (according to Fig. 6) as defined by (4):

$$CR_{p_k} = \sum \frac{MC_{v_i}}{MB_{v_i}} \quad \{v_i | v_i \in p_k\}, \tag{4}$$

in which, MB_{v_i} is the meter band or the maximum number of packets that switch v_i can forward and MB_{v_i} is the meter counter or the number of received packets of switch v_i.

Execution time: For the purpose of simplicity, we assume that all servers are homogeneous, so task i can be processed at the same time on all servers. Therefore, the processing time of task i is defined by PT_i and is the time spent by task i on a server. Now, consider the execution time of task i on server j which is influenced by three parameters as shown in (8):

$$ET_{i,j} = TT_{i,j} + WTi, j + PT_i, \tag{5}$$

where $TT_{i,j}$ is the task data transfer time to the remote server j, $WT_{i,j}$ is the waiting time of task i on the server j, and PT_i is the processing time of task i. It is clear that, when desired server j is a local one, $TT_{i,j}$ is zero. So, (5) can be rewritten as (6):

$$ET_{i,j} = \begin{cases} PT_i + WT_{i,j} & \text{if j} \in \text{local} \\ PT_i + WT_{i,j} + TT_{i,j} & \text{if j} \in \text{remote.} \end{cases} \tag{6}$$

Based on these functions, we can formulate the problem to minimize the completion time of task i as follows:

$$\text{minimize}(C\Gamma_T) \tag{7}$$

Subject to:

$$C\Gamma_T = \sum_{i=1}^{n} ET_{i,j} \quad \forall j \in S. \tag{8}$$

According to all described parameters, Algorithm 1 explains the steps of the task scheduling algorithm. In terms of server load balancing in Hadoop servers, when a task is allocated to a server, the queue lengths of the servers are recorded to calculate the waiting time of just arrived task. As mentioned above, waiting time has a significant role in selecting a server, so the algorithm selects the server with the least queue lengths. Using this approach, neither queues are overloaded nor underloaded.

Algorithm 1 Load-award scheduling by SDN in Hadoop

```
Given the submitted job with $m$ tasks and the $n$ server
Use capability of SDN to:
for ($t_i : i = 1, 2, \cdots, m$) do
    Get input data size of task $i$;
    for ($p_k : k = 1, 2, \cdots, K$) do
        Get the real-time bandwidth ($BW_{e_i}$) for each link;
        Calculate the $CR_{p_k}$
        Tabu search()
        $C\Gamma_T$ = Find a local server with minimum $WT_{i,j}$ for $t_i$;
        $C\Gamma_T$ = Find a remote server with minimum $WT_{i,j} + TT_{i,j}$ for $t_i$;
        if ($C\Gamma_a <= C\Gamma_b$) then
            Assign $t_i$ to the local server $j$;
        else if ($C\Gamma_a < C\Gamma_b$) then
            Assign $t_i$ to the remote server $j$;
        end if
    end for
end for
return The assignment for all $m$ tasks.
```

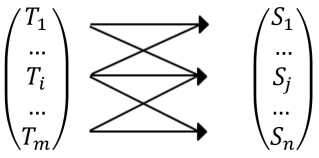

Fig. 7. The solution space of data placement problem.

V. Proposed Task Scheduling Algorithm

To solve the scheduling problem formulated by (7) and (8), we propose a meta-heuristic algorithm based on tabu search (TS) methodology. This algorithm is a general heuristic procedure introduced by Glover [24] to solve large scale optimization problems. The main portion of TS is the tabu list, that is used to avoid being trapped in local optima and revisiting the same solution. From the current solution, it defines a set of solutions, named the neighborhood and at each step, the best one in the set is selected as the new solution. Some attributes of the former solution are then stored in the tabu list. The moves in the tabu list are repeated until an aspiration criterion is satisfied. The neighborhood structures are used in a local search phase to compute a local minimum [25]. The main elements of TS that are introduced as follows, incorporated in various versions optionally.

