IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 25, 2018, accepted October 13, 2018, date of publication October 17, 2018, date of current version November 9, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2876385

Optimization of Computing and Networking Resources of a Hadoop Cluster Based on Software Defined Network

ALI KHALEEL, (Member, IEEE), AND HAMED AL-RAWESHIDY, (Senior Member, IEEE)
Wireless Networks and Communications Centre, Department of Electronic and Computer Engineering, College of Engineering, Design and Physical Sciences, Brunel University, London UB8 3PH, U.K.

Corresponding author: Ali Khaleel (ali.khaleel@brunel.ac.uk)

This work was supported in part by the Iraqi Ministry of Higher Education and Scientific Research and in part by the University of Diyala.

ABSTRACT In this paper, we discuss some challenges regarding the Hadoop framework. One of the main ones is the computing performance of Hadoop MapReduce jobs in terms of CPU, memory, and hard disk I/O. The networking side of a Hadoop cluster is another challenge, especially for large-scale clusters with many switch devices and computing nodes, such as a data center network. The configurations of Hadoop MapReduce parameters can have a significant impact on the computing performance of a Hadoop cluster. All issues relating to Hadoop MapReduce parameter settings are addressed. Some significant parameters of Hadoop MapReduce are tuned using a novel intelligent technique based on both genetic programming and a genetic Algorithm, with the aim of optimizing the performance of a Hadoop MapReduce job. The Hadoop framework has more than 150 configurations of parameters and hence, setting them manually is not difficult, but also time-consuming. Consequently, the above-mentioned algorithms are used to search for the optimum values of parameter settings. The software-defined network (SDN) is also employed to improve the networking performance of a Hadoop cluster, thus accelerating Hadoop jobs. Experiments have been carried out on two typical applications of Hadoop, including a Word Count Application and Tera Sort application, using 14 virtual machines in both a traditional network and an SDN. The results for the traditional network show that our proposed technique improves MapReduce jobs' performance for 20 GB with the Word Count application by 69.63% and 30.31% when compared to the default and Gunther work, respectively. While for the Tera Sort application, the performance of Hadoop MapReduce is improved by 73.39% and 55.93%, compared with the default and Gunther work, respectively. Moreover, the experimental results in an SDN environment showed that the performance of a Hadoop MapReduce job is further improved due to the advantages of the intelligent and centralized management achieved using it. Another experiment has been conducted to evaluate the performance of Hadoop jobs using a large-scale cluster in a data center network, also based on SDN, with the results revealing that this exceeded the performance of a conventional network.

INDEX TERMS Big Data, data center network, genetic algorithm, genetic programming, Hadoop, MapReduce, parameter settings optimization, shuffling flow, software defined network.

I. INTRODUCTION

Big data is a term that refers to large and complex data sets that cannot be processed, captured, stored or analyzed using traditional tools [1]. These amounts of huge data are generated from different, sources such as social media, sensor devices, the Internet of things, mobile banking amongst many more origins. Furthermore, many governments and commercial organizations are producing large amounts of, data such as financial and banking statements, healthcare providers, high education systems, research centers, the manufacturing sector, insurance companies and the transportation sector. Regarding which, International Data Corporation (IDC) reported that 2,800 Exabyte of data in the world were stored in 2012 and this is expected to reach up to 40,000 Exabyte over the next ten years. For instance, Facebook processes around 500,000 GB every day. The vast amount of data includes both structured, such as relational databases as well as, semi structured and unstructured data, such as texts, videos, images, multimedia, and web pages. These types of huge data with various formats have led to

the coining of the term big data [2]. However, these massive datasets are hard to be processed using traditional tools and current database systems. Hadoop MapReduce is a powerful computing technology tasked with supporting big data applications [3]. Hadoop is an open source framework that enables the implementation of the MapReduce algorithm for data processing purposes. It is scalable, fault-tolerant and able to process massive data sets in parallel. Moreover, large datasets can be distributed across several computing nodes of a Hadoop cluster to achieve better computation resources and power [4]. Hadoop has a complex structure that contains a number of parts that react with each other through several computing devices. Moreover, Hadoop it has more than 150 configuration parameters and recent studies have shown that tuning some of these can have a considerable effect on the performance of a Hadoop job [5], [6]. Because of the black box feature of the Hadoop framework, the tuning of parameters values manually is a challenging task as well as being time consuming. To tackle this issue, genetic algorithms (Gas) for Hadoop have been developed to achieve optimum or near optimum performance of the Hadoop MapReduce parameter settings. However, there are some traffic issues for Hadoop jobs especially in the shuffling phase during the transfer of intermediate output data from the mappers to the reducers. As a consequence, SDN is proposed to alleviate these traffic issues in a Hadoop cluster. We employed SDN for a small Hadoop cluster using 14 virtual machines connected to one physical switch and two open virtual switches. SDN was also used to evaluate the performance of Hadoop jobs in a large scale cluster in a data center network. The major contributions of this paper are as follows.

- Genetic programming is employed to construct a fitness function based on the running of Hadoop job samples that can be considered as CPU or I/O intensive. The interrelations among Hadoop parameters are represented by the constructed fitness function and described mathematically.
- A GA is also used in this work to optimize the configuration parameters of Hadoop. It is applied to the fitness function constructed by the genetic programming to search for the optimum or near optimum settings of the Hadoop parameters.
- For better optimization, SDN is used to improve the performance of Hadoop jobs. The networking aspect of a Hadoop cluster is optimized using SDN by achieving centralized control and agile management. This can improve the performance of a Hadoop job by accelerating the shuffling phase that can be network intensive. The optimized values of the Hadoop parameters are applied in the optimized network to evaluate the performance of a Hadoop job.
- An application-aware networking based on SDN is used for a Hadoop cluster in a data center network to improve further the performance of a Hadoop job by reducing the execution time of the exchanged shuffling flows between nodes during the shuffle phase. An effective routing algorithm based on SDN is proposed to accelerate the shuffling phase of a Hadoop job by allocating efficient paths for each shuffling flow According to the network resources demand of each flow as well as their size and number in the data center network. Accordingly, the proposed work improves the execution time of a Hadoop job. The proposed work also reduces the routing convergence time in the case of any link crashing or failure.

The remaining sections of this paper are organized as follows. Section II presents some related work, whilst in section III, a set of Hadoop MapReduce parameters are introduced. Section IV explains the implementation of genetic programing for building an objective function of the Hadoop MapReduce parameters. The implementation of GA for MapReduce parameter optimization is explained in section V and section VI presents a performance evaluation of the proposed work using a Hadoop cluster in Microsoft azure cloud. Section VII describes and discusses the experimental results of Hadoop jobs in a small cluster in Microsoft azure. Discussion and the experimental results of the small cluster based on SDN are provided in section VIII. Section IX presents and discusses the experimental results for a Hadoop cluster based on SDN in a data centre network and subsequently, the paper is concluded in section X.

II. RELATED WORK

Many ways have been proposed for the automatic tuning of Hadoop MapReduce parameter settings, one of which being PPABS [7] (Profiling and Performance Analysis-Based Self-tuning). In this framework, the Hadoop MapReduce parameter settings are tuned automatically using an analyzer that classifies MapReduce applications into equal classes by modifying k- means ++ clustering and a simulated annealing algorithm. Furthermore, recognizer is also used to classify unknown jobs into one of these equivalent classes. However, PPABS cannot tune parameters of an unknown job not included on these equivalent classes. Another approach, called Gunther, has been proposed for Hadoop configuration parameters optimization using genetic algorithm. However, all MapReduce jobs have to be executed physically to evaluate the objective functions of required parameters, because Gunther does not have an objective function for each of them. Moreover, the execution time for running MapReduce jobs for objective function evaluation is very long [8]. Panacea framework has been proposed to optimize Hadoop applications based on a combination of statistic and trace analysis using a compiler guided tool. It divides the search place into sub places and subsequently performs a search for best values within predetermined ranges [9]. A performance evaluation model of MapReduce is proposed in [10]. This framework correlates performance metrics from different layers in terms of hardware, software, and network. Industrial professionals proposed the Rule-Of-Thumb (ROT), which is merely a common practice for Hadoop parameter settings tuning [11], [12]. In [13] an online performance tuning system for MapReduce

is proposed to monitor the execution of a Hadoop job and it tunes associated performance-tuning parameters based on collected statistics. Reference [14] optimizes MapReduce parameters by proposing profile to collect profiles online during the execution of MapReduce jobs in the cluster. In [15] a self-tuning system for big data analytics, called starfish, is proposed to achieve the best configurations of a Hadoop framework so as to utilize cluster resources better in terms of CPU and memory. Narayan proposed the integration of SDN technology and Hadoop. The main idea of the proposed work is to identify the traffic of Hadoop intermediate data and the background traffic by using the flow rules, subsequently applying different quality of service (QoS) for them. The experimental results of this work showed that the execution time of a MapReduce job went down due to the sufficient amount of bandwidth being allocated for the shuffle traffic. However, this method is only suitable for small scale clusters and not for large ones in a data center network with a large number of switches and servers [16]. The work proposed in [17] presents an application-aware SDN routing scheme for Hadoop to speed up the data shuffling of MapReduce over the network. Another work was proposed in [18] to improve the job completion time. An application-aware network in SDN (AAN-SDN) for Hadoop MapReduce was suggested to provide both underlying networks functions and specific MapReduce forwarding logics. A flexible network framework (FlowComb) was proposed in [19] for big data applications to achieve high bandwidth utilization and fast processing time by predicting the network application transfers. Lin and Liao [20] used an SDN app for a Hadoop cluster to speed up the execution time of MapReduce jobs. The proposed method involved implementing the SDN app in the Hadoop cluster for easy deployment of the flow rules for Hadoop applications. However, only a small cluster with one physical switch was investigated and hence, the impact on the performance of Hadoop jobs in large clusters in a data center network using this method was not assessed.

