IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 6, 2020, accepted April 16, 2020, date of publication May 6, 2020, date of current version May 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2992880

HPCCloud Seer: A Performance Model Based Predictor for Parallel Applications on the Cloud

ABDALLAH SAAD[1,2] **AND AHMED EL-MAHDY**[1,3]

[1]Department of Computer Science and Engineering, Egypt-Japan University of Science and Technology, Alexandria 21934, Egypt
[2]Faculty of Engineering at Shoubra, Benha University, Cairo 11629, Egypt
[3](*On leave*) Faculty of Engineering, Alexandria University, Alexandria 11432, Egypt

Corresponding author: Abdallah Saad (abdallah.saad@ejust.edu.eg)

This work was partially supported by the National Telecommunications Regulatory Authority (NTRA), Egypt, through the Project "Super-HETs: Empowering 5G Heterogeneous Networks for Better Performance."

ABSTRACT With the continual increase in the high performance computing (HPC) market share, the need for a cheaper and widely available system rather than the expensive typical HPC systems increases. A promising alternative to HPC typical systems is the cloud computing environment which is characterised by being cheap, flexible, scalable and available. However, the cloud is based on virtualization which increases the latency to access the processing and network resources due to resource sharing. This makes the cloud an unpredictable environment to long run time programs such as HPC applications. Hence, modelling and understanding performance is essential for exploiting such environment. In this paper we propose a predictor for the execution time of the message passing interface (MPI) based applications on the cloud, as they are a major class of HPC applications. The predictor is based on an analytical performance model through considering the cloud resources as a queueing network, and the parallel applications as jobs contesting for the shared resources. The prediction based on the proposed model is measured on both a cluster of bare-metal servers and on a group of virtual machines. The overall accuracy of this prediction is 88% for 10 benchmarks, 5 from SPEC-MPI and 5 from NASA parallel benchmarks.

INDEX TERMS Cloud computing, high performance computing, message passing interface, performance modeling.

I. INTRODUCTION

The interesting features of cloud computing such as availability, elasticity, usability and the 'pay-as-you-go' business model, make it suitable for a wide spectrum of applications. These features also encourage high performance computing (HPC) customers to consider the cloud as a cheaper alternative to the expensive supercomputer and HPC clusters. Furthermore, the cloud can provide better turnaround time than HPC clusters, considering the waiting time imposed by cluster management systems [1]. As a consequence, the market size of the HPC cloud increased from about one billion US dollars in 2014 to about three billion US dollars in 2019 and predicted to reach 5.5 billion US dollars in 2022 [2]. A typical example is the current strong move to utilise cloud computing in network communications technologies (as in 5G), for intense, real-time, communication operations; this move essentially allows for reducing corresponding computation costs as well as functionality through consolidation of resources, that were traditionally localised in each communication antenna (base-band units) [3].

However, migrating HPC applications to the cloud faces two main obstacles [4]: the higher network latency on the virtual environment compared to the typical HPC platforms, and the lack of information regarding the underlying hardware that hosts the virtual machines used. The high network latency lessens the scalability of tightly coupled parallel applications; where the cloud unpredictable environment makes it harder for both the customer and the cloud service provider to estimate the cost of running parallel applications on the cloud. These obstacles influenced Egwutuoha *et al.* [5] to recommend using bare-metal servers instead of virtual machines in case of tightly coupled applications.

Cloud service providers have exerted many efforts to overcome these obstacles; First, they have provided a new type of HPC virtual machines (VMs) with higher computational powers. Although, these cloud HPC VMs offer a comparable

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

performance to HPC clusters for loosely coupled applications, the performance degrades for parallel tightly coupled applications relative to that of the HPC clusters. This is due to the unpredictable performance of communication because of the cloud resource sharing concept, where the communication medium is shared among running workloads even with dedicated processing resources [6]–[9]. Then, many cloud service providers such as Amazon, IBM and Alibaba, started offering dedicated bare-metal servers for HPC customers to break into the HPC market. The offered bare-metal machines have relatively higher network speeds than the ordinary cloud instances and they are not sharing their resources in multi-tenancy environment. However, they still suffer from heterogeneity and relatively lower network speeds than their comparable servers on the HPC clusters [10]. The heterogeneity and low network speeds complicate performance prediction and hence the cost prediction of running HPC applications on the cloud. Knowing the exact cost and time of running an HPC application on the cloud would, thus, attract more HPC customers to migrate their work to the cloud. Also, this knowledge allows the cloud service providers to afford more flexible and profitable business models for their HPC customers.