A. Solution Space

In TS algorithm, a solution is considered feasible until the stop condition is fulfilled and in such a case, the solution is repaired. For this purpose, the algorithm removes the action which has the least impact on the objective value, at each iteration. The solution space of the proposed task scheduling algorithm is shown in Fig. 7. In our case, each task has three input data replicas, so it can be assigned to three local servers.

B. Adaptive Memory

In this section, we describe the procedures used to store and fetch solutions from adaptive memory [26]. Since every solution is split into its elements, it should be noted that the memory is implicitly divided into a number of smaller memories. Adaptive memory is required during the fetch procedure.

C. Neighborhoods

The basic phase of TS is neighborhood that are placed in a sequence of four different structures:

- Inter-route move: A task is moved from one route to another and inserted at the best feasible place.
- Intra-route move: A task is moved from its current position to another feasible position in the same route.
- Swap: Two tasks from two different routes are swapped. Each task is inserted at the best feasible insertion place in its new route.
- Swap-with-new: A task not currently in the solution is swapped with a task in the solution and is inserted at the best feasible insertion place [26].

D. Fitness Function

The quality of a solution in memory depends on the objective value and its contribution to the diversity of solutions. The fitness function for the problem of task scheduling is to minimize the overall time of the complete a job. Although, the processing time of each task on the Hadoop servers are equal as mentioned earlier, so:

$$F_i = \text{minimize } (TT_{i,j} + WT_{i,j}) \qquad \forall j \in S. \tag{9}$$

Obviously, according to this fitness function, the TS algorithm looks for the solutions to assign a task that has the minimum data transfer time $(TT_{i,j})$ and waiting time $(WT_{i,j})$ as possible. As expected, when a task is allocated to a local server, the transfer time is zero, and hence the solution has a high chance of acceptance. However, sometimes, according to the state of the network, the transfer time plus waiting time on a remote server is much less than the waiting time on a local server. Since there are m servers, each operation of the task placement takes $O(\text{logm})$ time and this is done for n task with $O(n)$ time. In addition, the algorithm also needs to evaluate the routing paths to assign some tasks to the remote servers. Hence the proposed task-scheduling method takes $O(n(\text{logm} + \text{k}))$ time.

After finding a task assignment, it is not inserted in the final solution immediately. Tabu search algorithm also considers each neighborhood and the recent moves are declared for θ iterations (unless they can improve the best-known solution), where θ is randomly selected in the interval $[\theta\text{min}, \theta\text{max}]$, with $\theta\text{min} = 5$ and $\theta\text{max} = 10$ in our experiment. Now we depict the workflow diagram of the proposed task scheduling algorithm in Fig. 8.

VI. Results and Discussions

In this section, we use MapReduce workloads based on HiBench benchmark suites. The applications that are used to evaluate the effectiveness of the proposed method are listed below:

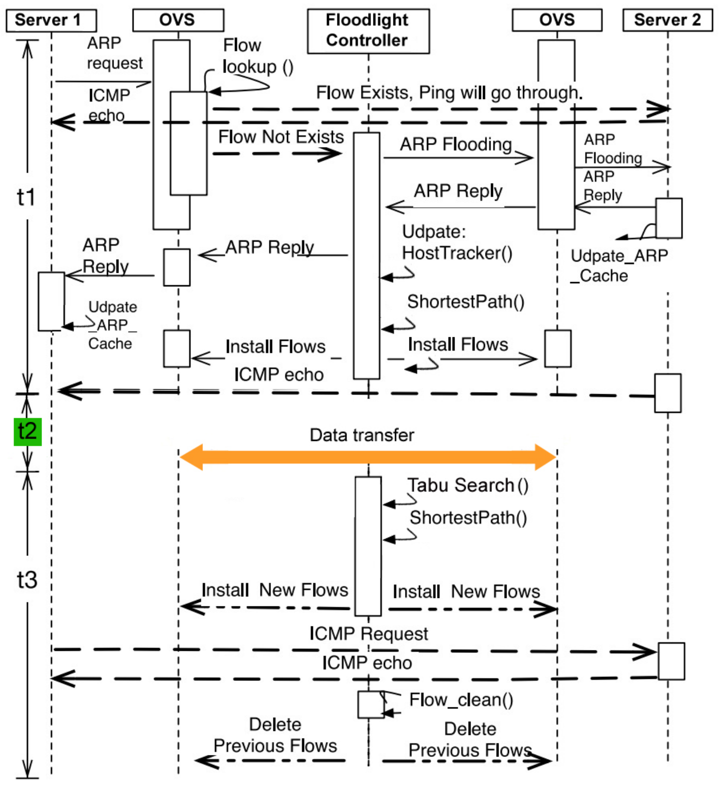

Fig. 8. The proposed task scheduling workflow.

- WordCount: This is the most common application used in MapReduce evaluation. It has text input data where each word is counted. The traces had 50 GB of input data.
- Sort: We also use the sort application which sorts the input data in the text format and is generated by a random text writer [27]. This application is widely used to evaluate the performance factor. In this application, 32 GB of data was sorted to obtain job traces.

A. Experimental Setup

The architecture of the proposed model which is shown in Fig. 9 can be executed in real data-center networks by making some minor changes. For the SDN controller, we use Floodlight for two reasons: First, it is based on Java, which gives the highest performance as stated earlier; and second, it is built using Apache Ant, which is very easy and flexible in use. Regarding the other components, open vswitches (OVSs) and Linux hosts are used as the network switches and physical servers respectively, which are implemented within the Mininet emulator.

In addition, OVS communicates with the ovsdb server using the open vswitch database protocol (OVSDB) and configures virtual network instances via netlink with the kernel, during communicating with the system through the netlink interface. OVS is also used to enable communication between various hosted virtual machines. Accordingly, for the encapsulation of

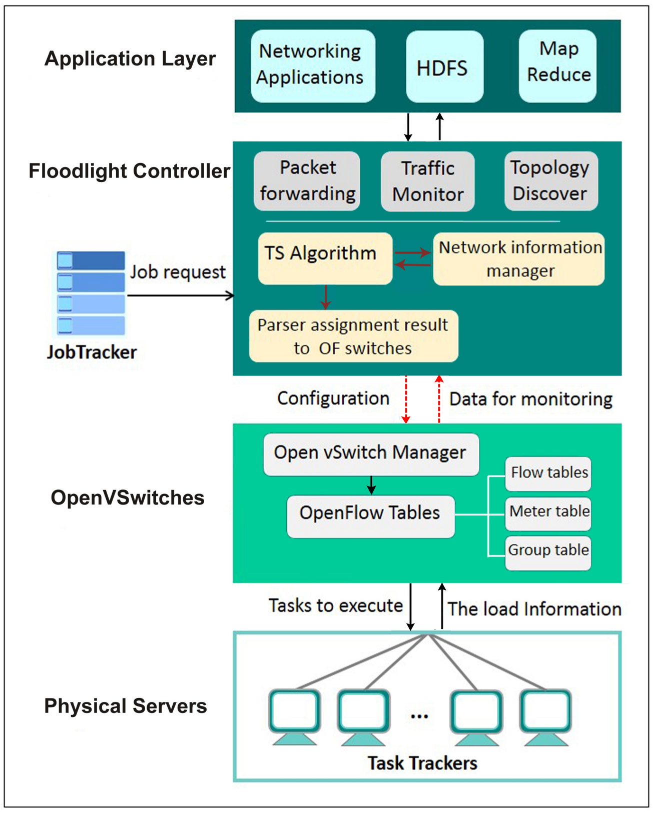

Fig. 9. The proposed task scheduling architecture.