III. HADOOP MapReduce PARAMETERS SETTINGS

Hadoop is a software platform written in java that enables distributed storage and processing of massive data sets using clusters of computer nodes. It provides large storage of any type of data (structured, semi structured and unstructured data) due to its scalability and fault tolerance. Furthermore, it has more than 150 tuneable parameters that play a vital role on the flexibility of Hadoop MapReduce jobs and some of them have remarkable influence on performance of Hadoop jobs. Table 1 presents the main parameters of Hadoop system that have the most significant impact on the performance of a Hadoop job.

Below further description of the main parameter settings mentioned in the table 1.

1) MapReduce.task.io.sort.mb: During sorting files, amount of buffer memory is required for each merge stream. This amount is determined by this parameter and by default it is set to be 1MB for each merge stream and the total amount is 100 MB.

TABLE 1. The main parameter settings of hadoop framework.

Parameters	Default
MapReduce.task.io.sort.mb	100
MapReduce.task.io.sort.factor	10
Mapred.compress.map.output	false
MapReduce.job.reduces	1
Mapreduce.map.sort.spill.percent	0.80
MapReduce.tasktracker.map.tasks. maximum	2
MapReduce.tasktracker.reduce.tasks. maximum	2
Mapred.job.shuffle.input.buffer.percent	0.70

2) MapReduce.task.io.sort.factor: This parameter determines the required number of merged streams during sorting files process. The default value is set to be 10 as explained in table 1.

3) Mapred.compress.map.output: The output results generated from mappers should be sent to the reducer through the shuffle phase. However, high traffic is generated during the shuffling process especially when the output data of mappers is large. Therefore, the results generated from mappers should be compressed to reduce the overhead in the network during the shuffling process and thus accelerate the hard disk IO.

4) MapReduce.job.reduces: a specific number of map tasks are required to perform the process of MapReduce job in Hadoop cluster. Number of map tasks is specified by this parameter. The default settings of this parameter are assigned to 1. Furthermore, this parameter has a significant effect on Hadoop job performance.

5) Mapreduce.map.sort.spill.percent: the default setting of this parameter is 0.80 which represents the threshold of in memory buffer used in the map process. The data of in memory buffer is spilled to the hard disk once the in memory buffer reaches to 80%.

6) MapReduce.tasktracker.reduce.tasks.maximum : each MapReduce job has several Map and Reduce tasks running simultaneously on each data node in Hadoop cluster by task tracker. Reduce tasks number is determined by this parameter and its default setting is set to be 2. This parameter can have an important impact on the performance of Hadoop cluster when better utilizing the cluster resources in terms of CPU and memory by tuning this parameter to the optimal value.

7) MapReduce.tasktracker.map.tasks.maximum: while number of reduce tasks is determined by parameter 6, this parameter defines number of map tasks running simultaneously on each data node. The default value of this parameter is 2. On the other hand, any change in the default settings of this parameter can have a positive impact on the total time of MapReduce job.

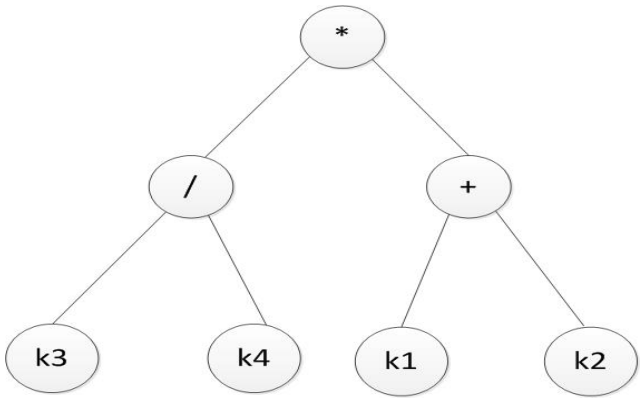

FIGURE 1. An example of a genetic algorithm.

8) MapReduce.reduce.shuffle.input.buffer.percent: the output of mapper during the shuffling process requires a specific amount of memory from the maximum heap size for storage purposes. The percentage of this mount is determined by this parameter and its default value is set to be 0.70.

IV. EVEOLVING HADOOP MapReduce PARAMETERS WITH GENETIC PROGRAMMING

Genetic programming (GP) [16] is a technique used to solve problems automatically with a set of genes and chromosomes. These are evolved using two essential genetic operations: crossover and mutation. In this work, GP is employed to create an objective function of the MapReduce parameters. The parameters of Hadoop MapReduce are represented as (k_1, $k_{2,.....,}k_{n,}$) and here, eight parameters are tuned using a genetic algorithm (GA). An objective function should be built first using GP. Hence, a mathematical expression or function between these parameter settings needs to be determined. GP is used to evolve an expression between these parameters using arithmetic operations ($*$, $+$, $-$, /). The fitness assigned to each parameter during the population process in GP should reflect how closely the output of the mathematical expression (function) for this parameter is to that for the original one. The arithmetic operations in GP are called functions, while the parameters (k_1,...,k_n) are the leaves of the tree, which are also called terminals. The mathematical expressions between the Hadoop MapReduce parameters are determined based on their data type. The mathematical expression should have same input data type and same number of input parameters. After its determination, the completion time of these functions needs to be calculated and compared with the real one. The best mathematical expression among the parameters (k_1,...,k_n) will be selected based on its approximated completion time, which should be very near to the real one. The tree in GP is used to hold both functions and terminals. As mentioned above, arithmetic operations ($*$, $+$, $-$, /) are called functions and (k_1,...,k_n) are called leaves or terminals. Fig. 1 shows an example of the representation of parameters using GP.

Algorithm 1 Genetic Programming

Input: Hadoop MapReduce job samples
Output: Relation between MapReduce parameters
1: For i = 1 to population size do
2: Create chromosome (i) with functions and terminals;
3: Fitness (i) = 0;
4: i++;
5: end for
6: while n < iterations terminated do
7: move chromosome(i) into form of tree(i);
8: for x = 1 to population size do
9: Compute estimated execution time for (x)
10: if difference between estimated and real time < TS THEN
11: fitness (i)++;
12: end if
13: x++;
14: end for
15: x++;
16: end for
17: Compute the fitness (i) of chromosome i
18: If fitness(i)= number of samples then
19: Chromosome(i)= best chromosome;
20: End while
21: If fitness(i)>best fitness value then
22: Chromosome(i)= best chromosome;
23: Fitness(i)= best fitness;
24: End if
25: Use selection, mutation and crossover on chromosome(i);
26: Gen= Gen+1;
27: i++;
28: End for
29: n++;
30: End while
31: Return best chromosome

The figure shows that the function ($*$) has two input arguments, which are ($+$) and (/) and the function ($+$) also has two (k_1, k_2). The completion time of MapReduce job of Hadoop parameters can be represented as $f(k_1, k_2, .., k_n)$. The approximated completion time of Hadoop MapReduce job represents the evolved function that will be compared to the real completion time of Hadoop MapReduce that pertains to the target function. According to [16], the approximated completion time of Hadoop MapReduce (evolved function) should be very near to the real completion time of the job (target problem or function). Algorithm 1 shows the procedures of GP.

In this work, a list of MapReduce jobs is used as input datasets and a large number of experiments was run for both Word count and Tera sort applications, being used to process different sizes of these input datasets, as presented in section

VII. The implementation of GP is performed to find all possible expressions between the Hadoop MapReduce parameters by generating hundreds of chromosomes and in this work, 600 were initially generated. All linear chromosomes are represented into form of graph tree and the fitness value of each is calculated based on the completion time of a Hadoop MapReduce job for each training dataset. The completion time of a Hadoop MapReduce job $f(k_1, k_2, .., k_n)$ for training datasets generated from genetic chromosomes is compared with the real completion time of the Hadoop MapReduce job. The difference between the approximated and real completion time of the Hadoop MapReduce job should not be more than 40s, which is referred as TS. The chromosome with the high fitness value is selected. The measure of fitness value is the same as the number of Hadoop MapReduce job used in this process. This measure is supposed based on the example of soccer player to test the fitness in [17]. The evolution process will terminate once the best fitness value is obtained, i.e. when reaches to the number of Hadoop MapReduce jobs used in the process. Moreover, genetic selections and operators are applied, such as mutation and crossover, to produce new chromosomes and update the current ones. The expression between the parameters is obtained after 40,000 iterations. Equation1 below represents the mathematical expression and the relation between the Hadoop MapReduce parameters, which is used as an objective function in the next algorithm (GA).

$$f(k_1, k_2, \ldots, k_8) = (k_3 + k_7) * (k_5/k_2) + (k_1 * k_6) - (k_4 + k_8) \quad (1)$$

