IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 December 2023, accepted 24 December 2023, date of publication 18 January 2024, date of current version 5 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3355130

RESEARCH ARTICLE

Parallel Deployment and Performance Analysis of a Multi-Hop Routing Protocol for 5G Backhaul Networks Using Cloud and HPC Platforms

SOMAYA A. ABOULROUS[1], AMANY ABDELSAMEA[2], ALI A. EL-MOURSY[3], (Senior Member, IEEE), MOHAMED SAAD[3], (Senior Member, IEEE), FADI N. SIBAI[4], SALWA M. NASSAR[2], AND HAZEM ABBAS[1], (Senior Member, IEEE)

[1]Department of Computer and Systems Engineering, Ain Shams University, Cairo 11566, Egypt
[2]Department of Computer and Systems Engineering, Electronics Research Institute, Cairo 12622, Egypt
[3]Department of Computer Engineering, University of Sharjah, Sharjah 27272, United Arab Emirates
[4]Department of Electrical and Computer Engineering, Gulf University for Science and Technology, Mishref, Hawalli 32093, Kuwait

Corresponding author: Ali A. El-Moursy (aelmoursy@sharjah.ac.ae)

This work was supported in part by the Cloud Computing Center of Excellence through the Science and Technology Development Fund (STDF), Egypt, under Grant 5220; and in part by the Distributed and Networked Systems Research Group Operating through the University of Sharjah, United Arab Emirates, under Grant 150410.

ABSTRACT The main goals of fifth generation (5G) systems are to significantly increase the network capacity and to support new 5G service requirements. Ultra network densification with small cells is among the key pillars for 5G evolution. The inter-small-cell 5G backhaul network involves massive data traffic. Hence, it is important to have a centralized, efficient multi-hop routing protocol for backhaul networks to manage and speed up the routing decisions among small cells, while considering the 5G service requirements. This paper proposes a parallel multi-hop routing protocol to speed up routing decisions in 5G backhaul networks. To this end, we study the efficiency of utilizing the parallel platforms of cloud computing and high-performance computing (HPC) to manage and speed up the parallel routing protocol for different communication network sizes and set recommendations for utilizing cloud resources to adopt the parallel protocol. Our numerical results indicate that the HPC parallel implementation outperforms the cloud computing implementation, in terms of routing decision speed-up and scalability to large network sizes. In particular, for a large network size with 2048 nodes, our HPC implementation achieves a routing speed-up of 37x. However, the best routing speed-up achieved using our cloud computing implementation is 15.5x, and is recorded using one virtual machine (VM) for a network size of 1024 nodes. In summary, there is a trade-off between a better performance for HPC vs. flexible resources of cloud computing. Thus, choosing best fit platform for 5G routing protocols depends on the deployment scenarios at 5G core or edge network.

INDEX TERMS 5G routing protocol, cloud radio access networks, cloud computing, HPC, ultra-dense network.

I. INTRODUCTION

Cloud computing has gained significant popularity as an Internet-based and cost-effective computing model to access a massive configurable and virtualized pool of shared computing resources on a user-demand basis [1]. Cloud users can scale up and scale down with high reliability and quick provisioning procedures unlike traditional computing models e.g., high performance computing (HPC). Cloud environments, from the HPC point of view, is a distributed

The associate editor coordinating the review of this manuscript and approving it for publication was Jjun Cheng.

computing model that can run large applications on centralized, scalable, and computationally powerful resources due to the virtualization benefits. HPC systems typically do not have the luxury of elastic provisioning and dynamic scalability upon-demand (cloud characteristics). Hence, most HPC applications recently started utilizing the cloud resources [2], called (HPC cloud) to build a parallel Virtual Cluster (VC) to move and run HPC applications on the cloud. In this approach, Infrastructure-as-a-Service (IaaS) cloud solution is adopted for HPC users. Some compute-intensive applications target cloud features such as centralized processing and management [3], as well as dynamic scalability and resources utilization. Many studies show that the cloud can be a better candidate than a dedicated HPC cluster to yield speed up for some of HPC applications and benchmarks [2], [4]. Their studies also show that the cloud degrades HPC performance due to virtualization effect on network latency, resource sharing among multi-tenants, and the idle time of slowest threads over VMs. On the other hand, HPC is considered as a perfect candidate parallel platform for a wide range of compute-intensive applications in terms of performance possibility and application scalability. However, HPC centers have a big challenge to provide higher computing performance with less resource input regarding cost, maintenance and operation. Finally, there are limitations that define best-fit applications for the cloud such as parallel performance, scalability, and cost-benefit. Both HPC and cloud computing have in common the advantage of computationally powerful servers and the capability of parallel processing for compute-intensive applications. These applications are mainly found in production and manufacturing, science and engineering simulation research, healthcare, and financial sectors. The wireless communication sector also utilizes cloud computing functionalities in fifth-generation (5G) systems.

Mobile wireless communication systems facilitate our daily activities. By providing seamless connectivity, these systems allow almost all the world's smart devices to communicate with extremely low latency and high-speed throughput. It is expected that the next generation of mobile wireless communications, 5G, will provide connectivity to the numerous smart devices and interconnected things [5], [6]. By 2023, there will be 5.3 billion total Internet users (66 percent of the world's population), up from 3.9 billion in 2018, and 29.3 billion devices on the network, according to the Cisco Annual Internet Report (2018-2023) [7]. Hence, the overall mobile data traffic is forecasted to increase to 77 exabytes per month by 2022, which is a seven-fold increase over that in 2017. 5G networks should not only support this massive volume of mobile data traffic and network capacity, but also support new services requirements such as high data rate (10 GB), low latency (1 ms), high reliability, and enhanced spectral efficiency (SE), which defined as the bandwidth-nominalized-data-rate (in bits/sec/Hz), and energy efficiency (EE). Therefore, the 5G development is not only concerned with achieving higher capacity, but it also represents a paradigm shift in mobile cellular networks to provide the capability for better coverage and new services at high quality-of-service (QoS) and lower operational cost for Mobile Network Operators (MNOs). This requires various technologies and architectures to be proposed for efficient and flexible 5G Radio Access Networks (RANs). The main projected technologies to accommodate 5G needs are network densification, Millimeter Wave (mmWave), and the virtualization of some base-station functionalities in a centralized cloud. The latter leads to the Cloud Radio Access Network (C-RAN) technology. Network densification or the ultra-dense network (UDN) is considered a critical technique to meet the requirements of explosive mobile data traffic in 5G mobile communications. Deploying a massive number of low-power small wireless cells can enhance the coverage in crowded areas and increase network capacity. As a result, these cells produce massive backhaul traffic to the core network. Consequently, because of the various 5G service needs (ultra-low latency, low power consumption, and high data rate), cost-effective backhauling in UDN is a vital issue that must be addressed [8], [9], [10]. Millimeter wave (mm-wave) technology is cost-effective, and it only performs well for short-distance line-of-sight (LOS) communications [11]. Therefore, multi-hop routing is highly needed for mm-wave backhauling in UDN to enhance the flexibility of path selection among wireless cells [12]. Since its processing is compute-intensive, the path selection process at wireless cells requires massive processing power. Besides that, using mm-wave between wireless small cells should avoid blockage and signal propagation loss, which affects spectral efficiency [8] and end-to-end latency of data flows [13]. In nutshell, the procedure of best route selection requires extensive computation and time-consuming effort in UDN, which necessitates speed-up, particularly with different 5G service requirements. C-RAN is the most prominent way to afford such extensive computation needed through virtualization and cloud computing concepts. C-RAN allows more dynamic traffic handling and the best path selecting in mm-wave backhauling. C-RAN is supposed to tackle network traffic growth for end-users by considering different 5G service use cases. Accordingly, RAN construction and optimization need new protocols [14].

The motivation of using cloud computing with 5G, especially in C-RAN architecture, is to promote centralized processing and management, scalability, and parallel execution [15]. C-RAN has major tenets from centralization and cloudification of RAN architecture to improve spectral efficiency and energy efficiency, and to reduce latency and network costs [15], [16], [17]. C-RAN is attracting MNOs to decrease the cost of network operations, maintenance, and upgrade procedures and increase throughput and decrease delay. As massive amounts of 5G network connections, large amounts of data need to be efficiently processed and analyzed in real-time. Therefore, this paper is set out to develop an efficient multi-hop routing protocol for 5G backhauling

in UDN and C-RAN architecture, utilizing the parallel platform offered by the cloud and HPC clusters. The major objective of this research is to compare the performance of virtual platforms on OpenStack private cloud to conventional parallel HPC platforms.

The following points summarize our contributions in this paper:

- We develop a parallel multi-hop routing protocol implementation of the algorithm proposed in [18].
- We deploy a virtual cluster using cloud computing and analyzing the performance of the parallel protocol.
- We analyze and compare the efficiency between virtual cluster results and HPC cluster results [19] in the speed-up and performance enhancement provided by the parallel multi-hop routing protocol.
- We evaluate of the scalability of (HPC and cloud computing) parallel platforms with different network sizes (network densification effect). Thus, our numerical study takes extremely large sizes of networks with thousands of communication nodes into consideration. It contrasts considerably with the literature that usually uses networks with tens of nodes.
- We set recommendations for the adoption of both platforms to be highly utilized by 5G protocols in terms of scalability, speed up, and resources or infrastructure requirements in 5G deployment.

The remainder of the paper is organized as follows. Section II presents related work for multi-hop routing protocols in 5G backhaul network. Our network model is shown in section III. Section IV describes the multi-hop routing algorithm that we use. Section V presents the parallel implementation details of the multi-hop routing protocol. The evaluation methodology is presented in section VI. Section VII discusses the cloud results and comparative results with HPC. Finally, Section VIII discusses our conclusions and upcoming work.

II. RELATED WORK

Multi-hop routing in backhaul networks is considered an optimization problem to find the best solution that achieves the network objectives and satisfies the design constraints. The objectives considered in the literature include some of 5G network requirements such as high spectral efficiency, ultra low-latency, high capacity, and high energy efficiency. Hence, flexible and efficient multi-hop routing protocol is critically needed for backhaul network in all 5G scenarios. The work in [8] proposed multi-hop routing schema in UDN with maximizing the transmission rate as the optimization objective under network channel and flow constraints using a Dijkstra algorithm. Also, [20] proposed a multi-hop multi-path selection scheme that achieves optimization objectives of selecting the path and allocating the data rate under latency constraints using reinforcement learning techniques. Both previous works do not find the optimal solution for selecting the best path in the backhaul network, and do not assure a fixed level of spectral efficiency. The authors of [21] proposed an energy-efficient routing reactive protocol to locate low-level energy nodes for reliable transmission in 5G network. However, their process takes several tens of seconds in the simulation results. The study in [22] proposed optimal routing algorithm for UDN using a combination of Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) to handle network scalability and seamless QoS and link reliability. However, the procedures of GA and PSO are slower than other classical route optimization algorithms, such as the Bellman-Ford algorithm [23]. Also, the proposed algorithm does not consider power consumption along the selected path. The study in [24] adopted a C-RAN architecture to present a centralized software-defined network algorithm to avoid data traffic congestion in backhauling routes via dynamic path selection. The work in [25] proposed a routing algorithm for Cloud Assisted-Mobile Ad Hoc Networks (CA-MANETs) to achieve lower energy consumption and increased network lifetime. The proposed algorithm works through two phases. First, the local path recovery phase aims to minimize energy consumption among mobile nodes and peer nodes. Then, the path discovery phase aims to select the shortest path with less routing cost. However, these procedures have a high computational cost, especially in UDN deployments. The work in [26] proposes a C-RAN architecture, and presents an algorithm for joint routing and VM selection such that the total energy cost is minimized and the task response time required by user is satisfied. The algorithm, however, neglects other 5G service requirements such as achieving a target spectral efficiency. In combination of UDN and C-RAN architectures, other studies introduced multi-hop routing protocols such as [27], in which a multi-hop relaying protocol for fronthauling signals among remote radio heads (RRHs) is proposed to minimize capacity-constrained fronthaul and the maximum allowable network delay as optimization objectives. The optimization design is restricted by a maximum number of relays per hops to three. The work in [28] proposed a RRH placement strategy to assign traffic and resources to a few low-traffic RRHs for maximizing backhaul survivability and energy efficiency. This solution depends on deep neural networks for learning and predicting the data traffic. Accordingly, it requires expensive GPUs and high processing to train for complex data.