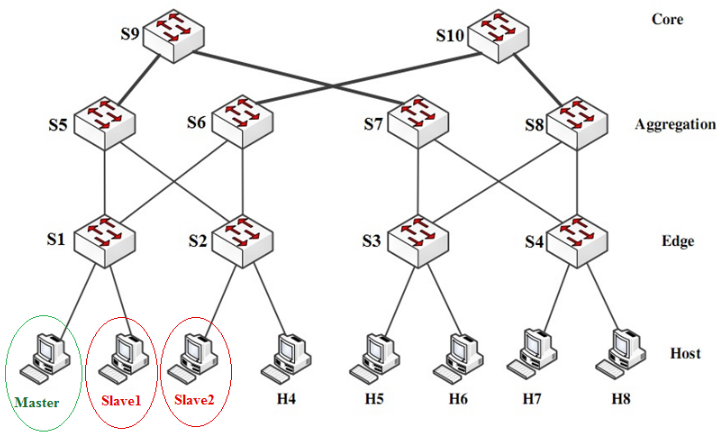

Fig. 10. Fat-tree topology of physical nodes and OF switches.

TABLE II
THE SPECIFICATION OF PROPOSED ALGORITHM SIMULATION.

Component	Configuration
Operation system	Ubuntu 16
Hadoop	Version 2.7
Topology	Fat-tree
Network's links	100 Mb/s
Size of data block	64 MB

traffic and the creation of overlay networks between the hypervisor hosts the generic routing encapsulation (GRE) tunnel is used. Being a simple and effective method of transporting data over a public network, GRE lets two peers share data and connect non-contiguous subnetworks.

According to this model, first of all, the job tracker introduces its demands in a job request (e.g., a JSON file). The job request, which is a set of tasks, is fed into the proposed algorithm that is running in the OF controller. Upon receiving the request, the algorithm contacts network information manager to obtain the current state of the network. Based on this state, the algorithm determines where these tasks will be allocated. One important thing we should note is that all the requests are processed by the controller and endures some overhead.

When SDN controller is used for big data applications, its performance could be degraded due to the rapid and frequent update requests on flow table as well as big data transmission and processing. Accordingly, the network controller is designed in such a way that it can accommodate larger flow tables and big data, because the topology-update component manages the network changes while traffic analysis component detects and predicts network failures and links congestion as well [28].

In our Experiment, the Hadoop cluster includes 10 OF switches as shown in Fig. 10. Other specifications are shown in the Table II.

While in SDN architecture, each link or switch has the capability of configuration, this is not always required for just task scheduling, as the same work is done in software-defined optical networks (SDONs) [29]. There is some optional configuration in the network which helps to enhance the efficiency of big data processing, but most of these changes are applied to limit the maximum rate of data flows forwarding to manage the traffic and prevent future issues and problem.

To evaluate the performance of existing task schedulers, first of all, we illustrate the resource usage by Hadoop schedulers in Fig. 11. The main Hadoop scheduling algorithms are managed to optimize the Hadoop cluster resources such as CPUs, memories, and disks. Those algorithms fall mainly into three categories: FIFO, capacity, and fairness schedulers.

As mentioned in the related works section, the default scheduling method in Hadoop is FIFO as known as Hadoop default scheduler (HDS). According to Fig. 11, we can find out the reason for this choice. In FIFO, a job tracker pulled jobs from a work queue, the oldest job first, so it does not need more calculation or has no concept of the priority or size of the job. Hence it is an efficient scheduling algorithm in terms of resource usage. However, the assumption of initial job starting and completion time cannot stand in a real network. In Fig. 12, it can be seen the results of running the proposed scheduling algorithm. The job completion time under running the WordCount and Sort jobs is shown in a variety of data.