V. HADOOP MapReduce PARAMETER SETTINGS TUNING USING A GENETIC ALGORITHM

A genetic algorithm (GA) is a metaheuristic one, which belongs to the group of evolutionary algorithms (EA) and was first proposed by John Holland to provide better solutions to complex problems. GAs are widely used to solve many optimization problems based on natural evolution processes. They work with a set of artificial chromosomes that represent possible solutions to a particular problem. Each chromosome has a fitness value that evaluates its quality as a good solution to the given problem [18]. GAs start with generating a random population of chromosomes. A set of essential genetic operations, such as crossover, mutation and update are applied on the chromosome to perform recombination and selection processes on solutions for specific problem. The selection process of chromosomes is performed based on their fitness value. The chromosome with high fitness has the chance to be chosen and create an offspring to generate the next population [19]. Algorithm 2 describes the procedure for GA implementation, where the equation 1 generated from GP is used as an objective function that needs to be minimized, which is expressed as:

$$f(k_1, k_2, .., k_n) = (k_3 + k_7) * (k_5/k_2) + (k_1 * k_6) - (k_4 + k_8)$$

Algorithm 2 Genetic Algorithm

Input: Data sets (MB)
Output: Optimised Hadoop MapReduce parameters
1: GA_process ( ) { **2**: Gen = 0
3: P= Intial_population ( );
4: fitness = evaluate_population ( ) ;
5: repeat {
6: repeat {
7: Use selection, crossover and mutation on population;
8: finess = evaluate_population ( ) ;
9: Gen = Gen+1;
10 :} until fitness (i) = bestfitness, $1 \leq i \leq$ popsize or generation $\geq$ iteration number;
}
}

In algorithm 2, an initial population of chromosomes is randomly generated and each MapReduce parameter is represented as one of these. It means that chromosome(i) = *k1,k2,..,kn*, where *n* is the number of parameters. As aforementioned, in this work, there are eight parameters that need to be tuned. After the generation of the population, the fitness value of each chromosome in it is evaluated based on the objective function *f(k1, k2,.., kn)*. The chromosome with high fitness is selected and genetic operators, which are selection, crossover and mutation, are applied to update the current population and generate a new one. The procedures are repeated until the best fitness values of chromosomes, which represent the optimized MapReduce parameters, are obtained or the number of iterations is finished. In this algorithm, 15 chromosomes are used as a population size and the number of iterations set to be 100. Furthermore, the probability of crossover P_c=0.2 and the probability of mutation P_m=0.1 are empirically determined and used as genetic operators. Roulette wheel spinning is employed as a selection process. The ranges and recommended values of the eight Hadoop MapReduce parameters are presented in table 2.

VI. PERFORMANCE EVALUATION ENVIRONMENT

The proposed work was implemented and evaluated using eight virtual machines (VMs) of a Hadoop cluster placed on Microsoft azure cloud. Each VM was assigned with 8 GB memory, 4 CPU cores and 320 GB storage for the whole cluster. Hadoop Cloudera (Hadoop 2.6.0-cdh5.9.0) was installed on all nodes, with one being configured as a master and the rest as slaves. The master node could also be run as a slave. For fault-tolerance purposes, we set the replication factor of the data block at 3 and the HDFS block size was 128 MB. Table 3 presents the specifications of the Hadoop cluster.

VII. EXPERIMENTAL RESULTS

Both the Word Count and Tera sort applications have been run as real job programs for Hadoop MapReduce framework to evaluate the performance of our proposed work on a Hadoop cluster. It can be clearly observed that there is a difference

TABLE 2. Hadoop mapreduce parameters recommended from the genetic algorithm.

Hadoop MapReduce parameters	Range	Parameters name
K_1	100-165	MapReduce.task.io.sort.mb
K_2	10-160	MapReduce.task.io.sort.factor
K_3	True	Mapred.compress.map.output
K_4	1-16	MapReduce.job.reduces
K_5	0.60-0.80	MapReduce.task.io.sort.spill. percent
K_6	2-4	MapReduce.tasktracker.map.t asks. maximum
K_7	2-4	MapReduce.tasktracker.reduc e.tasks. maximum
K_8	0.70-0.71	MapReduce.reduce.shuffle.in put.buffer.percent

TABLE 3. Hadoop cluster setup.

Intel Xeon X5550 server1 and uxisvm04 server 2	CPU	4 cores for each VM
	Processor	2.27 GHz
	Hard disk	360 GB
	Connectivity	1 GBit Ethernet LAN interconnectivity between two servers
	memory	64 GB
Operating System	Host Operating System	Microsoft windows server 2012 R2
	Guest Operating System	Ubuntu 14.04.4 LTS (GNU/Linux 4.2.0-27-generic x86_64)

among the tuned configurations of the Hadoop MapReduce parameter settings using our proposed system, the default one and Gunther's method. For instance, figure 2 shows that when the value of *io.sort.mb* increases, this leads to a decrease in the execution time of the Hadoop MapReduce job. Moreover, the *io-sort-factor* parameter defines the number of data streams to merge during the sorting of files. From figure 3, it can be clearly seen that when the value of this parameter goes up, the execution time of the job goes down. It can also be observed from figure 4 that when the number of reduce tasks is increased from 5 to 10, the execution time of the Hadoop MapReduce job decreases. However, increasing the number of reduce task results in longer execution time due to the overhead of network resources as well as over utilization of

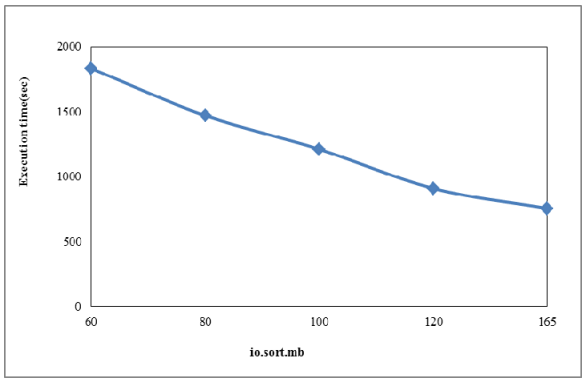

FIGURE 2. The effect of the *io.sort.mb* parameter.

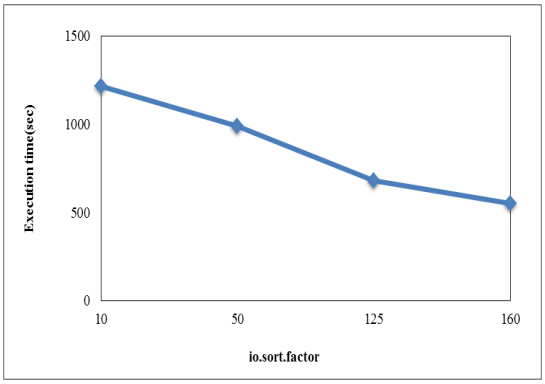

FIGURE 3. The effect of io.sort.factor.

computing resources, such as CPU and memory. Moreover, it is evident that any further increase in reduce tasks leads to the generation of high network traffic and consequently, an increase the overall time of the Hadoop job. Figure 5 shows that increase in the slots of map and reduce can play crucial role for better utilization of cluster resources and accordingly minimize the overall time. One slot has been configured per CPU core, in the cluster setup 4 cores has been allocated for each cluster node and therefore 4 slots has been employed to maximize the utilization of CPU. If additional slots are included in the setup, this exhausts the CPU and results in a delay in the processing time of the MapReduce job. Figure 6 shows the completion time of MapReduce jobs for different sizes of datasets by applying a compression parameter. It is observed that applying this parameter by switching its Boolean value empirically from false to true can reduce the completion time of a MapReduce job by alleviating the traffic consumption of the network and reducing the pressure on the I/O operation. However, the compression of input data and reduce output data is not available in some applications such as Tera sort. Moreover, the performance of this parameter is reduced when massive datasets are used such as, 40 or 50 GB. The reason for this is that any increase in dataset size leads to the generation of high volumes of shuffling traffic, especially in a static IP network environment. As a result, a software defined network is implemented on a Hadoop cluster to

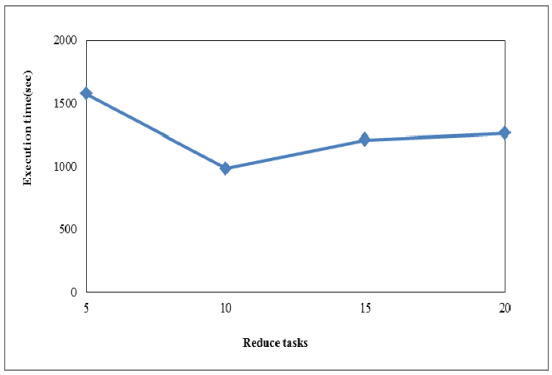

FIGURE 4. Reduce tasks influence.

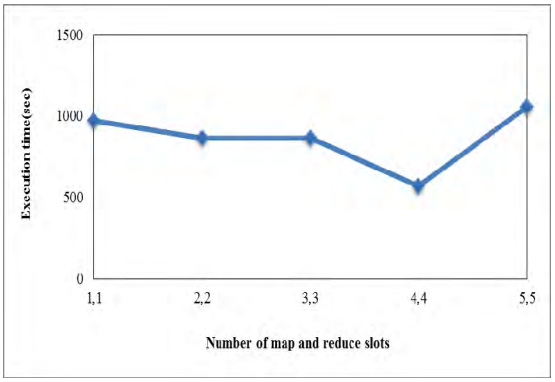

FIGURE 5. Map and reduce slots influence on MapReduce job.