In general, recent work is limited to a subset of 5G network requirements [29], [30], [31], [32]. Unlike previous work, the algorithm we use, which was initially proposed in [18], chooses the best optimal path with considering the 5G requirements such as a spectral and energy efficiency target. This algorithm is guaranteed to generate the exact optimal path, with an execution time of milliseconds for networks with few tens of nodes. However, if the network density is increased in 5G networks and beyond, the algorithm execution time may become problematic. The process of the Bellman-Ford algorithm used in [18] is iterative and computationally intensive, especially in large networks with hundreds or thousands of communication nodes [33]. Therefore, it gives rise to the algorithm performance improvement

using parallelization which has not been completely explored before. None of the literature work considers speeding up the path selection process for ultra-dense networks using parallel platforms (i.e., cloud computing and/or HPC).

Several parallel implementations for the shortest path problem algorithm (Bellman-ford) were proposed in the literature. However, these implementations are different in terms of objectives, the applied parallel programming, the platform for parallel implementation, comparison algorithms, and disadvantages as shown in Table 1. The work in [34] presented different parallel implementations of Bellman-Ford algorithm on GPU using parallel framework OpenCL. These implementations of Bellman-ford are Single Source Shortest Path (SSSP) and All Pair Shortest Path (APSP). Their work targets performance comparative analysis on Central processing units (CPUs) and Graphical processing units (GPUs) using real environment of two compute nodes. However, this work did not use hybrid execution of CPUs and GPUs. Another work in [35] presents a high-performance implementation of the Single Source Shortest Path (SSSP) Bellman-Ford algorithm that exploits the architectural features of recent GPU architectures of NVIDIA (Kepler GPU) to improve the performance and workload efficiency. CUDA parallel framework is used for NVIDIA GPUs. Their work compare with sequential Dijkstra and other parallel Bellman-Ford implementations on GPUs. However, this work lacks more low-level instructions parallelization such as OpenCL and CUDA. The study in [36] proposed a work-efficient, Multiple Source Shortest Path (MSSP) implementation of the Bellman-Ford algorithm is proposed to dramatically increase the performance of shortest path calculations for low-density high diameter graphs to apply in transportation networks. Their implementation is used both frameworks OpenMP and CUDA. Several sets of hardware were used to evaluate performance in [36], with multiple CPUs and GPUs servers used and real-world benchmark. Then, comparative analysis is done among different benchmarks and hardware. However, the paper did not use greater number of GPUs. Finally, the work in [37] proposed a new SSSP-Asyn (Single Source Shortest Path in asynchronous mode) technique, which is parallelized form of inter node Dijkstra and intra node Bellman Ford algorithm and implemented in Message Passing Interface (MPI) framework. Their results generated from PaRMAT (multi-threaded RMAT graph generator) simulator among 32 processors only. Their algorithm is evaluated using different network graph sizes. Our implementation is different from these implementations since we use MPI parallel programming for SSSP Bellman-Ford algorithm parallelization using HPC and cloud computing platforms to be applied in 5G networks. The motivation of using cloud computing with 5G promotes central routing decision and provides a potential for enhancing the overall performance of route selection using parallel processing which is not discussed before in literature work. In a previous effort, the study in [19] has developed a parallel multi-hop routing protocol for the

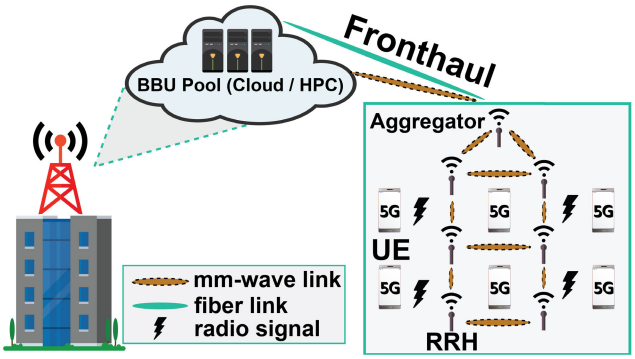


FIGURE 1. UDCSnet 5G network architecture.

algorithm originally proposed in [18] using HPC platform. This follow-up paper extends primarily results in [19] using cloud virtual cluster, in an attempt to speed up path selection and satisfy the needs of 5G UDNs. In contrast to [19], we use cloud computing as a cost-effective platform to parallelize the routing algorithm proposed in [18] using HPC over cloud, to speed up the path selection procedure, and to fulfill the requirements of 5G UDNs. The parallelization process starts with performance and code analysis for the serial algorithm. Then concurrency analysis is performed for the different tasks to reveal the data dependencies. Finally, the parallel algorithm is implemented using a parallel programming model.

III. NETWORK MODEL

Due to small cells densification advantages, UDN considers a capacity-enhancing and coverage-expanding approach for targeting 5G data and network growth requirements. C-RAN is one of the critical cutting-edge and green technologies adopted by 5G networks due to its advantages of centralization and cloudification. A combination of C-RAN and UDN, as known as ultra-dense cloud small cell network (UDCSnet), utilizes both small cells densification and centralized processing simultaneously. Accordingly, this network model improves spectral efficiency, coverage with high capacity demands, and energy efficiency [17], [38], [39]. Many studies [28], [39] adopt the UDCSnet architecture or Heterogeneous (H-CRAN) due to its flexibility and scalability for 5G system. In this paper, we also adopt the UDCSnet architecture as shown in Fig.1. There are heterogeneous base stations such as traditional macro base station (BS), which are placed on building rooftops to give full network coverage, in addition to antennas of smaller base stations (small cells) to give increased data rates and extended coverage for remote locations. Unlike the traditional RAN architecture, baseband processing functions of small cells are performed in centralized powerful data centers (cloud computing or HPC) using clustered BBU pools. The BBU pools accomplish the intolerable task of central processing and resource management. Remote Radio Heads (RRHs) of small cells are distributive deployed in a grid with network size N where N is the number of RRHs. RRHs play two

TABLE 1. Comparative analysis of parallel Bellman-Ford algorithm implementations.

Authors names	Paper objectives	Parallel programming frameworks	Platform for parallel implementation	Comparison algorithms	Disadvantages
Gaurav Hajela *et al.* **[34]**	presents and compare different parallel implementations of Bellman-Ford algorithm for shortest path problems (SSSP, APSP) using CPUs and GPUs	OpenCL	HPC servers	Serial implementations of SSSP and APSP Bellman-Ford on CPUs and their parallel executions on CPUs and GPUs	The paper did not use hybrid implementation of Bellman-Ford by partitioning the algorithm among CPUs ad GPUs in a balance manner.
Federico Busato *et al.* **[35]**	presents a high-performance implementation of Bellman-Ford algorithm for the Single Source Shortest Path (SSSP) problem to improve the performance and workload efficiency	CUDA	HPC servers with Kepler GPU	Boost library sequential Dijkstra, and other Bellman-Ford implementations for GPUs using graphs of different graph sizes	They did not use other techniques of low-level instructions parallelism such as OpenCL with CUDA.
Peter Heywood, Steve Maddock *et al.* **[36]**	proposes a work-efficient, Multiple Source Shortest Path (MSSP) problem implementation of the Bellman-Ford algorithm to increase the performance of shortest path processing for transportation networks	OpenMP and CUDA	HPC servers with GPUs	Different set of hardware and benchmarks for the proposed algorithm	There is lacking of using greater numbers of GPUs, through high-density GPU nodes such as the NVIDIA DGX-2 or through distributed computing to improve performance for large workloads
Yadav, Sangeeta *et al.* **[37]**	proposes a new SSSP-Asyn (Single Source Shortest Path in asynchronous mode) technique, which is parallelized form of inter node Dijkstra and intra node Bellman Ford algorithm	MPI	Simulator	Different network graph sizes	The paper did not use real platform

light functions to relay baseband signals from user equipment (UE) to the BBU pool (cloud computing or HPC), and vice versa. Therefore, the network size can be scaled up flexibly and cost-effectively. The fronthaul or transport network is the connection layer between the BBU pool and RRHs to give high bandwidth. Fronthauls can be realized using different technologies that include optical fiber or mm-wave communications. Using millimeter-wave communication for relaying signals among RRHs is considered a cost-effective way to guarantee a fixed spectral efficiency for backhauling in UDN. The procedures of this network model work as follows:

- The aggregator node, the nearest RRH to BBUs pool, forwards the relaying UE data traffic to the BBUs pool via fronthaul links for central processing of path selection.
- The BBUs pool is deployed at a centralized location with high computation and storage capabilities (i.e. cloud computing or HPC).
- The proposed parallel routing protocol is deployed on HPC or cloud clusters within the BBU pool, to accelerate routing decisions from a source RRH to a destination RRH.

The benefit of parallelization is two-fold. On the one hand, it utilizes the cloud resources available in C-RANs. On the other hand, it allows scalable and parallel processing in very large networks as anticipated in 5G. One of the envisioned 5G scenarios for our network model is the Enhanced Mobile Broadband (eMBB) scenario to deal with massively increasing data rates, dense user populations, and extremely high traffic capacity in hotspot areas. Another possible scenario is the Massive Machine-type Communications (mMTC) scenario for the Internet of Things (IoT), requiring low power consumption and high data rates for very large numbers of connected devices. Some of 5G applications of these scenarios are 4K high definition (HD) video, smart cities, smart homes, and industrial automation. In the following section we summarize the serial routing algorithm, which will be parallelized using HPC and cloud computing platforms.

IV. MULTI-HOP ROUTING ALGORITHM

We are given a multi-hop wireless network represented by a graph $G = (V, E)$, where V is the set of communication nodes (RRHs) and E is the set of links joining the nodes. We let $N = |V|$ and $M = |E|$ denote the number of nodes and links in the network, respectively. The multi-hop routing algorithm used in parallelization is originally proposed in [18]. It finds the optimal shortest path from source node $s \in V$ to destination node $d \in V$ that jointly addresses an end-to-end spectral efficiency level and allocates minimum transmit power among communication nodes along the selected path. The suitability to 5G systems highly depends on jointly achieving a spectral efficiency level (in bits/s/Hz) and high energy efficiency through minimizing total power consumption used in the backhauling network. It has been established in [18] that the maximum transmit power used by any link on some path L, denoted by $P_{max}(L)$, is given by

$$P_{max}(L) = \left(2^{\gamma|L|} - 1\right) \max_{l \in L} \frac{N_0 B}{G_l} \tag{1}$$

where γ is the required spectral efficiency target (in bits/s/Hz), $|L|$ denotes the hop-count of path L, l signifies

a link on path L, N_0 is the noise power spectral density, B is the channel bandwidth (in Hz) and G_l is the path gain from the transmitted of link l to its receiver. The problem addressed in [18] can be formulated as finding the path from a source node s to a destination node d such that (1) is maximized. It has been proven that the following algorithm is theoretically guaranteed to provide the exact optimal solution to the problem [18]:

Algorithm *Max-Power Minimization*

1) For each hop-count $h = 1, 2, \ldots, N-1$:
   a) Find L_h, the widest path from s to d with at most h hops, using $w_l = G_l/N_0B$ as the link metric.
   b) Calculate maximum transmit power $P_{max}(L_h) = \left(2^{\gamma|L_h|} - 1\right) \max_{l \in L_h} \frac{N_0 B}{G_l}$.
2) Return the path with the smallest $P_{max}(L_h)$.