This paper[1] proposes an analytical performance model that predicts the running time of HPC applications on the cloud, either on the cloud VMs or on the cloud bare-metal multicore servers. The proposed model targets MPI based applications which are an important class in the HPC domain. The model is based on queueing networks that represents the cloud's underlying shared resources and the contention of the running applications on it. The model uses two classes of input parameters to compute the expected execution time of the HPC application on the cloud. The first class is related to the workload of the HPC program running on the cloud, which includes the number of processes running in parallel, the number of communication operations done through out the program running time, and the average size of the communicated messages. The second class of parameters describes the specification and state of the underlying physical resources that hosts the execution of the HPC program, which includes CPU speed, number of available processing units, VMs specs and their current allocation, and network latency. For the CPU and network parameters, the model uses a non-linear solver to acquire them to allow for general applicability. Despite the largeness of the time needed for the workload profiling process to acquire the workload parameters and the time needed for the non-linear solver to acquire the CPU and network parameters, this parameters acquisition process is done initially for only one time. After that, the prediction can be done in a very short time for different configurations of parameters increasing the possibility to explore more of the design space.

[1]This paper extends an earlier work in progress version published in the proceedings of the 2019 ACM 11th Rapid Simulation and Performance Evaluation: Methods and Tools [11]

The proposed model helps answering several design questions such as; What is the effect of increasing the number of running processes on the same processing resources on the performance of the running program? What is the point where scaling up the parallelism of the running program is worthless from the performance/cost view point? How the distribution of processes over servers affects performance, given the communication distances between the servers and the heterogeneity of servers processing powers? Is it performance/cost worthy to run a given program on bare-metal servers on the cloud or to run it on an IaaS VM cheaper service?

The verification of the proposed model is conducted via two sets of experiments, both used five different benchmarks of the SPEC MPI-2007 benchmarks suite [12] as the experiment workloads. The first set measures the model accuracy when running the workloads on three different clusters sizes of cloud bare-metal servers of two, four and eight servers. The second set of experiments measure the accuracy of the model running the same workloads but on different clusters of virtual machines hosted on the same bare-metal servers used in the first set of experiments. The experiments show that the model prediction for the SPEC benchmarks has an average accuracy of 87 % compared to the actual measured execution times.

This paper has the following contributions:

- An analytical performance model for the tightly coupled HPC applications running on the cloud. The model predicts the execution time of MPI-based applications, under various hardware and software configurations.
- The model's accuracy analysis on an actual cloud virtual environment as well as on the cloud bare-metal servers, using five different benchmarks from the SPEC MPI-2007 benchmarks suit [13].
- A fair comparison with a closely well-known [14] related work [15] is provided, comparing five kernels from NASA parallel benchmarks suite (NPB) [16] on both the cloud virtual and physical machines.
- A comparison between the execution times of the benchmarks on physical and virtual machines, accompanied by a thorough analysis to the causes of this difference.
- The effect of the virtualisation on the proposed model's CPU and NW parameters is investigated.

The rest of this paper is organised as follows: related work is presented in Section II. Section III describes the proposed performance model and its corresponding parameters acquisition. The methodology used in the modelling process is depicted in Section IV. Section V presents and discusses the model validation experiments. Finally, Section VI concludes the paper and discusses future work.

II. RELATED WORK

The prospects of using the cloud as a more affordable option in place of traditional HPC clusters have been studied many times recently. For instance, Ramakrishnan *et al.* [6], Li *et al.* [7] and Zhai *et al.* [8] studied the possibility of using cloud computing for HPC and e-Sience applications.