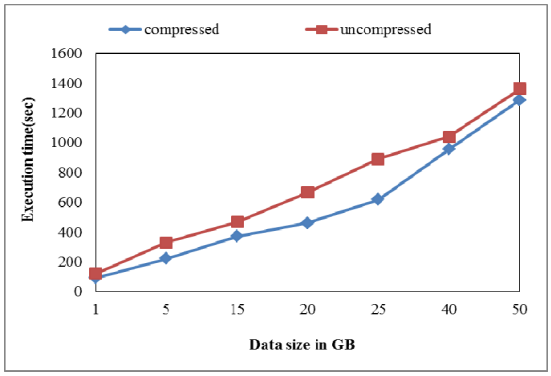

FIGURE 6. The influence of compression parameter.

reduce the shuffling traffic generated from a MapReduce job. The following section describes the implementation of a Hadoop cluster based on SDN.

Table 4 shows the optimized values of the Hadoop MapReduce parameters for each size of dataset on eight virtual machines. To show the performance of our method, different sizes of data, including 1 GB, 10 GB and 20 GB, were generated. The tuned parameters were used for both the Word Count and Tera sort applications. The execution time of both the word count and Tera sort applications based on the tuned settings by our proposed method is compared with the execution time of the two applications based on the default setting

TABLE 4. Hadoop mapreduce parameter settings recommended by a genetic algorithm on eight virtual machines.

Name	Default	Optimised Values using Genetic algorithms		
		1GB	10GB	20GB
mapreduce.task.io.sort.mb	100	100	140	165
mapreduce.task.io.sort. factor	10	50	125	160
mapred.compress.map. output	false	True	True	True
mapreduce.job.reduces	1	16	10	10
mapreduce.map.sort.spill.percent	0.80	0.87	0.68	0.77
mapreduce.tasktracker.map .tasks.maximum	2	4	4	4
mapreduce.tasktracker. reduce.tasks.maximum	2	4	3	4
mapreduce.reduce.shuffle .input.buffer.percent	0.70	0.70	0.71	0.71

as well as the settings achieved by Gunther. Both Word count and Tera sort were run twice and it emerged that our proposed method can improve the performance of a MapReduce job in a Hadoop cluster, most notably with large input data sizes. Figure 7 and figure 8 show the completion time of a Hadoop MapReduce job using the proposed method in comparison with the default one and Gunther's method. From figure 7, it can be observed that the performance of the Hadoop Word Count Application is improved using the proposed approach by 63.15% and 51.16% for the 1 GB dataset when compared with the default and Gunther's settings, respectively. Furthermore, the experiments carried out on a 10 GB dataset show that our proposed method improves the performance of the Word Count Application by 69% and 37.93% when compared with the default and Gunther's method, respectively. Finally, the proposed method also achieved better performance than the default and Gunther settings on the Word Count application by 69.62% and 30.31%, respectively, for 20 GB.

From figure 8, it can be clearly seen that our proposed method improved the Tera Sort application performance by 52.72% over the default system and 44.28% when compared to the Gunther settings for 1GB. For 10 GB, the performance was improved by 55.17% as compared to the default one and was 51.25% better than with Gunther's method. Finally, Tera Sort application performance for 20 GB was improved by 73.39 % and 55.93 % more than the default and Gunther settings, respectively.

VIII. A HADOOP CLUSTER BASED ON SDN

Software defined networking (SDN) [20] is an emerging technology that provides agile and dynamic management for

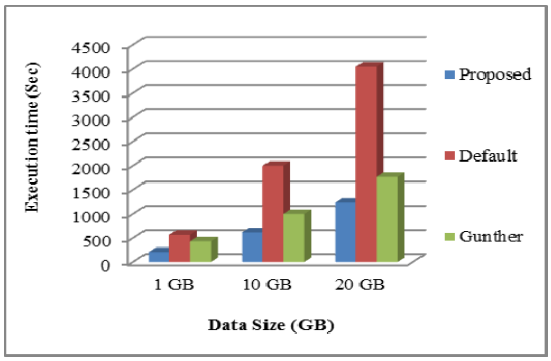

FIGURE 7. Comparison of word count application.

FIGURE 8. Comparison of tera sort application.

the network through central and intelligent programming. In this novel technology, the control plane is decoupled from the data plane to provide more flexibility and agility, which leads to network performance improvement by obtaining better routing decisions. The controller communicates with the OpenFlow switch through the OpenFlow protocol. In this work, SDN is implemented to improve the performance of Hadoop networking by efficient utilization of bandwidth for shuffling traffic. Different sorts of traffic are generated from a Hadoop cluster, such as shuffle phase traffic, HDFS data transfer, HDFS read and write along with Hadoop monitoring messages. It is worth noting that the shuffling traffic represents the most traffic produced by both Word Count and Tera Sort in a Hadoop cluster followed by HDFS read and write. In the proposed system, SDN is employed with OpenVswitch to allocate more bandwidth for the traffic generated by the shuffling phase when the mapper transfers its output to the reducer. However, identifying the network resources of shuffling traffic is a challenging task, because the core framework of Hadoop does not include sufficient information regarding network resources demand for this traffic. A Hadoop cluster has a single job tracker and several task trackers. The progress of Hadoop jobs is monitored by the job tracker, whilst each task tracker sends heartbeat messages to the job tracker about its status. However, these messages lack sufficient information about the network resources. To address this, our proposed system installs software engines on each Hadoop host to record the required information of network resources for

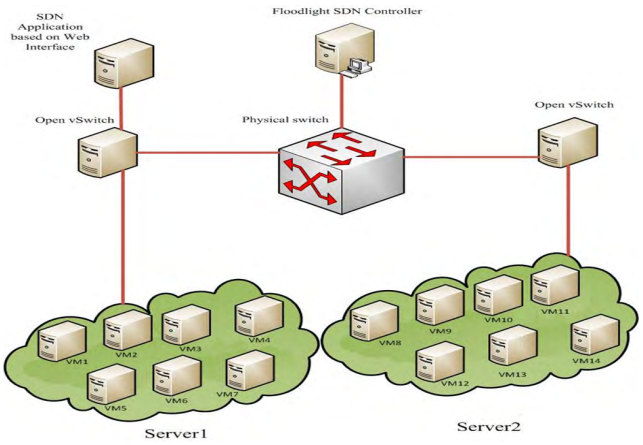

FIGURE 9. Small scale hadoop cluster in an SDN environment.

each shuffling flow. This information contains the size of map output data (intermediate data) being transferred over each flow to the reducers. Furthermore, software engines determine the required network bandwidth for each shuffling flow and record sufficient information, such as the IP address of the source and destination nodes as well as the size of each flow. Then, all the required information is delivered to the SDN controller to assign an efficient bandwidth for shuffling flows. The SDN controller installs flow entry in each Open vSwitch for each shuffling flow and moves the shuffling flows to a queue with higher bandwidth. On the other hand, flow rules are installed in Open vSwitch for other types of traffic, such as control messages and HDFS read/write, to switch them to another queue with low bandwidth allocation. The TCP communication between the task trackers to send the map output data in a Hadoop cluster is performed using port 50060. Open vSwitch matches the incoming packets to identify them by their port number. In the proposed system, 14 virtual machines, installed on two servers, were used with two packages of Open vSwitch installed on two PCs, with one floodlight SDN controller being installed on one PC. SDN application was also installed on one PC. The two servers were connected to two open virtual switches, which are connected to a single physical switch with 1GB link capacity. Figure 9 shows the proposed cluster based on an SDN environment. Both Word Count and Tera Sort applications were used to evaluate our proposed system using SDN technology. The experimental results show that our proposed system based on an SDN environment improves the performance of the Word Count application by reducing the completion time up to 12.4% for 30 GB when compared to a TCP/IP environment. Moreover, this rises to 21.9% for 40 GB, while for 50 GB, the completion time is reduced by 32.8% when compared to a TCP/IP Hadoop cluster, as shown in figure10. Figure11 shows the performance for the Tera Sort application using the proposed system for different data sizes ranging from 30-50 GB. It emerges that the proposed system reduces the completion time of Tera Sort for 30 GB on

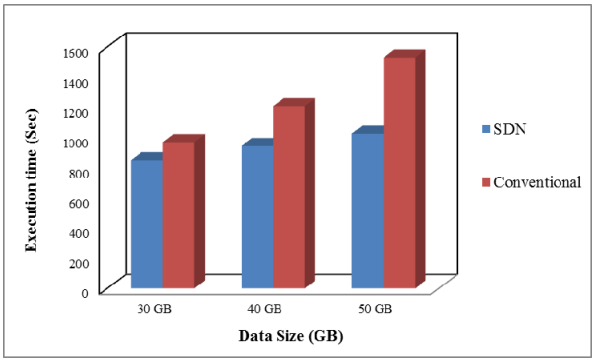

FIGURE 10. Word count performance in an SDN environment.