We have shown the comparison of the proposed algorithm with the most common scheduling algorithms like HDS [16], BAR [17], and BASS [20] in terms of completion time in Figs. 13 and 14 for WordCount and sort jobs applications, respectively. As mentioned earlier, HDS is the Hadoop default scheduling method that works based on FIFO approach,

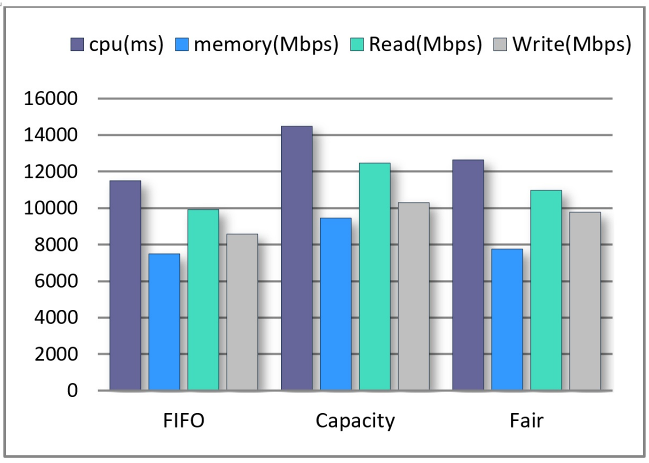

Fig. 11. Hadoop schedulers based on resource usage.

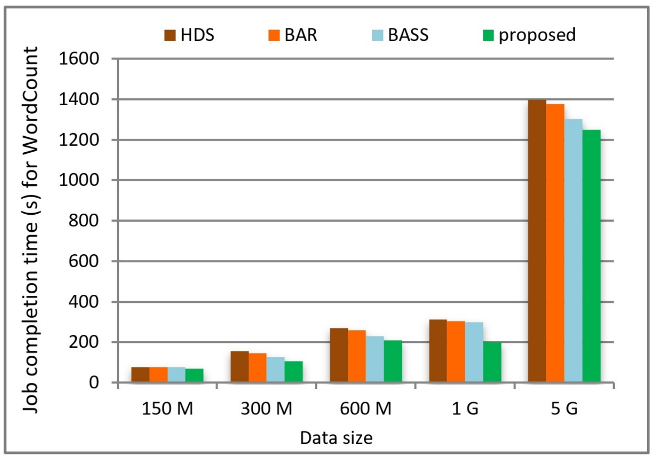

Fig. 13. Job completion time for WordCount job.

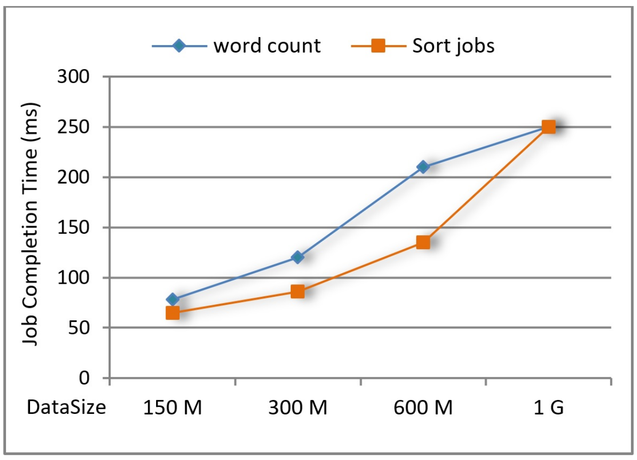

Fig. 12. Job completion time of the proposed algorithm (both WordCount and Sort jobs).