Wang and Ng [9] studied the virtualization effect on the performance of the Amazon EC2 data centre's network. Gupta *et al.* [17] studied how HPC applications perform differently on different underlying systems such a cluster, a grid or a private cloud. These studies show that even when the underlying system consists of dedicated high performance machines packed with a 10 Gigabit Ethernet network, the communication is still a bottleneck. Also, they show how the unstable and unpredictable performance increases the cost of long running applications in a dynamic environment; e.g. HPC applications running on a virtualization environment. Marathe *et al.* [1] performed a comparison between the performance of the cloud and HPC clusters, based on a number of metrics, including turnaround time and cost, instead of execution time. They use different scales of workloads to evaluate performance, turnaround time and cost metrics among 5 different clusters and a group of Amazon EC2 cluster compute instances. The results show that communication intensive applications perform poorly on Amazon EC2. On the other hand, the cloud queue waiting time to start the service could be several times faster than the queue wait time in an oversubscribed cluster, which can badly affect the turnaround time for some applications.

Another line of research is concerned with predicting the performance of HPC applications running on a cloud system. Shi *et al.* [15] propose a methodology to analyse program scalability using complexity analysis assisted by instrumentation. Amdahl's and Gustafson's laws are the basis for this methodology. They used their methodology to predict the execution time of running parallel programs on both HPC cluster and cloud system. A detailed comparison between our proposed work and this work is provided in Section. V-C.2.

Egwutuoha *et al.* [5] recommend running tightly coupled parallel applications on a cluster of bare-metal servers on the cloud. This arrangement allows getting the advantage of the small queue waiting time before starting the requested service (higher cloud availability), while avoiding the slower network of the cloud (compared to HPC cluster) caused by cloud virtualization.

Barnes *et al.* [18] propose a modified regression model to predict parallel programs' scalability. They run a number of programs on different number of processing nodes. These runs are then fed as a training set to deduce the performance of running the parallel program on processing nodes' count with untrained configuration. While they achieve relatively accurate best-fit predictions, their model lacks the flexibility needed to test and explain how changing the main parameters affects the results. Examples of such parameters include cores per server, allocation of servers within the underlying network, the current state of the network and the amount of processing power available per server currently. Bridges *et al.* [19] introduce a work-in-progress that use a simple closed queuing network model to model the essential characteristics of communication operations of MPI applications. They experiment their proposed model by simple communication benchmarks on a two multicore machines cluster. Their main focus is to model the communication operations, neglecting the computation operations as well as the characteristics of the hardware.

Cunha *et al.* [20] used a statistical model and a memory based learning algorithm to predict the turnaround time of HPC applications on both cloud and on-premise systems. Accounting for the uncertainty of their predictions, they introduce a tool to make the job placement decision based on either local resources or on the cloud environment. This work uses a machine learning technique to predict the turnaround time, where our proposed solution uses a queueing network model that reflects both the cloud system's computing and networking resources in mathematical equations, with parameters that explain the system and the running workload. Also, this work focuses on the turnaround time prediction for computation intensive workloads, where our proposed model is tackling the problem for the communication intensive applications.

Recently, queuing network models for performance prediction of virtual environments were proposed [21], [22]. Other studies also provided performance evaluation of some applications running on Xen [23], [24]. Bennani and Menasce [25] proposed application-aware multi-class open queuing networks to predict the response time and throughput for online and batch workloads. Hu *et al.* [26] investigated auto-regressive models that map CPU allocation to the mean response time with a fixed workload. Fuzzy logic has also been proposed to model the nonlinear relationship between a virtualized Web-server's workload and its CPU usage [27]. However, the above mentioned work did not specifically target, or propose performance models for HPC workloads on the cloud.

III. PERFORMANCE MODEL OVERVIEW

The sharing of the underlying hardware and the underlying relatively slow network would potentially result in contention on both the CPUs and network. Therefore, the model considers queueing networks for modelling such contentions. Moreover, to generalise the applicability of our model, we decompose the model into two main components: one for modelling the running workload (software), and the other for modelling the underlying system (hardware). This provides for model portability.

The rest of this section is structured as follows: We discuss the workload model in Section III-A; we then describe the underlying hardware and how it interacts with the workload model in Section III-B; and we explain how we determine model parameters in Sections III-C, III-D and III-E.