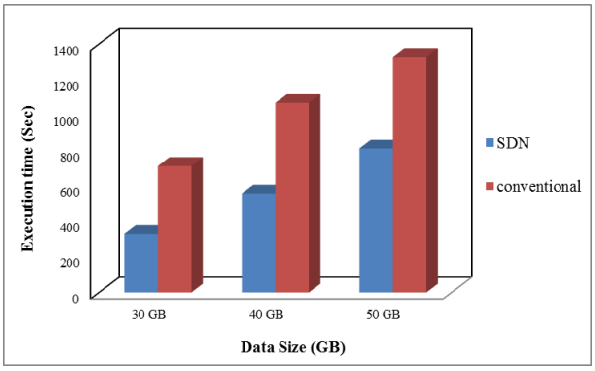

FIGURE 11. Tera sort performance in an SDN environment.

average by 53%. Furthermore, the completion time for 40 GB and 50 GB is reduced by 48.1 % and 38.7%, respectively, over a TCP/IP environment. It is worth noting that performance of Tera Sort application decreases with larger data sizes due to the high volume of shuffling traffic that is generated form these jobs.

IX. A HADOOP CLUSTER BASED ON SDN IN A DATA CENTER NETWORK

We expanded our set up to be implemented in a data center network with a large scale Hadoop cluster with many switches and computing nodes. The advantage of using many switches is the capacity for utilization of bisection network bandwidth. We employed SDN in the data center network to achieve intelligent services and agile network management. Furthermore, we used large the Hadoop cluster with different sizes of network topology to measure the convergence routing. The following section explains some adopted routing techniques and network topologies in a Data center network.

A. ROUTING TECHNIQUES AND NETWORK TOPOLOGIES

Before the discussion of our proposed work, it is important to explain some routing techniques, like ECMP and some data center network topologies. Multipath techniques are widely used in the modern data center network for forwarding and distributing flows across multiple paths so as to achieve better bandwidth utilization. ECMP is used to distribute the flows across multiple equal cost paths to exploit the full capacity of network bandwidth. However, it has some limitations, such as the static scheduling of flows across multiple paths. That is, it uses a hashing value policy to allocate flows with certain paths. It also lacks a global view of the entire network, missing its current load as well as the individual characteristics of flows and their future network demand. As a result, we propose in this paper an effective routing algorithm based on application level information to estimate the demand of all shuffling flows during the MapReduce process, as explained in section B.

The characteristics of the most popular topologies of three-tier architectures, like fat tree topology, have been studied. From this study, we have identified some limitations and bottlenecks of this topology. Fat tree topology is divided into multiple pods, with each including the switches of the edge and aggregation layers. The connection inside the pod is considered as a local pod connection, because the traffic remains inside it. On the other hand, the connection between different pods is considered as a remote connection, because the traffic of connecting pod passes through one or more core switches. This hierarchical architecture limits the locations of end hosts and also creates loops in the network due to the redundant paths that connect the end hosts when multipath techniques are used, like ECMP. As a result, a spanning tree is used to prevent loops by selecting a single path and disabling all other redundant paths. However, this routing scheme of a spanning tree leads to poor network utilization, because the flows in the data center network will employ few paths and leave others redundant, only reutilizing them in the case of any outage or failure. We illustrate some examples of flow transfer based on fat tree topology.

There are three cases of transfer flows between two hosts in a data centre based on fat tree topology. The first, involves sending shuffling flow from host 1 to host 2. In this case, there is only a single path between them, because both hosts are located in the same rack. Hence, all possible paths between the two hosts go through edge switches only and the generated traffic remains inside the rack, with there being no need to traverse any aggregation switches. In the second case, host 1 sends its flow to host 4, which is located in a different rack, but within the same pod. In this case, the connection between them is an intra-pod connection, because all the possible paths between these two hosts will pass through edge and aggregation switches. The third case is in relation to transferring flows between two hosts located in different pods, such as host 2 and host 8. In this case, there are multiple paths between them to transfer flows. However, the produced traffic between the two hosts has to traverse edge, aggregation and core switches, because each host is located in a different pod and all possible paths should go through different core switches. The situation becomes more sophisticated when some hosts in different pods exchange shuffling flows at the same time and might contend for the same links, especially in the aggregation and core switches, thus creating congestion

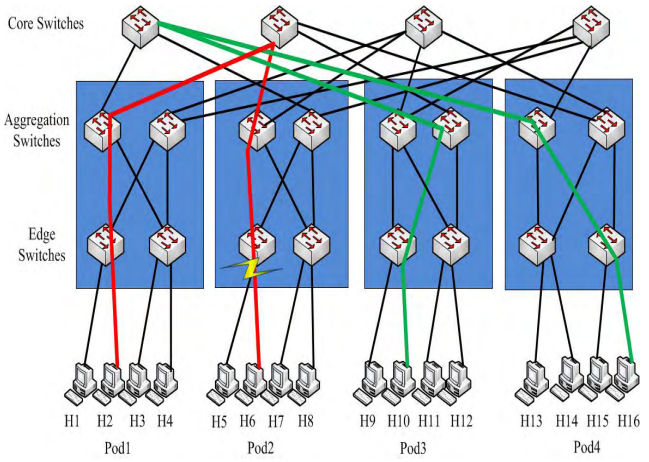

FIGURE 12. Path allocations challenging in fat tree topology.

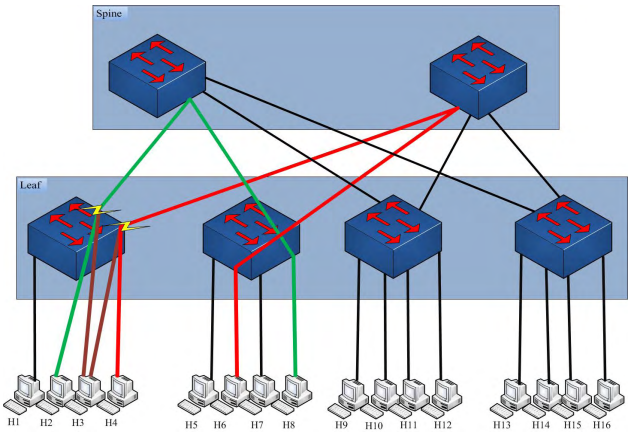

FIGURE 13. Path allocation using ECMP in leaf-spine topology.

that makes the bandwidth utilization of the core and aggregation links becoming over-utilized.

It is supposed that multiple hosts exchange their flows at the same time. Specifically, host 2 sends its flow to host 6, host 10 sends it flow to host 16 and host 5 sends its flow to hosts 15, respectively, all simultaneously. Fig. 12 illustrates the path between hosts 2 and 6 as well as that between hosts 10 and 16 in bold lines. It is observed that there are multiple paths between all the hosts. However, it is noted that there is a challenge to assign even a single path among the multiple paths in the data center network for hosts 5 and 15 because of the congestion that has occurred in the network. The main cause of this is the architecture of fat tree topology that constrains the location of end hosts. Since host 5 is located in pod 2 and host 15 is in pod 4, it is a challenging task to assign a path between the two hosts even though we selected the right side of pod 2 to avoid the overlapping. It is impossible to avoid the overlapping in pod 4, because the right side in pod 2 can only reach the left side of pod 4 and consequently, this creates congestion between the two hosts. As a result, it has become crucial to design an efficient type of data center architecture, like leaf-spine topology. Unlike fat tree topology, this consists of two layers. The first is the leaf layer that includes several switches connected to end hosts in the network. It is connected to the spine layer that represents the second or top layer. Leaf-spine topology is widely adopted in large data centers and cloud networks due to its remarkable features, such as scalability, reliability and effective performance. However, applying multipath algorithms, such as ECMP, as a forwarding technique for shuffling flows to utilize more bandwidth in the leaf-spine topology is not an effective way, because it is a static scheduling algorithm and it does not consider the network utilization or flow size. For instance, there are three different hosts in the same rack, which are connected to the same switch in the leaf layer transferring their flows to other hosts in different racks. The first case, is when host 2 sends its shuffling flow to host 8, whilst the second, is when host 4 sends its shuffling flow to host 6 and the third case is when host 3 transfers his shuffling flows to host 10, as shown in fig.13. We observed that host 3 might compete for the same heavy loaded link in the leaf switch, because of the allocation technique of ECMP, whereby it might choose the same heavy loaded link for two large shuffling flows, thus resulting in a congestion and collision. The reason for this, is because, as aforementioned, ECMP lacks a global view of entire network. Moreover, with ECMP algorithm, the flow is routed based on its hash value. Hence, flows might result in using the same path and creating congestion in some links in the leaf and spine switches. It is also seen in fig.13, that all possible paths of shuffling flows for all cases might compete for the same leaf and spine switches, which leads to overload on some link switches. Furthermore, crashing or failure might occur on some links that belong to the allocated path for shuffling flows in the leaf and spine switches. As a consequence, we propose an effective routing algorithm based on SDN that performs the routing process, which respects the network resources demand of each shuffling flow as well as their size and number. The proposed algorithm is also able to reroute the shuffling flows to another available path in the case of any failure or crashing on any link in the network. The proposed algorithm is explained in the following section.

B. THE IMPLEMENTATION OF THE PROPOSED METHOD FOR A HADOOP CLUSTER IN A DATA CENTRE NETWORK

Our proposed work consists of three modules as follows.