The following two comments are worth noticing. The *Max-Power Minimization* algorithm can be implemented using the Bellman-Ford (BF) algorithm. The BF shortest path algorithm can be modified to produce the widest path from a single source node to every other communication node (destination node). This is known as the maximum capacity path (widest path) problem. Furthermore, the BF algorithm has the implicit property of identifying the optimal and widest path from the source to the destination, among paths of at most h hops at its hth iteration. The *Maximum-Power Minimization* algorithm is implemented by calling BF algorithm only *once*, as opposed to $N-1$ times. To illustrate the Bellman-Ford procedure, the following definitions are needed.

- $w_{i,j}$: weight of link $(i, j) \in E$, i.e., $w_{i,j} = \frac{G_{i,j}}{N_0 B}$.
- $Width_h^i$: width of the widest path from the source node $s \in V$ to any other node $i \in V$ such that the path has at most h hops.
- $Length_h^i$: hop-count of the widest path from the source node $s \in V$ to any other node $i \in V$ such that the path has at most h hops.
- $pred_h^i$: predecessor of node $i \in V$ on the widest path from the source node $s \in V$ such that the path has at most h hops. The widest path with at most h hops from the source node $s \in V$ to destination d can be explicitly constructed by tracing $pred_h^i$ backwards from d until s.

The procedure for the widest path computation of the Bellman-Ford algorithm is explained below.

Procedure *Bellman-Ford*

1) INITIALIZATION:
   Let $Width_h^s = \infty$ for $h = 0, 1, \ldots, N-1$, and $Width_0^i = 0$ for all $i \neq s$.
2) ITERATION:
   for h=1,2, ..., N-1
     for all $i \in V$ do $Width_h^i := Width_{h-1}^i$; $Length_h^i := Length_{h-1}^i$;
     for all $(i, j) \in E$ do:
       if $\min\{Width_{h-1}^i, w_{i,j}\} > Width_h^j$, then /* (S_1) */
         $Width_h^j := \min\{Width_{h-1}^i, w_{i,j}\}$
         /* (S_2) */
         $Length_h^j := Length_{h-1}^i + 1$;
         $pred_h^j := i$.

Now, algorithm *Max-Power Minimization* is implemented using a single run of Procedure *Bellman-Ford*, with the following two changes. At the end of each of the $N-1$ iterations of Step 2, the maximum transmit power $P_{max}(L_h)$ needed for path L_h is calculated using (1). Moreover, after Step 2, an additional step is added, in which the path with the smallest $P_{max}(L_h)$ is returned as the solution to the problem.

The functionalities of serial algorithm implementation can be described by the following tasks:

1) *updateNetwork* function: relies on a single call to Bellman-Ford algorithm to search out the widest paths for pair of given source-destination nodes at every hop count.
2) *caculatePmax* function: computes *Pmax* for the widest path for a pair of source-destination node at every hop count $h = 1, 2, \ldots, N-1$. The path with the smallest *Pmax* value is then returned.

The *Max-Power Minimization* algorithm has a running time in the order of milliseconds for networks with tens of nodes [18]. The ultra densification of the network in 5G, however, may introduce difficulties in the algorithm's execution time. In addition, the BF algorithm is a more computationally demanding procedure than Dijkstra algorithm, especially in large networks with hundreds or thousands of communication nodes [33]. As a result, the serial algorithm extensively needs parallelization. The primary aim of this study is to parallelize the *Max-Power Minimization* algorithm proposed in [18], using both cloud computing and HPC platforms, to explore the efficiency of these platforms to accelerate the path selection between a pair of source-destination nodes in our network model. The main goal of this research is to parallelize the *Max-Power Minimization* algorithm to have an efficient multi-hop routing protocol in 5G networks that utilizes cloud computing and HPC platforms. The parallelization process starts with performance and code analysis for the serial algorithm. Then, concurrency analysis is performed for the different tasks to reveal the data dependencies. Finally, the parallel algorithm is implemented using a parallel programming model.

V. PARALLEL PROTOCOL IMPLEMENTATION

A. PERFORMANCE ANALYSIS

The parallelization methodology begins with an analysis of execution to detect hotspots in the serial algorithm by the Gprof (GNUproler) tool [40], which is used for

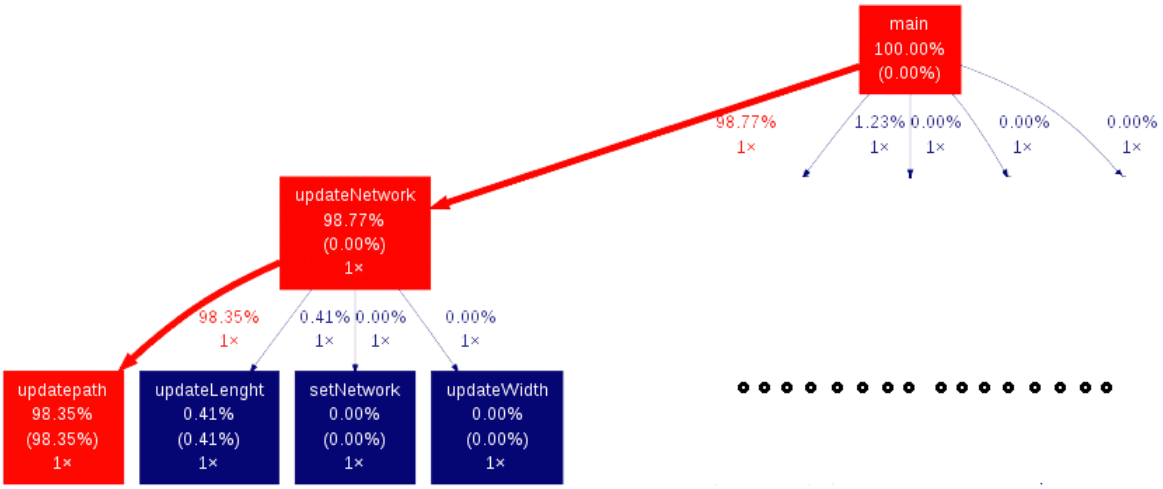


FIGURE 2. Part of the callgraph of the serial multi-hop routing algorithm.

performance analysis for Linux/Unix applications. This study provides us with the relative execution time for each function, which is an important indicator for hotspots investigation to exploit parallelization opportunities then, in a balanced manner, distribute the load among the processing units. The tool generates a call graph that declares the relationships among functions and their dependencies. Also, it defines the execution time percentage for each function and its callee in a colorful manner. The high-percentage function in the graph indicates a hotspot function. Fig. 2 shows a part of the output callgraph for the serial multi-hop routing algorithm that indicates how much time is spent in each function and its callee. As shown, The 'updateNetwork' function occupies 98.77% of the serial algorithm total serial execution time. The 'updateNetwork' function represents the Bellman-Ford algorithm procedure in four subfunctions as listed below:

- setNetwork function: defines the initialization of Bellman-Ford procedure.
- updateWidth function: updates the width of all paths that begin from the source node to any other node using iterative communication with their neighboring nodes.
- updateLength function: updates the length of all paths that begin from the source node to any other node.
- updatePath function: defines the execution of the widest paths estimate updates from a given source to destination for all hops in the network.

We concentrate on the 'updatePath' function because it takes almost all of the execution time of serial implementation 98.35%.

B. CODE ANALYSIS

The purpose of the 'updatePath' function is to find the widest path between a given source and a given destination node in the network. We analyze the function using an activity diagram to display a comprehensible summary of the execution flow as shown in Fig.3. The function iterates three times. The outer loop iterations indicate network hops, while the other loops iterate communication nodes. The outer loop is defined as the intermediate nodes that begin the path search from the source to all other nodes. The inner loop is then referred to as destination nodes. Width and length arrays represent output data structures with two dimensions: hop count and network communication nodes. The weighted link between two communication nodes is represented by a weight array. Single source shortest path Bellman-Ford algorithm checks all paths starting from a given source node to all nodes at most h iterative phases. At each phase, all edges or links are checked to update the widest path estimate to the destination node by going through intermediate nodes and taking the weight of their edge into consideration. Hence, the function 'updatePath' works as follows. There are two main phases in this process, one phase is for the number of communication nodes in the network and the other is for the network links. First, all one-hop routes are searched for the path from source to all communication nodes. Second, further iterations (for each possible hop count) are performed to traverse across all the network links. Then, the width estimate value of the destination node may be changed. The path from the source s to any destination node is extended by links through intermediate nodes. The core of 'updatePath' function follows this equation 2.

$$Width_h^j := \min\{Width_{h-1}^i, w_{i,j}\} \tag{2}$$

The widest path from source node s to destination node j is the minimum of the width of the path to the node following source node (intermediate node i) and the weight w from that node to node j. To find the widest path from any node i, start at i and follow the corresponding links of the network until node j is reached for h iterations. Finally, after successful completion of algorithm, width will contain the widest path to all the communication nodes from the source s at most h hops.

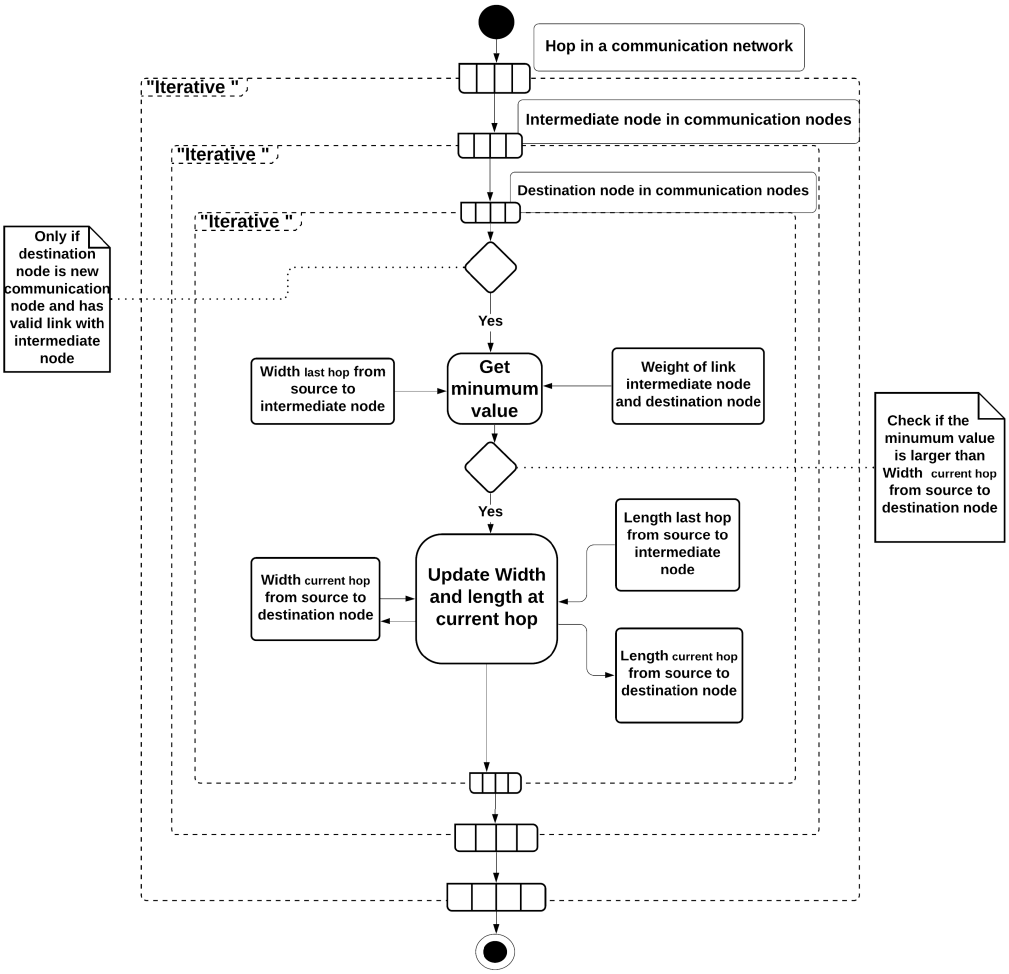


FIGURE 3. The activity diagram of the 'UpdatePath' function.