A. WORKLOAD MODELLING

We consider an application (MPI based) as a collection of n process. The process repeatedly enter the system, each time executing one cycle. During the execution cycle, the process performs computations and send/receive pair of operations. More specifically, given an application as the workload, we can express it using the following parameters:

- w: the number of (dynamic) instructions in the application,
- n: the number of processes that the application runs on, taking into consideration that each process is invoked iteratively to perform a job[2]; which is a number of instructions followed by a pair of send and receive operations,
- ϕ: the number of instructions that each process executes in each system invocation (average number of instructions executed during a single job life cycle),
- s: the number of sends per process which is equal to the number of jobs invoked to finish this process. This is according to the proposed model assumption that each job must perform a single pair of send/receive operations during one life cycle.

Given all theses parameters, the following equation models the given workload:

$$w = \phi ns(n) \tag{1}$$

Note that s and w are functions of n and they depend on the current benchmark.

Given that a sequence of computation instructions ϕ followed by a communication instruction is called a cycle, then we can consider each MPI program to be consisting of a fixed number of cycles. These cycles have a uniform distribution over the processes. Also, according to our assumption that each job has to perform a send/receive operation during its life cycle, the number of jobs per process is equivalent to the number of sends per process $s(n)$. Therefore, the total number of jobs invoked during the execution time of the workload is equivalent to $ns(n)$.

In the next subsection (§III-B), the life cycle of a job inside the processing nodes hosting the execution of the workload is presented.

B. QUEUEING NETWORK AND PROCESS LIFE CYCLE

To model the execution of an MPI program on a cluster of k processing nodes, we use a closed-queuing network system, as shown in Fig. 1. In the figure, we show the queuing network for nodes i and j (two arbitrary nodes in the cluster). They are representative of the rest of cluster nodes, as all the nodes have the same structure, sharing a centralised job pool. As mentioned before, we define a job as a single process invocation, and an MPI job pool as collection of ready jobs, where jobs initially exist.

The system has an 'IN' port and 'OUT' port through which jobs enter and exit the system, respectively; each entrance and exit for the same job forms an execution cycle. Immediately after exiting, the job is scheduled for another cycle. This process is iterative.

Each job is assigned to a processing node (e.g. node i). Executing a computation sequence (of length ϕ), the job uses the node's processing power via CPU i queue. Then a

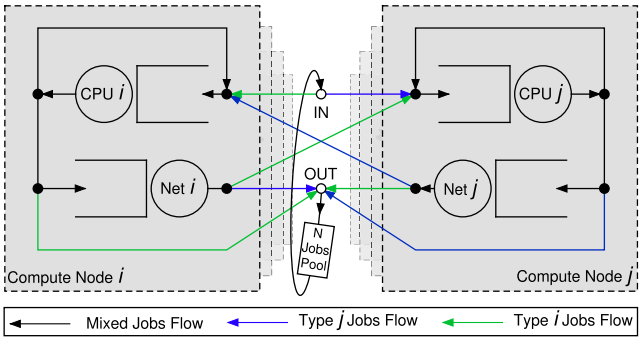

FIGURE 1. A cluster of k-machines modeled in a queueing network.

communication operation takes place between the job and another one, either located locally on the same node (node i) or remotely on another processing node, (e.g. node j). Fig. 1 shows also the job flow into our model; we used the colour codes green and blue for node i and node j jobs' paths, respectively, and black for mixed paths.

In case of local communication, the job will re-enter the CPU queue running for specific number of quanta/visits (modelling the communication overhead[3]) then exits the system back into the pool. On the other hand, in case of a remote communication operation, the operation is performed via the network queues of the two communicating processing nodes, Net i and Net j. During this remote communication, the job uses Net i to move from its locally assigned node (node i) to the remote node (node j). Net i here models the interconnection link to the remote processing node (Node j). Upon reaching the remote node (Node j), the job will enter the host CPU for a corresponding number of visits, accounting for the communication overhead needed to receive the response for its message. The job will then visit Net j queue to return back to its original node (node i), and then exits the system back to the pool, completing a job execution cycle.

In the upcoming subsections (§§III-C and III-D) we calculate the parameters needed to solve the queueing network (service time and visit ratio respectively) to get the system response time in §III-E.