1- Link monitor module: This module monitors network link status, such as link loading in the network and computes the link weight. It periodically gets the statistics information of all links loaded in the data center network from all the connected OpenFlow switches at specific intervals. Statistics such as per-table, per-flow and per-port are collected and stored as snapshots. All switches in the network are connected to the SDN control. However, the SDN controller lacks the required information of all links between the switches and hence, a link layer discovery protocol (LLDP) [16] is used to identify the needed information of all links and the switches

layer in the network topology. Statistic information about links loading is used by the routing module to calculate the paths accordingly. The current load of each link in the data center network is computed by using N transmitted bytes from the port within recent interval t over the bandwidth (B) of the link. The formula below calculates the current load of the link:

$$L_{Lk} = \frac{total\ number\ of\ transmitted\ bytes\ within\ t}{B} \quad (2)$$

It is supposed that all links have the same bandwidth and each has a fixed weight (W), in this case it is set to 1. It is very important to check whether the current load of each link (L_{Lk}) reaches or does not reach the peak of link depending on the link weight (W) by comparing it with (L_{Lk}). If $L_{Lk}<1$, it means that is has not yet reached the peak of the link. However, if $L_k=1$, it means that it has and this may cause link overloading, because of some heavier flows and consequently, result in improper path allocation. Hence, the weight of each link should be estimated based on the number of flows and the throughput of each. The natural demand of shuffling flows is estimated by the Hadoop engine module. It is worth noting that the current load reaches the link capacity, if it exceeds threshold γ which has been set to be 90% of the link capacity. Furthermore, we compute the path load for all flow paths in the leaf and spine switches by using the maximum load of each link, which belongs to the path as explained in the equation below.

$$L_p = max_{l \in p} l_l \quad (3)$$

Where, (p) is defined as the path used to route the shuffling flow from source to destination. Each link that belongs to the path (p) is represented by (l) and (l_l) pertains to the load of each link that traverses the path at the leaf and spine switches from the source node to the destination node. Once the path load of each link is computed, all information is delivered to the scheduling and routing component to select the convenient path that has the least path load (L_p). Then, it installs the flow entries into a set of switches of the selected path.

2- Hadoop monitor engine: In a Hadoop cluster environment when the map task in the mapper node writes its output data to the reducer node, shuffling traffic is generated during the shuffle phase of a Hadoop job. This traffic needs sufficient network bandwidth to accelerate the processing time of the Hadoop job. However, the main Hadoop framework does not contain sufficient information about the required network resources. Therefore, this module is proposed to identify the data transferred from the mapper node to the reducer node in a Hadoop cluster during the shuffle phase of a Hadoop job. The data is transferred through a number of flows during the shuffling phase. This module is responsible for recording all the required information of these flows from all the connected Hadoop servers. In a Hadoop cluster, as aforementioned, there is one job tracker and several task trackers. The job tracker is responsible for monitoring the progress of Hadoop jobs by receiving heartbeat messages from each task tracker, but these messages do not include information about the network resources. To obtain such information, a software engine has been installed on each Hadoop server. This engine detects when a map task has finished and starts to send its shuffling information to the reducers, whilst then recording the size of the map output data, which is transferred over the flow to the other reducers. After this process, the Hadoop engine will obtain the required network bandwidth for each shuffling flow. It maintains a table that contains all shuffling flows with their networking demands. Furthermore, all the collected shuffling information includes the source IP address, destination IP address and the size of each shuffling flow. The Hadoop monitor engine also determines the total amount of shuffled data and the number of shuffling flows transferred over each link. All information about shuffling flows is delivered by the Hadoop monitor to the scheduling and routing module to assign proper paths, according to the bandwidth needed for each shuffling flow and the current load of link utilization.

3- Scheduling and routing module: In the forwarding module of the OpenFlow floodlight controller, a packet-in message is generated to notify the controller that new flows have arrived at an OpenFlow switch. The switch checks the packet and if there is no match with its flow entries, the packet is forwarded to the controller. On the other hand, a flow-removed message is also generated when a flow expires in an open flow switch. In this work, we propose a scheduling and routing module to assign efficient paths for the exchangeable shuffling flows between different hosts in the data center network. This module performs the scheduling and routing of the shuffling flows on the chosen paths and it has two tasks. The first is the calculation of the possible paths based on the statistics from the link monitor module that includes the loads on all links in the network. It also uses the collected information by the Hadoop monitor engine to compute the possible paths of different shuffling flows. The collected information by the Hadoop engine module contains a list of shuffling flows including source/destination IPs, flow size and transfer volume over each link. All this information is recorded in a network table to be used for the calculation of the path load in the routing process. This table also contains scheduled flows and available capacity for each of them. Once all the information of shuffling flows has been received by the scheduling and routing module, it will compute the possible paths with low load based on the information collected from the link monitor module and Hadoop monitor engine. The second task is to assign efficiently the best possible paths for all shuffling flows, according to the bandwidth needed for each flow. We propose a scheduling and routing algorithm based on SDN to obtain an effective routing technique for shuffling flow, according to network utilization and flow size by computing the current load of all possible paths in the leaf and spine switches. Once the current load is determined according to equation 2, the shuffling flows are routed onto the proper paths. Our proposed work moves the large shuffling flows

Algorithm 3 Scheduling and routing algorithm

1: **For** each shuffling flow (*SF*) **do**
2: Collect *SF* size and its network resources demand from the SDN controller
3: Compute the current load of all possible paths for each *SF* according to equation3
4: Compare the size of each *SF* with the current load of all possible paths
5: Choose the shortest available path for *SF*and check
6: **If** the link of shortest path is active and its current load does not override the pre-defined threshold **then**
7: Keep *SF* routing on this path;
8: **Else**
9: **If** there is any failure in the link of the shortest path or its load exceeds the pre-defined threshold **then**
10: Choose another available path with light loaded or unused links calculated by equation3;
11: Re-route the shuffling flow on new chosen path;
12: **End if**
13: **End if**
14: **End for**

from heavy loaded links to lightly loaded ones so as to prevent congestion. What is proposed is demonstrated in Algorithm 3.

In this algorithm, we determine the size of the shuffling flow and the demand of the network resources using the Hadoop engine module. This module sends all the required information to the SDN controller. After that, the current load of all possible paths of each shuffling flow between any two hosts is computed using the information received from both the link monitor and Hadoop engine model, as mentioned before. Then, the shortest path with minimum load will be chosen. If the link of the shortest path is active and there is no failure or congestion, the routing of the shuffling flow is kept on this path. However, if there is any crash in the link or its current load exceeds the pre-specified threshold, which is set to 90% of the link capacity of this path, as mentioned for the link monitor module, then another unused or light loaded shortest path should be chosen. This is also computed based on equation 3 and the information received from the Hadoop engine and the shuffling flow is rerouted accordingly. It is worth noting that the SDN controller receives all the required information of link loading for all the Open vSwitches in different layers from the link monitor module, as detailed above.

X. EXPERIMENTAL RESULTS AND DISCUSSION

Two experiments were carried out on a Hadoop cluster based on SDN in the data center network to evaluate the performance of the proposed work. In the first experiment, we used EstiNet emulator software to build two different topologies: fat tree and leaf-spine topology. In both SDN and conventional networks, three layers of switches were used for fat tree topology. The first was the edge layer, which

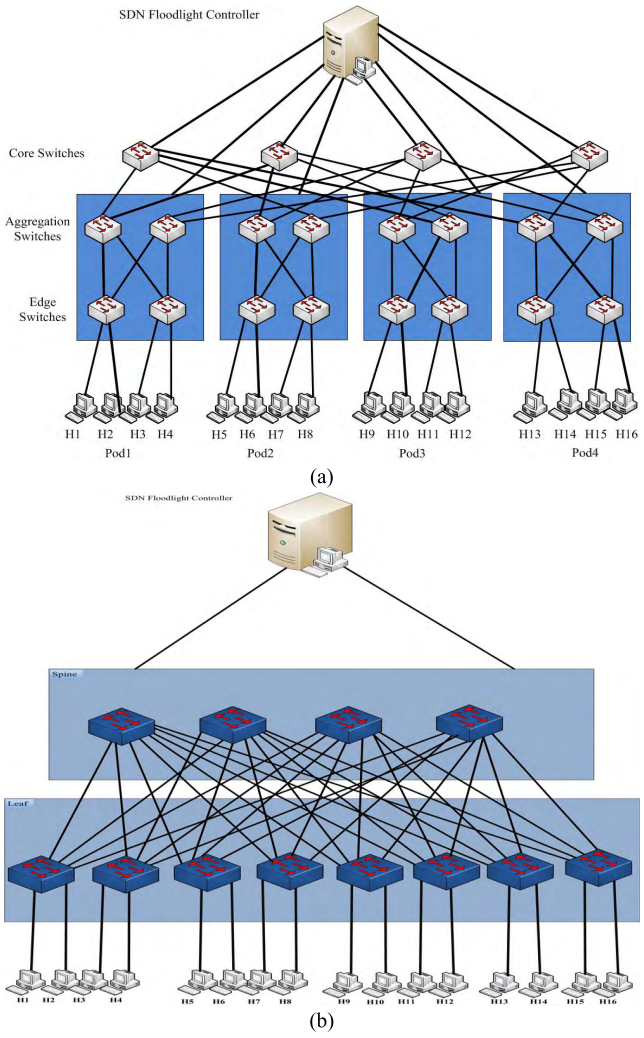

FIGURE 14. (a) Fat tree topology with 20 switches based on SDN. (b) Leaf-spine topology with 12 switches based on SDN.