C. CONCURRENCY ANALYSIS

Using loop iterations in the 'updatePath' function, the analysis indicates that the jobs may be processed concurrently by defining loop-carried dependencies. The relationships between iterations within a loop and through loops identify whether tasks can be distributed across parallel processing elements (processors) or not. Fig.4 depicts the Loop-carried Dependencies Graph (LDG) across loops in the function. Every LDG node represents two dimensional iteration, and edge shows the dependency between LDG nodes. The graph iterates through communication nodes and hop count using i or j, and h, respectively. The primary instructions of the 'updatePath' function are presented by S_1 and S_2, which are previously referred in the Bellman-Ford procedure in section IV. Also, Fig. 5 shows the dataflow for 'updatePath' function regarding the width data structure. The width represents as an input and output for S_1 and S_2.

LDG graph shows that S_1 uses the value $Width_h^j$, computed by S_2 in the previous hop iteration. The iteration h computes $Width_{h-1}^i$ read in iteration $(h-1)$. Therefore, a loop-carried dependence prevents parallelism across hop count (h) and intermediate node (i) iterations. S_2 updates the value of $Width_h^j$ at the same iteration. Hence, each processing element changes its output and input width values, iterations at the destination node (j) may be split across parallel processing elements.

D. PARALLEL IMPLEMENTATION

The underlying infrastructure of the hardware system is the fundamental factor that defines the parallel implementation. We utilize a multi-processor distributed memory architecture because of its scalability. For building our distributed-memory parallel implementation, the master-slave programming paradigm is the best match since it is a widely utilized approach for executing independent tasks while being supervised by a control processor. In this approach, one processor known as the master oversees carrying out the optimization functions. The master-slave job consists of three fundamental tasks; preprocessing, computation, post-processing.

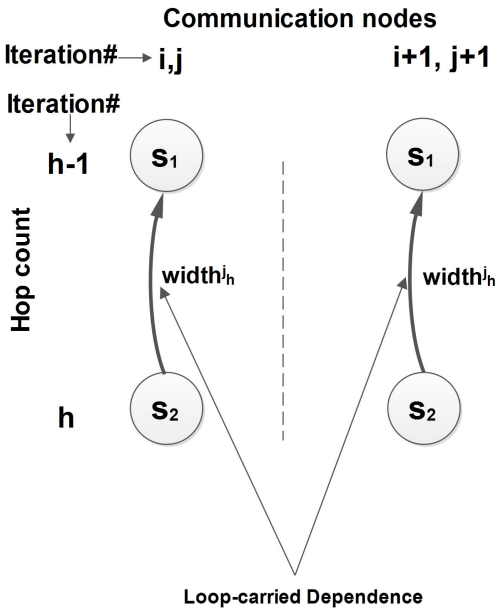


FIGURE 4. Loop-carried dependencies graph across 'updatePath' loops for the width data structure.

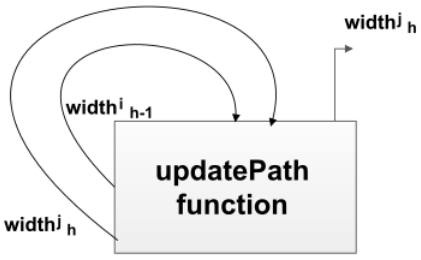


FIGURE 5. Dataflow across 'updatePath' function for the width data structure.

1) Preprocessing task: it is to initialize and set up tasks to collect the data needed by slave processors and send it to slaves. This task is done by the master processor.
2) Computation task: in this task, the slaves receive the data and code from the master, they do computation on the data then transfer the results back to the master.
3) Post-processing task: it is a gathering task to receive the local results, and perform other post-processing on these results by the master processor.

For building our application in a distributed-memory system, we utilized the Message Passing Interface (MPI), which is the most typical and dominant programming paradigm. The MPI offers a programming model in which processes interact with other processes by directly calling the library routines to send and receive messages. The benefits of the MPI programming model include direct control of data distribution by the programmer, process synchronization, explicit communication, and the ability to optimize the data locality. This allows MPI applications to operate on multiprocessing or multi-core systems with great scalability, flexibility, and performance.

Fig. 6 shows the parallel functions that are assigned to each processing unit, as well as their execution sequence and communication. Prior to transmitting data to other slave processors, the master processor must first configure the data to be transmitted, such as network settings (e.g., weights, links, width, and length), and then initialize the hop count value, which is then transferred to slave processors. MPI collective operations exploit to realize communication between processors in the same communicator group. These operations imply a barrier synchronization across all group members. Configuration settings are distributed to all slave processors at the same time using broadcast communication. Second, all processors get a width and length array partition, called *a chunk*. The number of network communication nodes divided among all the processors in a system. Each processor determines where it will begin processing in the width and length arrays after the data chunk has been calculated. This starting point is called *step* which is determined by the product of the logical number of processors (*rank*) and chunk size. Third, for each hop count iteration, each CPU initializes and then updates local width and length arrays according to the chunk size. Before increasing hop iteration, each processor broadcasts and receives other local arrays via collective communication to be aggregated in its buffer. To gather all the updated local arrays at width and length arrays, *allgather* communication is performed. All processors converge on a synchronization barrier to do another iteration of the hop count. Finally, once all hop iterations have been completed, the master processor updates the length and width arrays and performs post-processing on them. In order to select the path with the lowest maximum-power value, the master processor initially computes the maximum power for the paths from a given source to destination node resulting at each hop count.

VI. EVALUATION METHODOLOGY

A. EXPERIMENTAL SETUP

To evaluate our MPI parallel implementation, we carry out experiments using a virtual cloud cluster at Electronics Research Institute (ERI) HPC/cloud center of excellence [41]. The ERI HPC/cloud system is divided into two main platforms, HPC and OpenStack cloud which are controlled by Bright Computing Linux Cluster Manager version 8.1 [42]. The ERI HPC/cloud system has additional units (such as storage, UPS……) which allow the system to operate efficiently with full monitoring by the administrators. The ERI cloud platform is implemented with Bright OpenStack Liberty release [43] with kernal based virtual machine (KVM) hypervisor which is managed by Bright Computing Manager. The cloud platform consists of eight servers as shown in Fig. 7. One server acts as a controller with two 8-core processors (Intel(R) Xeon E5-2640), each running at 2.4 GHz, where each processor has 20 threads. The remaining seven servers act as compute servers. Five compute servers have two processors (Intel(R)

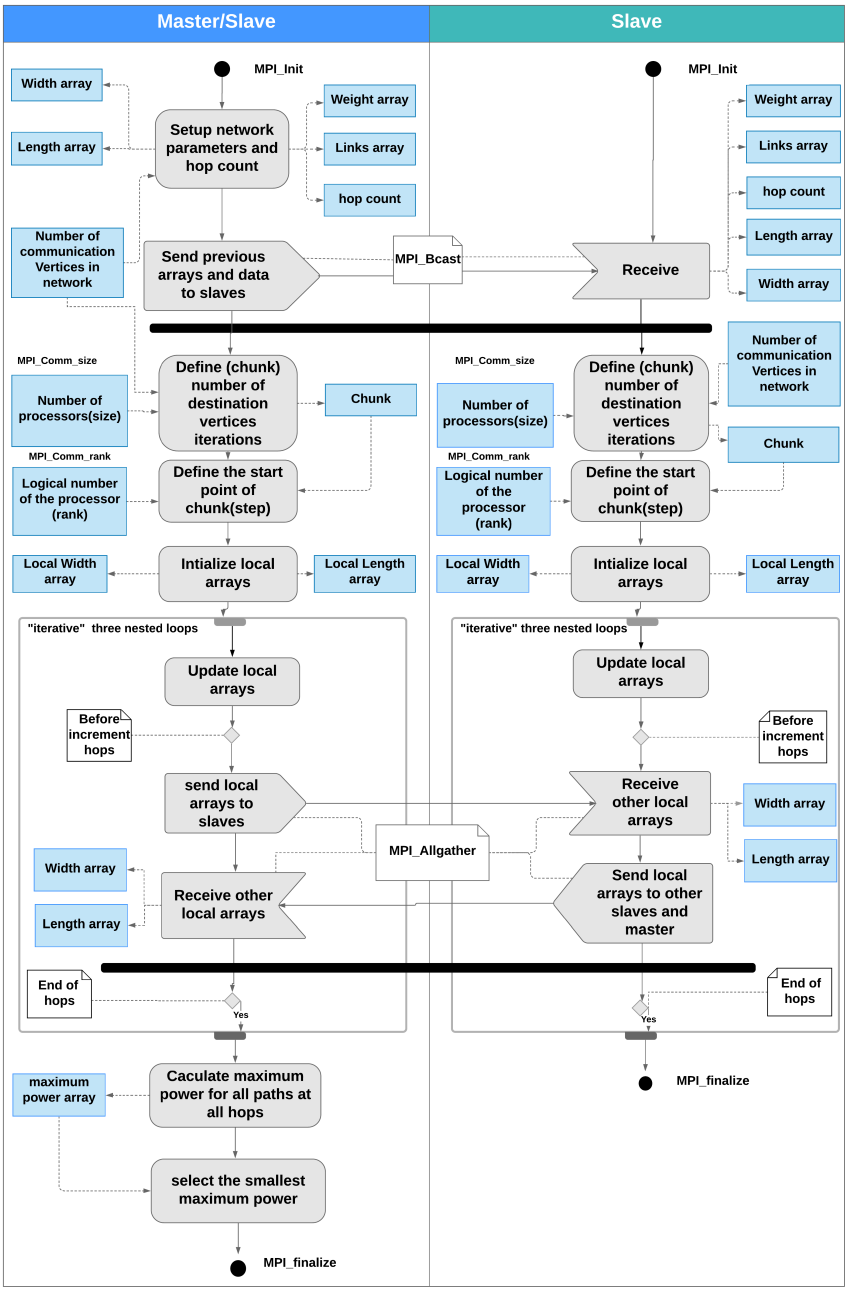


FIGURE 6. MPI master-slave model for parallel multi-hop routing.