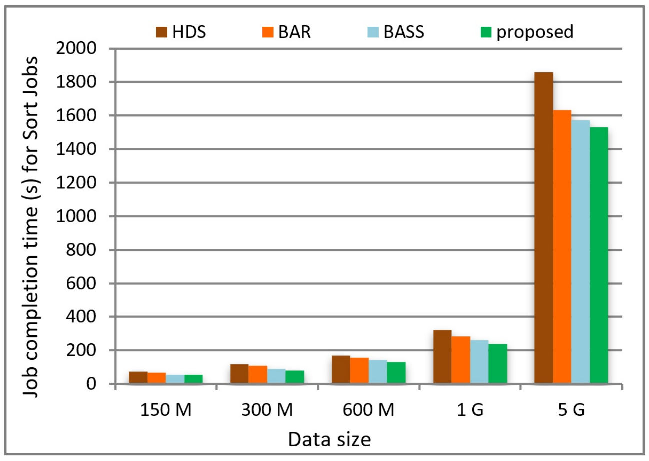

Fig. 14. Job completion time for Sort job.

regardless of the priority. It is obvious that the queue-based methods are inefficient. However, the balance-reduce (BAR) method, schedules tasks by adjusting data locality according to the workload of the cluster. In order to decide about choosing the remote or local server, it is done in two phases, balance and reduce. The balance phase returns a total allocation and the reduce phase repeats to generate a sequence of allocations and returns the best solution. Among these methods, HDS and BAR are the non-SDN approach and rely on traditional networks.

Bandwidth-aware scheduling using SDN as named BASS presents an algorithm to assign bandwidth in a time-slot manner and the occupation time of each link is divided into equal time slots by helping SDN. If a task has a demand for data movement through a path, the scheduler will allocate the related time slots, by guaranteeing that the bandwidths of all links on this path are reserved for the task. This algorithm has a better performance than the two previous algorithms. Among the scheduling methods, we can see that for WordCount job, the proposed scheduler has a minimum makespan compared with other methods. We mentioned that link bandwidth and data locality should be taken into account to aim the optimal overall performance. Sort job also illustrates the authenticity of the proposed scheduler in practice. It is clear that the proposed algorithm outperforms in terms of job completion time in both jobs.

The proposed algorithm has been executed on multiple virtual machines (VMs) as shown in Fig. 15. As expected, by increasing the number of VMs, We do not observe a significant increase in the time of job completion and this is due to the increasing of map/reduce operations. Fig. 16 vividly demonstrates the data locality ratio of the proposed method. In some cases, the experimented data locality ratio (LR) is low, taking the input data of 600 M as an example, LR of BASS is 55%, while LRs of BAR, BASS, and HDS are satisfying, but the makespan of the proposed algorithm is only 220 s and in comparison with other methods is lowest.

VII. Conclusion

In this paper, by profiting from SDN to gain the status of the network at any time, we get every link's bandwidth for improving the performance of big data processing and study map task scheduling problem in MapReduce. Using the abstraction of underlying networks, we designed an SDN

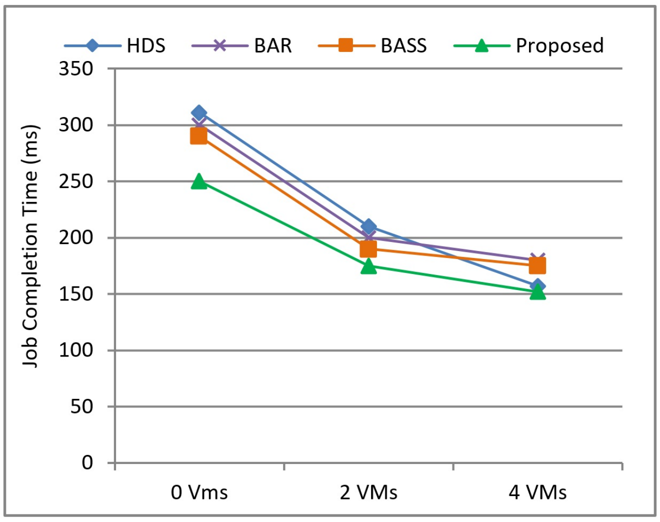

Fig. 15. The changes in job completion time based on the number of VMs.