was assigned with eight switches at the top of the rack. The middle layer or aggregation layer was also allocated eight switches and finally, four switches were used for the core layer. The emulated leaf-spine topology consisted of two layers, with the bandwidth of all the links in the SDN and conventional network being set at 10 Mbps, whilst the link delay was 1ms. We used 16 Hadoop nodes, with each being allocated four CPU cores and 8GB of RAM. All the Hadoop hosts were connected to the emulated fat tree and leaf-spine topology using EstiNet emulator software. The traffic produced by each Hadoop host went into the emulated network. We used the previously utilized two real application programs, namely Word Count and Tera Sort to evaluate the work performance. All switches in the emulated fat tree and leaf-spine topology were connected to the SDN (Floodlight) controller using a TCP connection. Another TCP connection was deployed to connect the floodlight controller to the Hadoop engine. The fat tree topology of data center network based on SDN was compared with the conventional network

to evaluate the performance of Hadoop MapReduce jobs. The leaf-spine topology based on SDN was also compared with the conventional network. Open shortest path first (OSPF) was used for the conventional network. Figure14a shows the fat tree topology of the proposed work based on SDN using 20 switches. The leaf-spine topology with 12 switches is shown in figure14b.The second experiment also involved the same software emulator in the first experiment, but with different network topology size, as shown in figure15a. In both the SDN and conventional networks, we used eight switches in the edge and aggregation layers and only two in the core layer. On the other hand, six switches were used in the leaf-spine topology for the SDN and conventional network, as shown in figure15b. The evaluation of the proposed method using both types of network was made based on the routing convergence time in the case of link failure. We also ran the Word Count and Tera Sort applications to evaluate the performance of Hadoop jobs under different network topology sizes for data center network.

To evaluate the routing convergence time in the case of link failure, we proposed that the failure is occurred in any link of all possible paths specified for each shuffling flow in both topologies. The routing change of the packets in the conventional network needs some time, because any change or update in link status and routing computation has to be performed by each router in the entire network. While in SDN, the controller is the brain of the entire network management and maintains the routing process of the whole network in a centralized manner. We used floodlight controller in SDN to manage and maintain the status of all links in the data center network using the link layer discovery protocol (LLDP), whilst the information of the network topology was maintained by the topology service responsible for calculating the routing computation. In the conventional network, the routing module uses the flooding method to transmit the information of link status to other routers in the data center network in a distributed manner. Two experiments were conducted to evaluate the convergence time of the routing process. As can be seen in figure16, the convergence time of the routing process for different sizes of topology is minimized using the SDN network for the leaf-spine topology, which is not the case with the conventional network. The reason for this, is because the convergence process in the SDN network is more flexible and faster than with the conventional network. The convergence process of the latter depends on the routers, whereby each maintains a routing table which forwards and queries each packet in the network using a specific path. When any change or update occurs in the routing process of packets, like link failure, router 1 will send its update to its neighbor router 2 that will check for any required changes or updates in its routing table, then sending its update to its neighbor and so on. This is means that the changes and updates will broadcast over the whole network and consequently, it leads to slowing of convergence time in the conventional network, especially when the size of network topology is increased. This is because the routers will be scaled when the size of network is increased. On the other hand, the floodlight controller in the SDN network is responsible for any change or update, such as link down, by using the OpenFlow control that installs

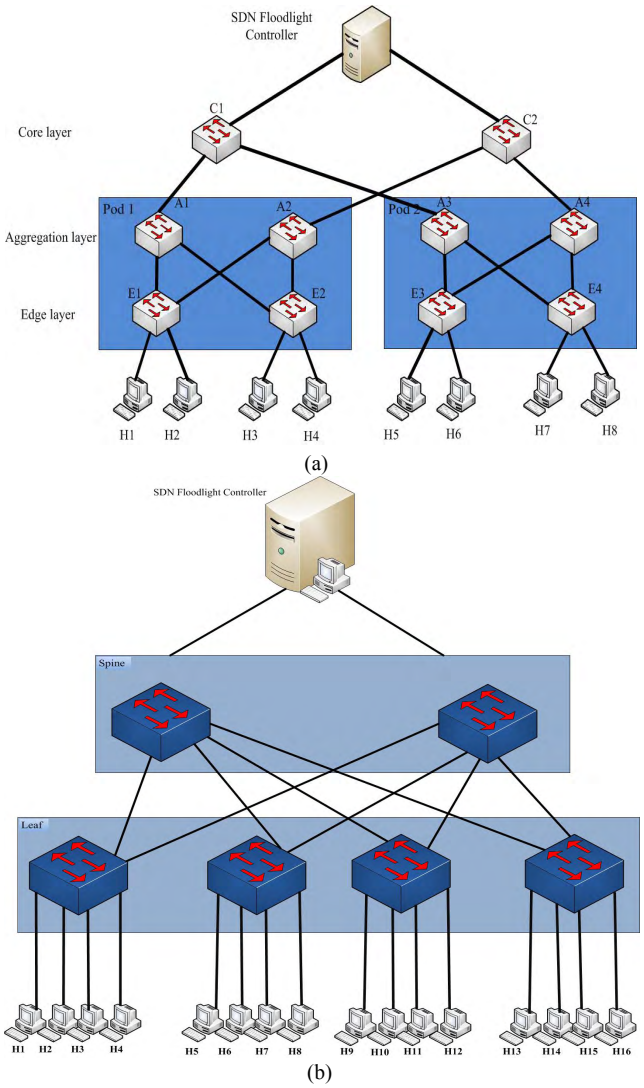

FIGURE 15. (a) Fat tree topology with 10 switches based on SDN. (b) Leaf-spine topology with 6 switches based on SDN.

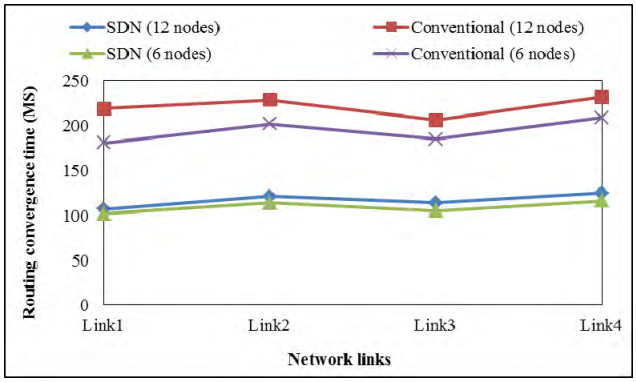

FIGURE 16. Routing convergence time using different topology sizes.

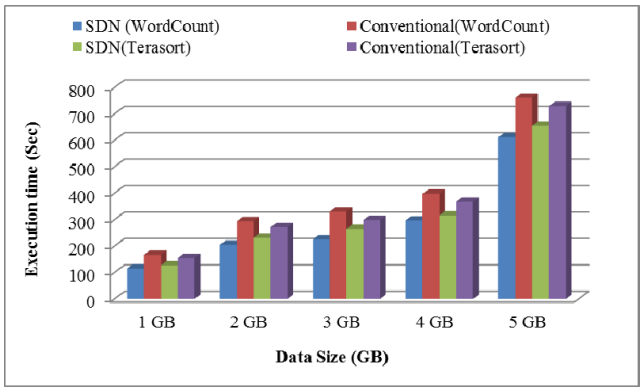

FIGURE 17. Hadoop execution time within leaf-spine using 12 switches.

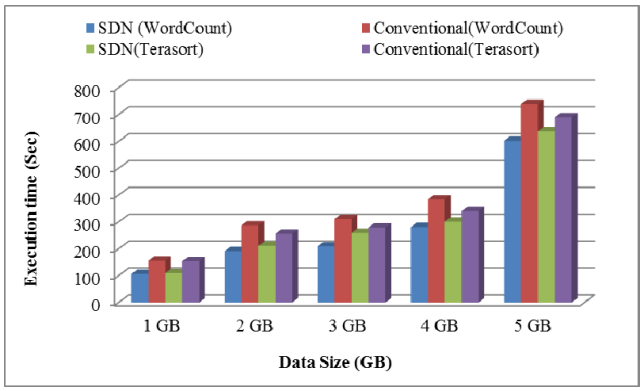

FIGURE 18. Hadoop execution time within leaf-spine using 6 switches.

flow entries into the switches. The controller can also add, delete and modify flow entries for all connected switches in the network. The SDN controller detects whether any link failure has occurred using PORT_STATUS. Furthermore, switches in the network notify the controller of any link down through error messages. When the controller receives the error messages from the connected switches, it computes new available routes based on the flow tables. As a result of the centralized manner of the SDN control, this makes the convergence routing time more rapid and agile. We ran the Word Count and Tera Sort applications to evaluate the performance of a Hadoop job using the proposed system in the leaf-spine topology. We used the optimized values of the Hadoop parameters in the proposed SDN network under different sizes of network topology. Moreover, different sizes of datasets ranging from 1GB to 5GB were used. In the first experiment, it can be clearly observed from figure17 that the execution time of the proposed work based on SDN is reduced when compared to the conventional network for the Word Count application. The execution time of Tera Sort application is also decreased using our proposed approach when compared to the conventional network. In the second experiment, the execution time of Word Count and Tera Sort applications is also shorter than with the conventional network as shown in figure 18. Furthermore, the execution time of both applications under 12 switches in the proposed SDN network was relatively same as that using six switches due to the centralized management of the SDN controller, which can deal with any issues of the routing process, such as congestion or link crashing, irrespective of network topology size. However, the execution time of both applications in the conventional network using 12 switches was increased when compared to utilizing six. As we mentioned above, the routing convergence time is increased when we use a larger network topology size, because of the distributed technique of the conventional network in case of congestion or link down. The dynamic routing of the scheduling and routing process based on an SDN environment has a significant impact on the performance of Hadoop jobs, which is not present in the static environment of a conventional network. We also run

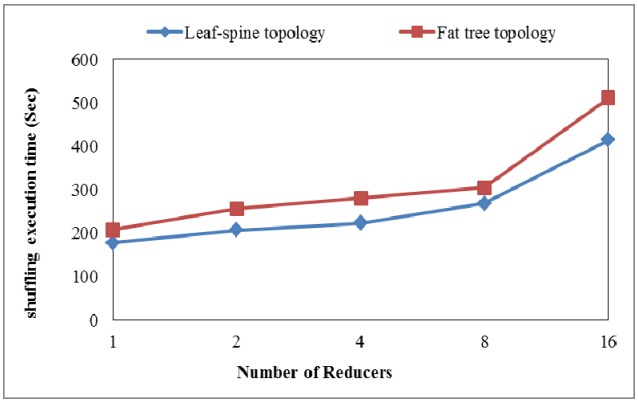

FIGURE 19. Shuffling execution time of the Tera Sort application using different numbers of reducers for both fat tree and leaf-spine topology.