Xeon E5-2670), each running at 2.6 GHz, where each processor has 16 threads. One compute server has two 10-core processors (Intel(R) Xeon E5-2640), each running at 2.4 GHz, where each processor has 20 threads. The remaining compute server has two 8-core processors (Intel(R) Xeon E5-2680), each running at 2.7 GHz, where each processor has 16 threads as shown in Table 2. The internal network has 10GB Ethernet switches. The Non-Uniform Memory Architecture (NUMA) design is used in a distributed-memory multiprocessing system, where memory access time relies on the location of the memory relative to the processor. Under NUMA, a processor can access its local memory faster than non-local memory (local memory to another processor or shared memory between processors) [44]. Our MPI parallel implementation on the virtual cloud cluster operates on eight VMs provisioned and managed by the bright OpenStack hypervisor [42]. This virtual cluster is characterized by the following:

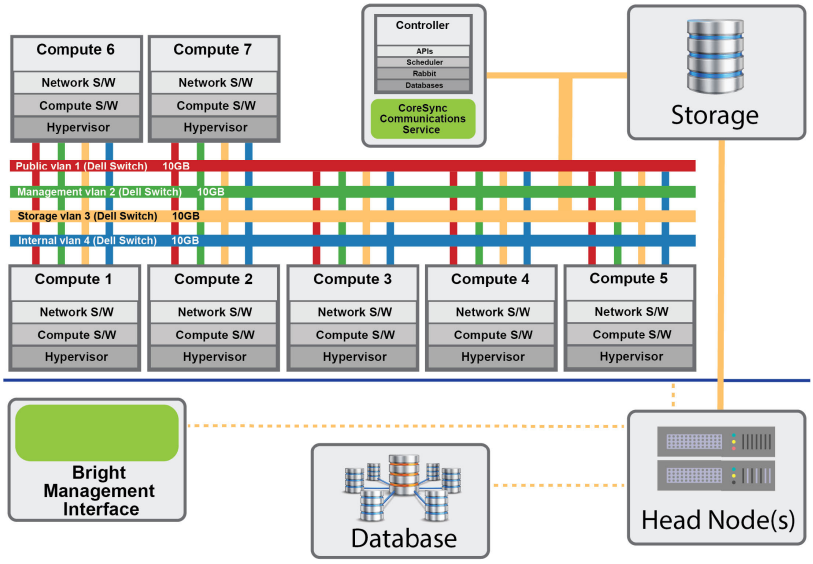


FIGURE 7. ERI Cloud system.

- Each virtual machine has 32 VCPUs to nearly meet the specifications of HPC slave servers, 7.8 GB RAM, and Centos 7.
- Virtual machines are interconnected using the same network (OpenStack private network).
- MPICH package is installed on all VMs

To begin, we set up passwordless ssh communication between VMs. Then, we configure Network File System (NFS) protocol services [45] on NFS-server (one of VMs) and NFS-clients (other VMs) to use the NFS shared folder between VMs. NFS is a stateless distributed file system protocol to allow users and applications to access and process remote data in the server as if it were local data. Finally, we mount the shared folder and install the MPICH (Message passing interface Chameleon) packages version 3.2.1 [46] on the same shared folder.

B. PERFORMANCE METRICS

We test our parallel implementation using deployment parameters that are classified into two main categories (application, computing platform).

1) Communication Network densification (Application parameter): Our parallel application runs for different network sizes which indicate the number of communication nodes (N). The memory requirements for various network sizes are shown in Table 3.
2) Cores per node (Computing platform parameter): The number of processors or cores running on a single compute node in a computing cluster is represented by this parameter. High-performance computing (HPC) servers or cloud-based virtual machines (VMs) are used to distribute MPI tasks. To keep things simple, an HPC server or cloud-based virtual machine is referred to as a node. The term *cores per node* refers to the number of processors or cores or virtual compute units (VCPU) that may be found on a HPC server or cloud VM, respectively. There are two techniques for distributing MPI tasks:
   - The Uniformly Distributed Scenario (UDS): When using MPI tasks, the MPI tasks (processes) are distributed in a round-robin fashion among available nodes by default (i.e. one task is assigned to

TABLE 2. Technical specifications for ERI-Cloud servers.

SPECs	Controller server	Compute server
Physical server/processors	2X(Intel (R) Xeon (R) CPU E5-2640 0 @ 2.4 GHZ, 25MB cache, 8 cores, 20 threads)	• 5 Compute servers 2X(Intel (R) Xeon (R) CPU E5-2670 0 @ 2.6 GHZ, 20MB cache, 8 cores, 16 threads) • 2 Compute servers 2X(Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz,25 MB cache, 10 cores, 20 threads)
Memory	DDR3 128GB RAM, NUMA architecture	DDR3 128GB RAM, NUMA architecture
Interconnect	2X 10GB Ethernet per controller server	1X 10GB Ethernet per compute servers
System software(OS)	Centos 7 and Hypervisor KVM	Centos 7 and Hypervisor KVM

TABLE 3. Data sizes and their system memory requirements.

Data size	Network size (Number of communication nodes)	System memory footprint (GB)
Small	512	5.27
Medium	1024	6.31
Large	2048	7.44

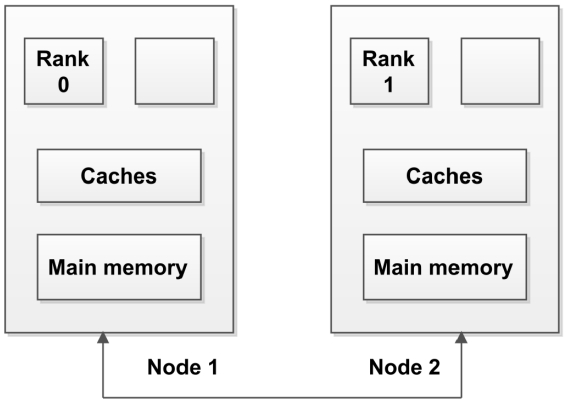
each node when several tasks are being executed simultaneously) as shown in Fig. 8(a).

- The Consolidated Distributed Scenario (CDS): Prior to moving on to the next node, each node is loaded with MPI jobs until its processors or cores are fully utilized as shown in Fig. 8(b). Because most nodes have 32 processing units, the number of MPI tasks that may run concurrently on a single node is limited to 32.

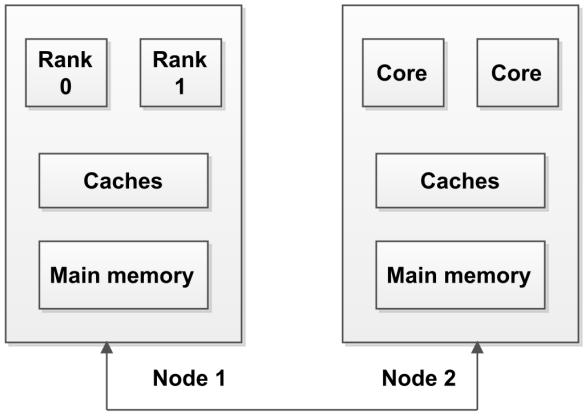
The UDS has the benefit of spreading computing effort for MPI tasks as the number of nodes increases, then reducing the computation time. However, the remote memory communication overhead across nodes is a drawback in this scenario. With more MPI tasks, the CDS requires less remote memory transfer. Unlike the previous scenario, CDS has the drawback of inter-node memory access overhead. It is possible for the memory on a single node to become overloaded, resulting in a slowdown. Table 4 illustrates the distribution of nodes and cores per node against MPI jobs in two scenarios (UDS, CDS).

TABLE 4. MPI tasks distribution in UDS and CDS scenarios.

MPI	UDS		CDS	
Tasks	*Number of nodes*	*Number of cores per node*	*Number of nodes*	*Number of cores per node*
1	1	1	1	1
2	2	1	1	2
4	4	1	1	4
8	8	1	1	8
16	8	2	1	16
32	8	4	1	32
64	8	8	2	32
128	8	16	4	32
256	8	32	8	32



(a) The Uniformly Distributed Scenario (UDS)



(b) The Consolidated Distributed Scenario (CDS)

FIGURE 8. Two MPI tasks (Rank 0, Rank 1) distribution example for both scenarios.

VII. RESULTS AND ANALYSIS

A. VIRTUAL CLUSTER ON THE CLOUD

Our parallel multi-hop routing protocol implementation is evaluated by measuring the relative execution time (computation time and communication time) and speed up compared to the serial implementation.

- The computation time is the sum of measured time for updating local width, length arrays for each hop count iteration by all processing elements (p). Then, the relative computation time is the computation time for processing elements to the total time taken on a single processing element in microseconds.
- The communication time is the sum of measured time for all-to-all gathering local arrays for each hop count iteration between processing elements (p). Then, the relative communication time is the communication time for processing elements to the total time taken on a single processing element in microseconds.
- Speedup is defined as the ratio of the total time (T_s) taken on a single processing element to the total time (T_p) required to solve the same problem on a parallel computer with identical processing elements (p).

In Fig. 9 and Fig. 10, the x-axis is the independent variable in our experiments which indicates the number of running MPI tasks. Then, the number of nodes and the number of cores per node are explicitly presented by Table. 4. The y-axis is the dependent variable that represents relative execution time. The results are semi logarithmic-scaled for the x-axis only, using base-10 logarithm.

The relative execution time is shown in Fig. 9 for small data size in UDS and CDS scenarios. Due to the virtualization overhead and latency overhead among nodes (VMs), increasing the number of MPI tasks up to 8 causes a slight increase in the relative communication time for UDS. Then, the communication time increases significantly when the number of MPI tasks exceeds 32. The percentage of change rate in communication time from 8 MPI tasks

to 64 MPI tasks represents more than 100% increase due to communicating large number of nodes. Also, increasing the number of VMs requires high access for the physical cores. Therefore, the virtualization overhead increases due to massive context switches and small data transfers among VMs.

For CDS, the change rate of communication time decreases by 50% for up to 32 MPI tasks since one VM can still tolerate running MPI tasks with insignificant virtualization overhead. Because the number of running MPI tasks on the cloud server exceeds the number of physical cores, the communication time increases as the number of MPI tasks exceeds 32. The virtualization effect and inter-VM communication becomes significant to access the memory among MPI tasks starting at 32 MPI tasks. The higher the number of utilized VMs per server, the higher the virtualization overhead.

This virtualization overhead becomes more significant when the number of running MPI tasks exceeds the number of physical cores in the cloud server. From these results, we observe that the rate of change of CDS communication time for small data sizes can be reduced by nearly 50% at 32 MPI tasks. In both scenarios, the computation is performed in a fraction of the time, which means 100 times faster than the serial execution. After running 32 MPI tasks, UDS takes less time to compute than CDS because the overhead of context switches increases across MPI tasks running on the same server.

Fig. 10 shows the relative execution time in both scenarios (UDS, CDS) for large data size. For UDS, the communication time decreases up to 32 MPI tasks, unlike small data size. The communication to computation ratio is lower in large data size since each node spends more time processing than communicating. The change rate in communication time from 8 MPI tasks to 64 MPI tasks represents only a 40% increase, since each MPI task handles a significant volume of data. Beyond 64 MPI tasks, the communication time increases significantly like small data size because of more virtualization overhead.

For CDS, the change rate of communication time decreases by nearly 70% up to 32 MPI tasks. The communication time increases dramatically as the number of MPI tasks exceeds 32, in a similar manner as for small data size. At 256 MPI tasks, the computation time is more than 100 times faster in both scenarios (UDS and CDS). After running 32 MPI tasks, UDS takes less time to compute than CDS because increasing the number of MPI-tasks on the same cloud-server causes more memory overhead. Besides, each node handles a subset of data than distributing the computational job among them. From the results, we observe that the virtualization overhead in large data is less than that in small data because of the lower communication to computation ratio. Therefore, using a large data size can enhance execution time in both UDS and CDS.