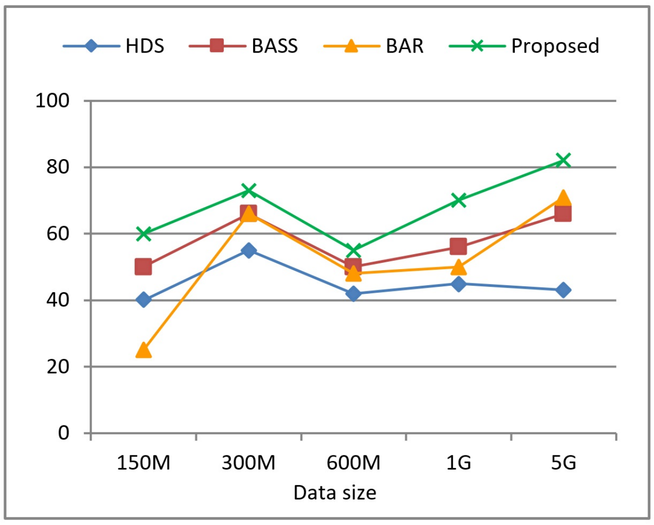

Fig. 16. Data locality ratio (LR) among schedulers, LR = data local task number/total task number.

controller and a problem-solving methodology based on a tabu search algorithm that allocates batches of tasks and gains to high resource utilization of the data center networks. Given the practical interest in this type of problem, the proposed approach opens a way for further progress in this area. We proved that it is relatively easier to deal with high data traffic and failure information via a logically centralized SDNcontroller, even though it imposes some overhead on the controller but the advantages of our approach outweigh this overhead and any flow format of big traffic data with arbitrary granularity can be used for traffic engineering. It is also relatively easier to apply traffic engineering to switches in the network by helping the tables of the OF switches. We have designed our task scheduling algorithm in Hadoop, which runs in Mininet emulator and Floodlight controller and the experimental results demonstrate that our method improves the job completion time and resource utilization compared with recent works.

REFERENCES

[1] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Applicat.,* vol. 19, no. 2, pp. 171–209, Apr. 2014.

[2] Apache Hadoop framework. [Online]. Available: https://hadoop.apache.org/

[3] V. N. Gudivada, R. Baeza-Yates, and V. V. Raghavan, "Big data: Promises and problems," *Comput.*, vol. 48, no. 3, pp. 20–23, Mar. 2015.

[4] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*. vol. 103, no. 1, pp. 14–76, Dec. 2014.

[5] R. Masoudi and A. Ghaffari, "Software defined networks: A survey," *J. Netw. comput. Applicat.,* vol. 67, pp. 1–25, May 2016.

[6] C. Severance, "Roy t. fielding: Understanding the rest style," *Comput.*, vol. 48, no. 6, pp. 7–9, Jun. 2015.

[7] N. McKeown *et al.,* "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Review,* vol. 38, no. 2, pp. 69–74, Mar. 2008.

[8] OS. Version. 1.5. 0 (Protocol version 0x06). Open Networking Foundation. 2015.

[9] S. Jamali, A. Badirzadeh, and MS. Siapoush, "On the use of the genetic programming for balanced load distribution in software-defined networks," *Digit. Commun. Netw.,* vol. 4, no. 4, pp.288–296, Nov. 2019.

[10] H. Hu, Y. Wen, TS. Chua, and X. Li, "Toward scalable systems for big data analytics: A technology tutorial," *IEEE Access*, vol. 2, pp. 652–687, Jun. 2014.

[11] S. Sharma and V. Mangat, "Technology and trends to handle big data: Survey," in *Proc. IEEE ICACCT*, Feb. 2015.

[12] Y. Demchenko, C. De Laat, and P. Membrey, "Defining architecture components of the big data ecosystem," in *Proc. IEEE CTS*, May 2014.

[13] A. Gani, A. Siddiqa, S. Shamshirband, and F. Hanum, "A survey on indexing techniques for big data: Taxonomy and performance evaluation," *Knowl. Inf. Syst.,* vol. 46, no. 2, pp. 241–284, Feb. 2016.