C. SERVICE TIME CALCULATION

In general, the service time is the time spent by a server to serve a job from its waiting job queue. For the CPU, the service time is defined as the time taken to process (execute) the job's ϕ instructions. Hence, we can calculate the service time of the CPU as:

$$s_{\text{cpu}} = k\phi \tag{2}$$

where the constant k represents a single instruction execution time.

Given c cores in a CPU, we can deduce that increasing the number of jobs in the CPU queue enhances the service time.

[2]This is not to be confused with an MPI job; a job here refers to a queuing network job, which models a process execution cycle

[3]The communication overhead represents the time spent by a job waiting for a response from another job.

However, when the server's capacity is reached, the enhancement stops. Hence:

$$s_{\text{cpu}} = \frac{k\phi}{\min(n, c)} \tag{3}$$

where the term $\min(n, c)$ represents the maximum scalability gained using the available cores for running n parallel processes on the underlying computation resources. If we have a fixed workload w, and we substitute for $\phi = w/ns(n)$ from (1), the equation becomes:

$$s_{\text{cpu}} = \frac{kw}{s(n)n\min(n, c)} \tag{4}$$

Now we consider the network part; we define the Net queue's service time as the time that two nodes need to communicate a message of an average size between them (fixed size for each configuration). The average message size changes as the number of running processes change; thus we have to model the average message size relative to the change in the number of processes executing the workload. For this purpose, we define the following parameters:

- $m(n)$: the average message size as a function of n,
- $s(n)$: the number of send operations per process as a function of n,
- $s(n)n$: the total number of messages sent during workload execution.

Through monitoring and profiling various different workloads (10 parallel benchmarks, as mentioned later in §V-B), it was clearly obvious that the average message size of a workload's communication operations is inversely proportional to the number of running processes executing this workload. The equation $m(n) = m_a/n + m_b$ is the best model fitting this relationship; where m_a and m_b are two constants that can be calculated by fitting the profiling data of the workload which includes its communication behaviour, relative to n. Also $s(n)$ can be modelled by a logarithmic function as $C\ln(n) + D$; where C and D are two constants that can be fitted from the profiling data of the workload's communication behaviours.

Now, to calculate the the communication time of sending $m(n)$, we use a simplified cost model [28], as follows:

$$s_{\text{net}} = T_s + m(n)T_w \tag{5}$$

where:

- T_s: start-up time needed by the two communicating nodes to handle the message,
- T_w: transfer time per word.

To make the model simpler, we assume that $T_s = 0$. Hence,

$$s_{\text{net}} = (m_a/n + m_b)T_w \tag{6}$$

where T_w can be calculated as described by our Saad and El-Mahdy [29].

The model may suffer from two flaws: firstly, the model approximation be increasing the simplicity; secondly, the possible inaccuracy in the workload profiling and the hardware's resources probing. We account for these possible flaws by multiplying the right hand side of Equation 6 by a constant (Net_Constant), as follows;

$$s_{\text{net}} = \text{Net\_Constant} \times (m_a/n + m_b)T_w \tag{7}$$

where, Net_Constant is a constant factor accounts for any inaccuracy in the measurement of the network characteristics due to the possible flaws previously mentioned in this section. However, adding this constant doesn't change the model equations so much, as ($m_a \times$ Net_Constant) can be substituted by m'_a and ($m_b \times$ Net_Constant) can be substituted by m'_b.

We use a non-linear problem solver (Gauss-Newton Algorithm [30]) to compute the constants in the CPU service time equation (k and w) given a fixed size workload and the constant in the Net service time equation (Net_Constant) as described in Section. §IV. To do that, we give initial values to these constants. The outcome service time represents an initial guess to the solver, to obtain the best fit for these constants. Hence, we can define $s_{\text{CPU}_{\text{guess}}}$ and $s_{\text{Net}_{\text{guess}}}$ as follows:

$$s_{\text{CPU}_{\text{guess}}} = \frac{\text{CPU\_Constant}}{s(n)n\min(n, c)} \tag{8}$$

$$s_{\text{Net}_{\text{guess}}} = \text{Net\_Constant} \times (m_a/n + m_b)T_w \tag{9}$$

where CPU_Constant $= kw$.

D. VISIT RATIO CALCULATIONS

We define a queue's visit ratio as the average number of visits made by the workload's jobs to this queue (within their life cycles) to the total visits they do for all the queues in the queueing network.