For various data sizes in the UDS and CDS, Figs 11(a) and 11(b) illustrate the overall speed up while increasing the number of MPI tasks and nodes. For UDS, the speed up scales sub-linearly up to 8 nodes per 8 MPI tasks for all data sizes. We observe that all data sizes have their peak at 8 nodes per 32 MPI tasks. Beyond 64 MPI tasks, the speed up drops since the context switches increase among MPI tasks running on the same cloud-server VMs. Accordingly, the network virtualization overhead increases. The large data size achieves the best speed up of 9.8x while running 32 MPI tasks per 8 VMs since the communication overhead for large data size is limited due to infrequent communication of nodes. Therefore, large data is less latency-sensitive than other data sizes.

For CDS, the speed up is almost linear up to 8 MPIs per VM. While running 16 MPI tasks per one VM, the small data size requires less memory on the same server. The number of MPI tasks surpasses the number of physical cores in any server when using MPI tasks up to 32. The huge context switches for large and medium data sizes have less overhead than small data size. However, the medium data size has the best speed up of 15.8X on one VM since it is less memory-sensitive than the large data size. With increasing the number of VMs from 2 to 64 MPI tasks, the larger data has less latency overhead and virtualization overhead than other data sizes. We observe that the memory overhead noticeably affects the performance scaling in the CDS scenario for large data size. Also, large data size achieves more speed up than other data sizes in the UDS scenario, since the communication overhead is limited due to infrequent communication of nodes (less latency). Therefore, for medium data size, the CDS scenario is the best fit achieving a net speed up of 15.5x. For large data size, it is better to adopt UDS task distribution scenario.

To efficiently utilize all capabilities of cloud computing, a set of experiments with different configurations are performed, to show the best behavior of our parallel implementation on the OpenStack virtual platform. Table. 5 shows the experiment cases with five different configurations. Fig. 12 shows the relative execution time of VMs configurations on the same server or different servers with a variety of assigned VCPUs per VM to run 32 MPI tasks for large data size. We select 32 MPI tasks to compare the utilization of one VM with other configurations that are represented in cases as shown in Table 5. The case 2 uses 4 VMs on the same cloud server while case 3 uses 4 VMs on different cloud servers. The case 4 uses 2 VMs on the same cloud server while case 5 uses 2 VMs on different cloud servers. All cloud VMs

TABLE 5. Different experiments for VMs configurations.

Experiments	Number of VMs	Number of VCPUs
1	1	32
2	4 (same cloud server VMs)	8
3	4 (different cloud server VMs)	8
4	2 (same cloud server VMs)	16
5	2 (different cloud server VMs)	16

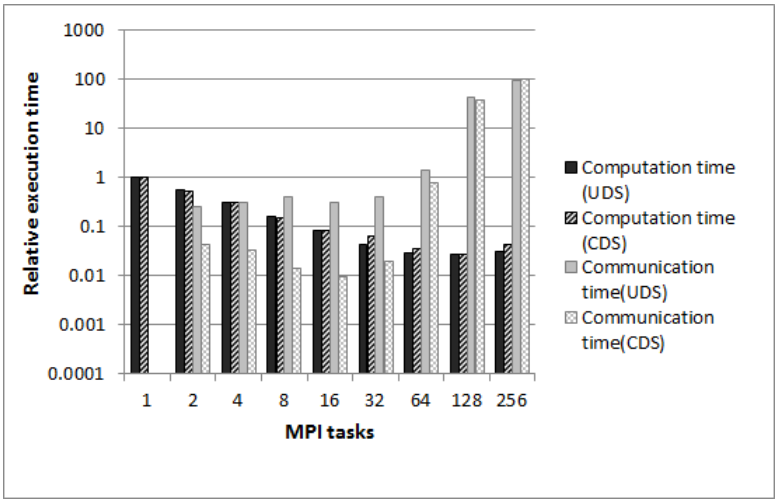


FIGURE 9. Small data size relative execution time in UDS and CDS.

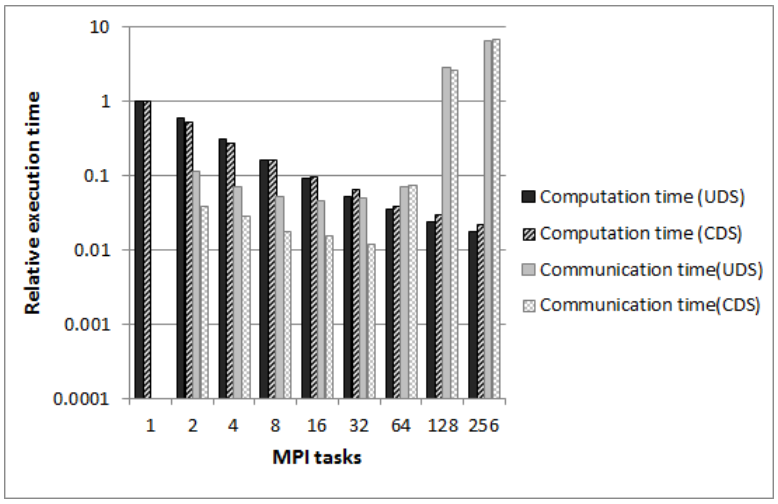
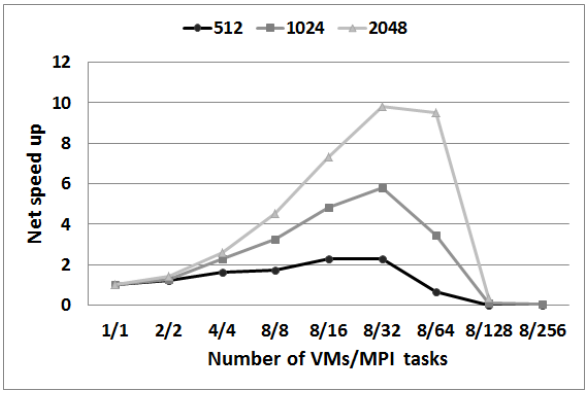
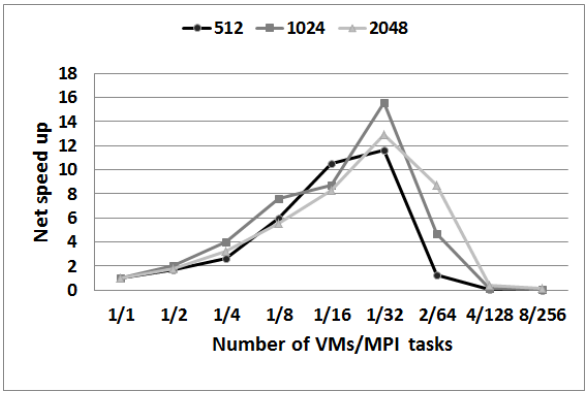


FIGURE 10. Large data size relative execution time in UDS and CDS.

are in the same internal network in all cases. We observed that the best configuration case in relative execution time is case 1 and the worst configuration case is case 2 because the communication time increases while increasing the number of VMs per cloud server. Accordingly, the context switches increase among MPI tasks running on the same cloud-server VMs and the memory-access overhead in case 2. Therefore, case 1 has the least communication time in all cases. In the different cloud-server VMs configurations (case 3, case 5), the virtualization effect in communication time is less than other cases (case 2, case 4), while increasing the number of VMs. This is due to the low latency for large data transfers. The computation time decreases using a smaller number of VCPUs that does not exceed the number of physical cores in any running cloud server. Therefore, case 1 is the worst computation time due to using 32 VCPUs in one cloud server VM. Also, case 3 and case 5 achieve the best computation time in all cases because of the smaller number of used VCPUs. Therefore, we conclude that our MPI parallel protocol uses collective communication operations among MPI tasks that require low latency. Therefore, it experiences higher virtualization overhead when running on the same cloud server, and this overhead increases as the number of VMs per server increases. For large data size, the best recommendation for VMs configurations is to be on different cloud servers with a suitable number of VCPUs regarding the physical cores in cloud servers while using more than one VM. For small data size, the communication time is more significant on the same cloud server than large data size because the number of running MPI tasks on the same cloud server exceeds the number of physical cores as shown in Fig. 9. Accordingly, the virtualization overhead increases due to massive context switches and small data transfers among VMs. Then, the best recommendation is to use the small data size case while using a number of VCPUs less than physical cores in the cloud server.



(a) Overall speed up in the UDS scenario.



(b) Overall speed up in the CDS scenario.

FIGURE 11. The overall speedup vs MPI tasks per VMs for various data sizes.

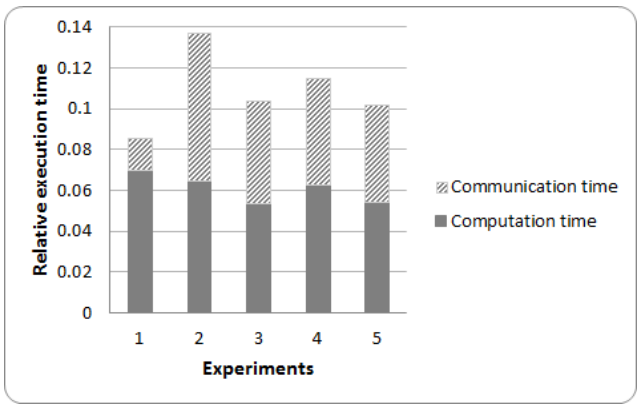


FIGURE 12. Relative execution time of different VMs configurations using the large data size (2048).

B. HPC AND CLOUD COMPUTING RESULTS COMPARISON

The major purpose of this section is to compare the performance of virtual platforms on OpenStack private cloud, the most popular open-source cloud platform, to conventional parallel platforms (HPC) results presented in [19]. To this end, we compare the performance of a virtual cluster of several VMs running the MPI implementation to the HPC system described in [19], as to better understand the impact of network latency on performance. Then, we evaluate the scalability of both platforms and set recommendations for them to be utilized by our 5G parallel protocol. To compare performance and scalability for various network sizes and MPI task distribution scenarios (UDS and CDS), Fig. 13 and Fig. 14 are shown, respectively.

In Fig. 13, for all data sizes up to 32 MPI tasks, we notice that HPC and the cloud perform similarly in terms of speed up. However, HPC has better speed up since the cloud has the virtualization overhead besides latency overhead. The large data size is the most dominant data size to achieve peak speed up for both platforms in the UDS scenario. The speed up of HPC is double the speed up of the cloud for the large data size up to 32 MPI tasks. The large data size is less latency-sensitive than other data sizes, especially when increasing the number of physical cores per node and the number of nodes at running 128 MPI tasks. We observe that HPC could efficiently utilize more resources in terms of the number of cores and nodes as compared to the cloud, due to the absence of any virtualization overhead in the UDS scenario, especially for the large data size. Large data size could efficiently utilize the cloud resources (nodes and cores) in UDS up to 8 VMs with achieving a speed up of 9.8x. The peak speed up in HPC is four times the peak speed up in the cloud. The peak cloud speed up is 9.8x when running 32 MPI tasks, and the HPC speedup is 37.4x when running 128 MPI tasks.