Word Count and Tera Sort applications using both fat tree and leaf-spine topology under different sizes of datasets to evaluate the performance of a Hadoop jobs under different topologies further. Figure 19 shows the execution time of a Hadoop job for the Tera Sort application using both of fat tree and leaf-spine topology under different numbers of reducers. From this figure, it can be clearly observed that the execution time of shuffling flows in the leaf-spine topology can be reduced when compared with the fat tree topology. The reason for this is that fat tree topology is mainly designed to process north-south traffic (*i.e form the core switches to the edge switches*). On the other hand, the traffic between hosts (west-east traffic) in the fat tree topology is representing a challenging task, because some hosts in the network might connect to the same port and then compete for bandwidth, which results in a delay in the response time. Furthermore, the communication between two hosts in the fat tree topology needs to traverse through a hierarchical path from the edge layer to the core layer, thus resulting in latency and traffic bottlenecks.

XI. CONCLUSION

Both a genetic algorithm and genetic programming have been used to tune the configuration parameters of Hadoop

MapReduce automatically. By optimizing the configuration parameter settings, the computing aspect of a Hadoop framework has been improved. This improvement has led to reduce the completion time of Hadoop MapReduce jobs. Further optimization has been performed using software defined network technology. Two applications, namely Word Count and Tera Sort, have been run to evaluate the MapReduce job performance of the Hadoop framework. This work was evaluated using a cluster consisting of 14 VMs placed on the internal cloud at Brunel University London. Another cluster of 14 virtual nodes was employed based on SDN. The results in the traditional network using 14 VMs have shown that our proposed method betters the MapReduce job performance in a Hadoop cluster over Gunther's approach and the default system in a traditional network. Moreover, the results using 14 VMs based on an SDN environment have demonstrated that the performance of Hadoop jobs is superior to that for the traditional network. Another experiment was run to evaluate the performance of Hadoop jobs in a large scale network, namely a data center network also using SDN. The experimental results showed that the performance of Hadoop jobs is higher than for a conventional data center network.

ACKNOWLEDGMENT

The authors also thank Brunel University London for providing the efficient environment to implement this work experimentally.

REFERENCES

[1] J. Nandimath, E. Banerjee, A. Patil, P. Kakade, S. Vaidya, and D. Chaturvedi, "Big data analysis using Apache Hadoop," in *Proc. IEEE 14th Int. Conf. Inf. Reuse Integr. (IRI)*, Aug. 2013, pp. 700–703.

[2] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and Map Reduce," in *Proc. Nirma Univ., Int. Conf. Eng. (NUiCONE)*, 2012, pp. 1–5.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Opear. Syst. Design Implement.*, 2004, pp. 107–113.

[4] A. Pavlo *et al.*, "A comparison of approaches to large-scale data analysis," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2009, pp. 165–178.

[5] (2009). *7 tips for Improving MapReduce Performance*. Accessed: Aug. 15, 2017. [Online]. Available: http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/

[6] Apache Hadoop. *Apache*. Accessed: Aug. 15, 2017. [Online]. Available: http://hadoop.apache.org/

[7] D. Wu and A. Gokhale, "A self-tuning system based on application profiling and performance analysis for optimizing Hadoop MapReduce cluster configuration," in *Proc. 20th Int. Conf. High Perform. Comput. (HiPC)*, Dec. 2013, pp. 89–98.

[8] G. Liao, K. Datta, and T. L. Willke, "Gunther: Search-based auto-tuning of mapreduce," in *Proc. Eur. Conf. Parallel Process.*, 2013, pp. 406–419.

[9] J. Liu, N. Ravi, S. Chakradhar, and M. Kandemir, "Panacea: Towards holistic optimization of MapReduce applications," in *Proc. 10th Int. Symp. Code Gener. Optim.*, 2012, pp. 33–43.

[10] Y. Li *et al.*, "Breaking the boundary for whole-system performance optimization of big data," in *Proc. Int. Symp. Low Power Electron. Design*, 2013, pp. 126–131.

[11] *Hadoop Performance Tuning*. Accessed: Aug. 15, 2017. [Online]. Available: https://hadoop-toolkit.googlecode.com/files/White paper-HadoopPerformanceTuning.pdf

[12] T. White: *Hadoop: The Definitive Guide*, 3rd ed. Sebastopol, CA, USA: Yahoo Press, 2012, p. 688.

[13] M. Li *et al.*, "MRONLINE: MapReduce online performance tuning," in *Proc. 23rd Int. Symp. High-Performance Parallel Distrib. Comput.*, 2014, pp. 165–176.

[14] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of mapreduce programs," *VLDB Endowment*, vol. 4, no. 11, pp. 1111–1122, 2011.

[15] H. Herodotou *et al.*, "Starfish: A self-tuning system for big data analytics," in *Proc. CIDR*, 2011, pp. 261–272.

[16] S. Narayan, S. Bailey, and A. Daga, "Hadoop acceleration in an OpenFlow-based cluster," in *Proc. SC Companion High Perform. Comput. Netw. Storage Anal. (SCC)*, 2012, pp. 535–538.

[17] L.-W. Cheng and S.-Y. Wang, "Application-aware SDN routing for big data networking," in *Proc. IEEE Glob. Commun. Conf. GLOBECOM*, Dec. 2015, pp. 1–6.

[18] S. Zhao and D. Medhi, "Application-aware network design for Hadoop MapReduce optimization using software-defined networking," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 804–816, Dec. 2017.

[19] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu, "Transparent and flexible network management for big data processing in the cloud," in *Proc. USENIX Work. Hot Top. Cloud Comput.*, 2013, pp. 1–6.

[20] C.-Y. Lin and J.-Y. Liao, "An SDN app for Hadoop clusters," in *Proc. IEEE 7th Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, Nov./Dec. 2015, pp. 458–461.

[21] R. Poli *et al.*, *A Field Guide to Genetic Programming*. Morrisville, NC, USA: Lulu.com, 2008.

[22] M. Walker, "Introduction to genetic programming," Univ. Montana, Missoula, MT, USA, Tech. Rep., 2001.

[23] J. McCall, "Genetic algorithms for modelling and optimisation," *J. Comput. Appl. Math.*, vol. 184, no. 1, pp. 205–222, 2005.

[24] S. N. Sivanandam and S. N. Deepa, "Genetic algorithms," *Introduction to Genetic Algorithms*. Berlin, Germany: Springer, 2008, pp. 15–37.

[25] *SDN Architecture Overview*, Open Networking Foundation, Menlo Park, CA, USA, 2013, pp. 1–5.

ALI KHALEEL is currently pursuing the Ph.D. degree in electronic and computer engineering with Brunel University, London, U.K. His research interests include big data analytics, cloud computing, high performance computing, parallel computing, and software-defined networks.

HAMED AL-RAWESHIDY received the Ph.D. degree from Strathclyde University, Glasgow, U.K. He was with the Space and Astronomy Research Centre, Iraq; PerkinElmer, USA; Carl Zeiss, Germany; British Telecom, U.K.; Oxford University; Manchester Metropolitan University; and Kent University. He is currently a Professor of communications engineering with Strathclyde University.

Dr. Al-Raweshidy is also the Director of the Wireless Networks and Communications Centre (WNCC) and the Director of PG studies (ECE) at Brunel University, London, U.K. WNCC is the largest centre at Brunel University and one of the largest Communication Research Centre in U.K. He published over 400 papers in international journals and referred conferences. He has edited the first book in *Radio Over Fibre Technologies for Mobile Communications Networks*.

He acts as a consultant and involved in projects with several companies and operators, such as Vodafone, U.K.; Ericsson, Sweden; Andrew, USA; NEC, Japan; Nokia, Finland; Siemens, Germany; Franc Telecom, France; Thales, U.K. and France; and Tekmar, Italy. He is a principal investigator for several EPSRC projects and European project, such as MAGNET EU project (IP) 2004–2008. His current research area is 5G and beyond, such as C-RAN, SDN, IoT, M2M, and Radio over Fibre.