[14] J. Wan *et al.,* "A manufacturing big data solution for active preventive maintenance," *IEEE Trans. Ind. Inform.,*" vol. 13, no. 4, pp. 2039–2047, Feb. 2017.

[15] Hadoop: Fair scheduler. [Online]. Available: https://hadoop.apache.org/docs/r3.3.4/hadoop-yarn/hadoop-yarn-site/FairScheduler.html

[16] Hadoop: Capacity scheduler. [Online]. Available: https://hadoop.apache.org/docs/r3.3.4/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

[17] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "Bar: An efficient data locality driven task scheduling algorithm for cloud computing," in *Proc. IEEE/ACM CCGRID*, May 2011.

[18] Z. Li, Y. Shen, B. Yao, and M. Guo, "OFScheduler: A dynamic network optimizer for MapReduce in heterogeneous cluster," *International J. Parallel Programming*, vol. 43, no. 3, pp. 472–488, Jun. 2015.

[19] L. Cui, FR. Yu, and Q. Yan, "When big data meets software-defined networking: SDN for big data and big data for SDN," *IEEE Netw.*, vol. 30, no. 1, pp. 58–65, Jan. 2016.

[20] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-aware scheduling with SDN in Hadoop: A new trend for big data," *IEEE Syst. J.*, vol. 11, no. 4, pp. 2337–2344, Nov. 2015.

[21] S. Zhao, A. Sydney, and D. Medhi, "Building application-aware network environments using SDN for optimizing Hadoop applications," in *Proc. ACM SIGCOMM.*, Aug. 2016.

[22] IT. Akhthar, "Tenant-aware bigdata scheduling with software-defined networking."

[23] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang. "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Trans. Netw.*. vol. 24, no. 1, pp. 190–203, Nov. 2014.

[24] F. Glover and M. Laguna, *Tabu search. InHandbook of combinatorial optimization*, Springer, Boston, MA, 1998, pp. 2093–2229.

[25] P. Hansen, N. Mladenović, and J. A. Moreno Perez, "Variable neighbourhood search: Methods and applications," *4OR*, vol. 6, no. 4, pp. 319–360, Dec. 2008.

[26] I. Mathlouthi, M. Gendreau, and JY. Potvin, "A metaheuristic based on tabu search for solving a technician routing and scheduling problem," *Comput. Operations Research,* vol. 125, p. 105049, Jan. 2021.

[27] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Proc. IEEE ICDEW*, Mar. 2010.

[28] W. Queiroz, MA. Capretz, and M. Dantas, "An approach for SDN traffic monitoring based on big data techniques," *J. Netw. Comput. Applicat.*, vol. 131, pp. 28–39, Apr. 2019.

[29] AS. Thyagaturu, A. Mercian, MP. McGarry, M. Reisslein, and W. Kellerer. "Software defined optical networks (SDONs): A comprehensive survey," *IEEE Commun. Surveys Tuts.* vol. 18, no. 4, pp.2738–2786, Jul. 2016.

Mina Soltani Siapoush received her M.Sc. degree from the Computer Engineering department of Islamic Azad University, Ardabil, Iran in 2018. Her research interest is dependable systems, big data, and software-defined networking.

Shahram Jamali is an Associate Professor leading the Autonomic Networking Group at the Department of Engineering, University of Mohaghegh Ardabili. He teaches computer networks, network security, computer architecture, and computer systems performance evaluation. Dr. Jamali received his M.Sc. and Ph.D. degree from the Dept. of Computer Engineering, Iran University of Science and Technology in 2001 and 2007, respectively. Since 2008, he is with the Department of Computer Engineering, University of Mohaghegh Ardabil and has published more than 100 conference and journal papers.

Amin Badirzadeh is a Researcher in Software Engineering. He received his M.Sc. degree from Computer Engineering department of Islamic Azad University, Ardabil, Iran in 2017. His research focus is on networking and software-defined networking.