In Fig. 14, we observe that the peak speed up of HPC and the cloud occurs with medium data size when running one node in the CDS scenario. Beyond 32 MPI tasks per node, only HPC achieves speed up when increasing the number of nodes up to 4 in the case of large data size. The cloud results are inferior to the HPC results due to network virtualization and context switches overhead among MPI tasks of VMs deployed in the cloud server. In a CDS scenario and large data size, HPC could utilize additional resources, such as more nodes and cores. We also observe that the medium data size case achieves the best speed up of 15.5x in the cloud when running 32 MPI tasks per VM (CDS). Finally, we could conclude that choosing the best platform for our parallel implementation protocol is highly dependent on changing the application parameters (network size), and accordingly the computing platform parameters (MPI tasks distributing scenarios). HPC outperforms the cloud in terms of scalability, while achieving speed up for large data size. We also conclude that cloud computing could be utilized by medium data size without distributing MPI tasks (CDS per VM) to achieve a peak speed up of 15.5x. In terms of efficient utilization of all capabilities, large data size can utilize cloud resources up to 8 VMs with achieving a peak speed up of 9.8x in UDS scenario per 32 MPI tasks. The virtual cloud cluster could enhance the performance for medium and large data size using fewer resources (one VM) as compared to HPC. Therefore, the cloud outperforms HPC since it utilizes less hardware resources in 5G BBU pool deployment, as well as the ability to assign cloud processing resources closer to mobile users and real-time applications

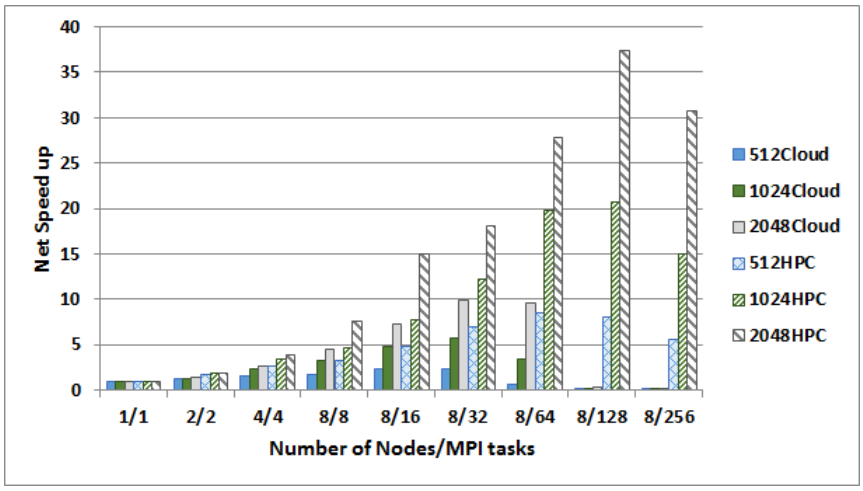


FIGURE 13. The overall speed up vs MPI tasks per nodes for HPC and cloud platforms using UDS scenario.

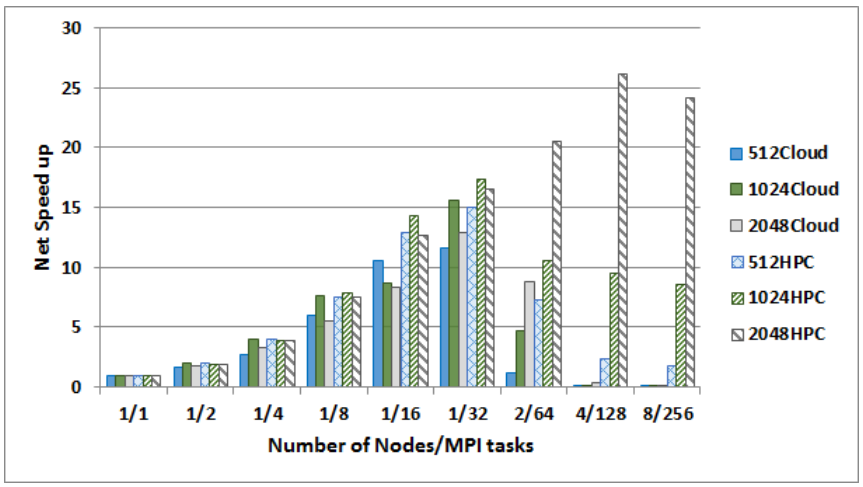


FIGURE 14. The overall speed up vs MPI tasks per nodes for HPC and cloud platforms using CDS scenario.

at the network edge. Then, MNOs have the benefits of cloudification and virtualization of 5G network resources for C-RAN.

VIII. CONCLUSION AND FUTURE WORK

This paper highlights the potential usefulness of utilizing cloud computing in an attempt to speed up 5G routing protocols and satisfy the needs of 5G UDNs. Cloud computing provides many advantages for 5G networks, such as centralized processing and management, elastic provisioning, and dynamic scalability upon-demand. The major objective of this research is to compare the performance of virtual cluster on the cloud to the conventional HPC platform and set recommendations for adopting both platforms. One of the most significant findings in this paper is that the best routing speed-up achieved using our cloud computing implementation was 15.5x, and was recorded using a few hardware requirements (one VM) for a medium network size of 1024 nodes. By comparing the results of our protocol for both platforms with the original serial algorithm, we conclude that there is a marked improvement in running bigger network sizes with less execution time. To set recommendations for the adoption of cloud computing and HPC platforms to be highly utilized in terms of speed up, scalability, and resources or infrastructure requirements in 5G deployment, a set of experiments are done using different application and system parameters (network size, number of cores and nodes). We conclude that our parallel protocol is a latency-sensitive application, and cloud virtualization adds more overhead. Accordingly, the cloud outperforms HPC that it could be deployed at the edge of medium to large networks to serve mobile users and real-time applications, yielding a peak speed up with a few hardware requirements (one VM). We also found that the location of VMs and the number of VCPUs per VM in the cloud server play a crucial role in determining the latency and virtualization overhead for different network sizes. Therefore, deploying virtual cloud clusters necessitates enhancing the cloud resource management to reduce network

virtualization overhead, especially for our parallel 5G routing protocol. We conclude that our HPC cluster implementation surpasses that of a virtual cloud cluster in terms of routing speed-up of 37x for a large network size with 2048 nodes. Therefore, HPC is more suitable than the cloud in terms of higher linear parallel performance, scalability, and platform utilization at the core network of 5G.

In the future work, various parallel implementations, such as shared memory (OpenMP) and hybrid multiprocessing programming, may be useful to assess the performance gains in both platforms. Also, cloud computing is efficient when multi-core processor speeds increase, MPICH software versions facilitate the in-compute server communication, and the container virtualization is used to reduce the latency of the VM virtualization layer.

ACKNOWLEDGMENT

This work was supported in part by the Cloud Computing Center of Excellence through the Science and Technology Development Fund (STDF), Egypt, under Grant 5220; and in part by the Distributed and Networked Systems Research Group Operating through the University of Sharjah, United Arab Emirates, under Grant.

REFERENCES

[1] M. Nazir, "Cloud computing: Overview & current research challenges," *IOSR J. Comput. Eng.*, vol. 8, no. 1, pp. 14–22, 2012.

[2] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and bus. Applications: Taxonomy, vision, and research challenges," *ACM Comput. Surveys*, vol. 51, no. 1, pp. 1–29, Jan. 2019.

[3] P. Mvelase, H. Sithole, S. Masoka, and M. Bembe, "Hpc in the cloud environment: Challenges, and theoretical analysis," in *Proc. Int. Conf. Sci. Comput. (CSC)*, 2018, pp. 94–101.

[4] A. Gupta, L. V. Kale, F. Gioachin, V. March, C. H. Suen, B.-S. Lee, P. Faraboschi, R. Kaufmann, and D. Milojicic, "The who, what, why, and how of high performance computing in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, vol. 1, Dec. 2013, pp. 306–314.

[5] M. Peng, Y. Li, Z. Zhao, and C. Wang, "System architecture and key technologies for 5G heterogeneous cloud radio access networks," *IEEE Netw.*, vol. 29, no. 2, pp. 6–14, Mar. 2015.

[6] P. T. Dat, A. Kanno, and T. Kawanishi, "Radio-on-radio-over-fiber: Efficient fronthauling for small cells and moving cells," *IEEE Wireless Commun.*, vol. 22, no. 5, pp. 67–75, Oct. 2015.

[7] Cisco. (2020). *Ciso Annual Internet Report(2018-2023).* [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html,

[8] W. Feng, Y. Li, D. Jin, L. Su, and S. Chen, "Millimetre-wave backhaul for 5G networks: Challenges and solutions," *Sensors*, vol. 16, no. 6, p. 892, Jun. 2016.

[9] M. Jaber, M. A. Imran, R. Tafazolli, and A. Tukmanov, "5G backhaul challenges and emerging research directions: A survey," *IEEE Access*, vol. 4, pp. 1743–1766, 2016.

[10] M. Kamel, W. Hamouda, and A. Youssef, "Ultra-dense networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2522–2545, 4th Quart., 2016.

[11] X. Xu, M. Liu, J. Xiong, and G. Lei, "Key technology and application of millimeter wave communications for 5G: A survey," *Cluster Comput.*, vol. 22, no. S5, pp. 12997–13009, Sep. 2019.

[12] S. Chen, F. Qin, B. Hu, X. Li, and J. Liu, "Ultra-dense network architecture and technologies for 5G," in *Proc. 5G Mobile Commun.*, Cham, Switzerland: Springer, 2017, pp. 403–429.

[13] R. N. Mitra and D. P. Agrawal, "5G mobile technology: A survey," *ICT Exp.*, vol. 1, no. 3, pp. 132–137, Dec. 2015.

[14] A. Gupta and R. K. Jha, "A survey of 5G network: Architecture and emerging technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015.

[15] M. A. Habibi, M. Nasimi, B. Han, and H. D. Schotten, "A comprehensive survey of RAN architectures toward 5G mobile communication system," *IEEE Access*, vol. 7, pp. 70371–70421, 2019.

[16] R. T. Rodoshi, T. Kim, and W. Choi, "Resource management in cloud radio access network: Conventional and new approaches," *Sensors*, vol. 20, no. 9, p. 2708, May 2020.

[17] M. F. Hossain, A. U. Mahin, T. Debnath, F. B. Mosharrof, and K. Z. Islam, "Recent research in cloud radio access network (C-RAN) for 5G cellular systems–A survey," *J. Netw. Comput. Appl.*, vol. 139, pp. 31–48, Aug. 2019.

[18] M. Saad, "Joint optimal routing and power allocation for spectral efficiency in multihop wireless networks," *IEEE Trans. Wireless Commun.*, vol. 13, no. 5, pp. 2530–2539, May 2014.

[19] S. A. Aboulrous, A. A. El-Moursy, M. Saad, A. Abdelsamea, S. M. Nassar, and H. Abbas, "Parallel multi-hop routing protocol for 5G backhauling network using HPC platform," in *Proc. 15th Int. Conf. Comput. Eng. Syst. (ICCES)*, Dec. 2020, pp. 1–6.

[20] T. K. Vu, M. Bennis, M. Debbah, and M. Latva-Aho, "Joint path selection and rate allocation framework for 5G self-backhauled mm-wave networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 4, pp. 2431–2445, Apr. 2019.

[21] S. M. Nejakar and P. G. Benakop, "Energy efficient routing protocol for improving lifetime in 5G networks," in *Proc. 2nd Int. Conf. Emerg. Trends Sci. Technol. Eng. Syst. (ICETSE*, May 2019, p. 8.

[22] D. Dev Misra, K. K. Sarma, U. Bhattacharjee, P. K. Goswami, and N. Mastorakis, "Optimal routing in the 5G ultra dense small cell network using GA, PSO and hybrid PSO-GA evolutionary algorithms," in *Proc. 24th Int. Conf. Circuits, Syst. Commun. Comput. (CSCC)*, Jul. 2020, pp. 39–44.

[23] S. W. AbuSalim, R. Ibrahim, M. Z. Saringat, S. Jamel, and J. A. Wahab, "Comparative analysis between Dijkstra and bellman-ford algorithms in shortest path optimization," in *Proc. Conf. Ser. Mater. Sci. Eng.*, vol. 917, Sep. 2020, Art. no. 012077. [Online]. Available: https://doi.org/10.1088%2F1757-899x%2F917%2F1%2F012077

[24] S. Vakilinia and H. Elbiaze, "Latency control of ICN enabled 5G networks," *J. Netw. Syst. Manage.*, vol. 28, no. 1, pp. 81–107, Jan. 2020.

[25] H. Riasudheen, K. Selvamani, S. Mukherjee, and I. R. Divyasree, "An efficient energy-aware routing scheme for cloud-assisted MANETs in 5G," *Ad Hoc Netw.*, vol. 97, Feb. 2020, Art. no. 102021.

[26] P. Liu, G. Xu, K. Yang, J. Ge, and Z. Kuang, "Joint routing and mobile VM selection algorithm in multihop C-RAN networks," *Int. J. Commun. Syst.*, vol. 31, no. 7, May 2018, Art. no. e3402.

[27] O. L. A. Lopez, H. Alves, R. D. Souza, and M. Latva-aho, "Hybrid wired-wireless backhaul solutions for heterogeneous ultra-dense networks," in *Proc. IEEE 87th Veh. Technol. Conf. (VTC Spring)*, Jun. 2018, pp. 1–5.

[28] B. Tian, Q. Zhang, Y. Li, and M. Tornatore, "Joint optimization of survivability and energy efficiency in 5G C-RAN with mm-wave based RRH," *IEEE Access*, vol. 8, pp. 100159–100171, 2020.

[29] R. Dogra, S. Rani, H. Babbar, and D. Krah, "Energy-efficient routing protocol for next-generation application in the Internet of Things and wireless sensor networks," *Wireless Commun. Mobile Comput.*, vol. 2022, pp. 1–10, Mar. 2022, doi: 10.1155/2022/8006751.

[30] X. Wang, J. Hu, H. Lin, S. Garg, G. Kaddoum, M. J. Piran, and M. S. Hossain, "QoS and privacy-aware routing for 5G-enabled industrial Internet of Things: A federated reinforcement learning approach," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 4189–4197, Jun. 2022.

[31] L. H. Binh and T.-V.-T. Duong, "An improved method of AODV routing protocol using reinforcement learning for ensuring QoS in 5G-based mobile ad-hoc networks," *ICT Exp.*, Jul. 2023, doi: 10.1016/j.icte.2023.07.002.

[32] H. Alqahtani, L. Niranjan, P. Parthasarathy, and A. Mubarakali, "Modified power line system-based energy efficient routing protocol to improve network life time in 5G networks," *Comput. Electr. Eng.*, vol. 106, Mar. 2023, Art. no. 108564. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790622007790

[33] B. Popa and D. Popescu, "Analysis of algorithms for shortest path problem in parallel," in *Proc. 17th Int. Carpathian Control Conf. (ICCC)*, May 2016, pp. 613–617.

[34] G. Hajela and M. Pandey, "Parallel implementations for solving shortest path problem using bellman-ford," *Int. J. Comput. Appl.*, vol. 95, no. 15, pp. 1–6, Jun. 2014. [Online]. Available: https://doi.org/10.5120%2F16667-6659

[35] F. Busato and N. Bombieri, "An efficient implementation of the bellman-ford algorithm for Kepler GPU architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2222–2233, Aug. 2016.

[36] P. Heywood, S. Maddock, R. Bradley, D. Swain, I. Wright, M. Mawson, G. Fletcher, R. Guichard, R. Himlin, and P. Richmond, "A data-parallel many-source shortest-path algorithm to accelerate macroscopic transport network assignment," *Transp. Res. Part—C, Emerg. Technol.*, vol. 104, pp. 332–347, Jul. 2019.

[37] S. Yadav and A. Khan, "SP async: Single source shortest path in asynchronous mode on MPI," 2021, *arXiv:2103.12012*.

[38] D. Pliatsios, P. Sarigiannidis, S. Goudos, and G. K. Karagiannidis, "Realizing 5G vision through cloud RAN: Technologies, challenges, and trends," *EURASIP J. Wireless Commun. Netw.*, vol. 2018, no. 1, p. 136, Dec. 2018.

[39] M. Peng, Y. Li, J. Jiang, J. Li, and C. Wang, "Heterogeneous cloud radio access networks: A new perspective for enhancing spectral and energy efficiencies," *IEEE Wireless Commun.*, vol. 21, no. 6, pp. 126–135, Dec. 2014.

[40] S. L. Graham, P. B. Kessler, and M. K. McKusick, "Gprof: A call graph execution profiler," *ACM SIGPLAN Notices*, vol. 39, no. 4, pp. 49–57, Jun. 2004.

[41] ERI. (2015). *Scientific Cloud Computing Center of Excellence*. [Online]. Available: http://www.eri.sci.eg/?q=en/Cloud_Center

[42] (2015). *Bright Openstack*. [Online]. Available: http://info.brightcomputing.com/OpenStack

[43] Rackspace Cloud Computing. (2018). *Liberty., The Twelve Release of Openstack*. [Online]. Available: https://www.openstack.org/software/liberty/

[44] A. Ali and K. S. Syed, "An outlook of high performance computing infrastructures for scientific computing," *Adv. Comput.*, Elsevier, vol. 91, pp. 87–118, 2013.

[45] G. Arnold, "Internet protocol implementation experiences in PC-NFS," in *Proc. ACM Workshop Frontiers Comput. Commun. Technol.-(SIGCOMM)*, Aug. 1988, pp. 8–4. [Online]. Available: https://doi.org/10.1145%2F55482.55485

[46] MPICH. (1992). *High-Performance Portable MPI* . [Online]. Available: http://www.mpich.org/

SOMAYA A. ABOULROUS received the B.Sc. degree in computer science and engineering from Menofia University, Cairo, Egypt, in 2014, the M.Sc. degree (Scientists for Next Generation—SNG) from the Academy of Scientific Research and Technology (ASRT), Cairo, and cooperated with the Electronics Research Institute (ERI), in 2016, and the M.Sc. degree in computer and systems engineering from Ain Shams University, in 2022. Her research interests include performance optimization, high-performance computing, and cloud computing.

AMANY ABDELSAMEA received the B.Sc. degree from the Faculty of Engineering at Shoubra, Benha University, in 1999, and the M.Sc. and Ph.D. degrees from the Faculty of Engineering, Cairo University, in 2006 and 2017, respectively. She is currently a Researcher with the Electronics Research Institute. In 2022, she was a part-time Instructor with New Cairo Technological University, Egypt. Her research interests include high-performance computing, parallel computing, distributed computing, cluster computing, grid computing, cloud computing, green computing, and big data analytics.

ALI A. EL-MOURSY (Senior Member, IEEE) received the Ph.D. degree in high-performance computer architecture from the University of Rochester, Rochester, NY, USA, in 2005. He was with the Software Solution Group, Intel Corporation, Santa Clara, CA, USA, until 2007. In 2007, he joined the Electronics Research Institute, Giza, Egypt. He has also participated with the IBM Cairo Technology Development Center, Egypt, as a Visiting Research Scientist, from February 2007 to January 2010. In September 2010, he joined the Department of Electrical and Computer Engineering, University of Sharjah, Sharjah, United Arab Emirates, as an Assistant Professor, where he was promoted to an Associate Professsor, in January 2017, and a Full Professor, in January 2023. His research interests include high-performance computer architecture, multi-core multi-threaded micro-architecture, power-aware micro-architecture, simulation and modeling of architecture performance and power, workload profiling and characterization, parallel programming, high-performance computing, parallel computing, the IoT architecture, and cloud computing.

MOHAMED SAAD (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from McMaster University, Hamilton, ON, Canada, in 2004. He was a Researcher with the Department of Electrical and Computer Engineering, University of Toronto, Canada, and the Advanced Optimization Laboratory, Department of Computing and Software, McMaster University. He is currently a Professor with the Department of Computer Engineering and the Assistant Dean of Graduate Studies with the College of Computing and Informatics, University of Sharjah, United Arab Emirates. His research interests include networking, communications, and optimization, with current activity focused on the optimal design of wireless and wired communication networks and optimal network resource management. He was a recipient of the Best Paper Award from the IEEE Symposium on Computers and Communications, Riccione, Italy, in June 2010; the University of Sharjah Annual Incentive Award for Distinguished Faculty Members, for Excellence in Research, in April 2010 (University-Wide); the two Best Teaching Awards from the IEEE Women in Engineering Society, University of Sharjah, in 2007 and 2009; and the Natural Sciences and Engineering Research Council of Canada (NSERC) Post-doctoral Fellowship, from 2005 to 2006. He is an Associate Editor of *Frontiers in Communications and Networks*.

FADI N. SIBAI received the B.S. degree in electrical engineering from The University of Texas at Austin, Austin, TX, USA, and the M.S. and Ph.D. degrees in electrical engineering from Texas A&M University. From 1990 and 1996, he was an Assistant Professor of electrical engineering with The University of Akron. From 1996 to 2006, he managed programs and engineering teams with Intel Corporation, Santa Clara, CA, USA. From 2006 to 2011, he directed the Computer Systems Design Program and the IBM Cell Competence Center, UAE University. From 2011 to 2019, he was with Aramco. He received the IBM's Highest Research Award from UAE University. In 2022, he joined the College of Engineering and Architecture, Gulf University for Science and Technology, Kuwait, as the Associate Dean. Previously, he served as the acting Dean with the School of Engineering, American International University, and the Dean of the College of Computer Engineering and Science, Prince Mohammad bin Fahd University. He has authored or coauthored more than 230 publications and technical reports and served on the organizing or program committees for more than 20 conferences. He holds PMP, CISSP, CCNA, and CQRM certifications. He is a member of PMI, (ISC)2, and Eta Kappa Nu.

SALWA M. NASSAR received the Ph.D. degree in the field of parallelism in programming languages from the Department of Electronics and Communication, Faculty of Engineering, Cairo University, in 1984. Her professional experience started, in 1974, by being a Research Assistant, with the Department of Computer and Systems, Electronics Research Institute. She became an Instructor and an Assistant Professor with the Department of Computer and Systems, Electronics Research Institute. She taught at the American University in Cairo (AUC), from 1987 to 1997. She became an Associate Professor, in 1991, and a Full Professor, in 1996. She was the Head of the EU information Point InP. She is an Ex-ERI President and the PI of the Cloud Center of Excellence and the Head of the HPCloud Group, ERI. She had 50 publications. She has led a number of European and USA-funded projects. Her research interests include parallel processing, parallel logic languages, modeling and simulation of parallel programs, distributed systems, computer networks, parallel applications, parallel virtual machines, grid computing, cluster computing, and cloud computing. She is a member of the IEEE Computer Society, the Information Technology Academia Collaboration (ITAC) Steering Committee, and a Juror in the Egypt. She received the WSIS-Award, in 2006, organized by the MCIT, Egypt, in 2006.

HAZEM ABBAS (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical and computer engineering from Ain Shams University, Cairo, in 1983 and 1988, respectively, and the Ph.D. degree from Queen's University, Canada, in 1993. He was the Chairperson with the Department of Computer and Systems Engineering, Ain Shams University, and the Dean of the Faculty of Media Engineering and Technology, German University, Cairo. He was with the Royal Military College of Canada, Kingston, ON, Canada, the IBM Toronto Laboratory, and Mentor Graphics. He is currently a Professor in computer and systems engineering. His research interests include machine learning and computational intelligence. He chaired the IEEE Signal Processing Chapter in Cairo.