We need to take into consideration the average CPU quanta of the job spent during its overhead processing for communication, to compute the visit ratio. First, the communication overhead is calculated; this is done by profiling the parallel application running on a physical machine with all jobs communicating locally, hence no network communication time.

Thus the time a job needs to finish its life cycle (Job_Cycle_time) consists of three parts:

- T_{comp}: computation time spent executing the ϕ instructions of the job,
- T_{nw}: network time spent by the job communicating remotely,
- T_{oh}: communication overhead time.

Thus,

$$\text{Job\_Cycle\_time} = T_{\text{comp}} + T_{\text{nw}} + T_{\text{oh}} \tag{10}$$

Hence, for each process, its cycle time is defined as: Process_time = Job_Cycle_time $\times\ s$.

To get the running application's execution time (T_{exe}) and the waiting time for all MPI communications made by each processes (MPI_Wait), we profile the parallel application by running it with a number of processes that is fewer than the available number of processing units; hence no CPU contention. We assume that all processes start and finish at the same time, so that the application's execution time is simply

equal to the execution time of a single process. Hence, we can define the cycle time of a process as Process_time $= T_{\text{exe}}$ and the Job_Cycle_time as Job_Cycle_time $= T_{\text{exe}}/s$.

The T_{nw} is neglected because the application is profiled on a single machine. Also T_{oh} from profiling can be defined as: $T_{\text{oh}} = \text{MPI\_Wait}/s$. Hence, T_{comp} is defined as $T_{\text{comp}} = T_{\text{exe}}/s - T_{\text{oh}}$.

Now that we have Job_Cycle_time, T_{comp} and T_{oh}, we can compute the ratio of the computation time to the total cycle time of a job (V_{comp}), as well as the ratio of communication overhead time to the total cycle time of a job (V_{comm}), where $V_{\text{comp}} = T_{\text{comp}}/\text{Job\_Cycle\_time}$ and $V_{\text{comm}} = T_{\text{oh}}/\text{Job\_Cycle\_time}$. After we compute these two ratios, the visit ratio of a job to each processing node's CPU queue can be calculated as follows: Given a computing node i whose CPU queue is visited by all the jobs located originally on this node to make their computations. In addition, these jobs will re-enter the CPU queue to make their communication overheads if these communications are local. Finally, the same CPU queue may be visited by jobs residing in other computing nodes if they are making remote communications with adjacent jobs on node i. Hence, we can define the visit ratio of the CPU queue of node i as:

$$V_{\text{CPU}_i} = \frac{n_i}{n} V_{\text{comp}} + \frac{n_i}{n}\frac{n_i - 1}{n} V_{\text{comm}} + \frac{n - n_i}{n}\frac{n_i}{n} V_{\text{comm}} \tag{11}$$

where:

- $\frac{n_i}{n}$: the ratio of jobs on node i to the total number of jobs in the system,
- $\frac{n_i}{n}\frac{n_i-1}{n}$: the ratio of jobs of node i that may communicate locally,
- $\frac{n-n_i}{n}\frac{n_i}{n}$: the ratio of jobs outside node i that may communicate remotely with jobs of node i.

In a similar way, we can compute the network device queue's visit ratio (V_{net_i}). The remotely communicating jobs of node i visit its network queue (net_i). Likewise, jobs that are not located on node i and communicating with its jobs remotely will visit net_i during their return. Hence, V_{net_i} can be defined as follows:

$$V_{\text{net}_i} = \frac{n_i}{n}\frac{n - n_i}{n} + \frac{n - n_i}{n}\frac{n_i}{n} \tag{12}$$

where:

- $\frac{n_i}{n}\frac{n-n_i}{n}$: the ratio of node i jobs that communicate remotely with jobs located outside the node,
- $\frac{n-n_i}{n}\frac{n_i}{n}$: the ratio of located outside node i jobs that are communicating remotely with its jobs.

E. JOB RESPONSE TIME CALCULATIONS

We define the system response time, R, as the execution time of a ϕ long sequence of instructions. As there are $s(n)$ sequences of ϕ instructions for each process to execute, the response time of each process can be defined as $s(n)\phi$. Moreover, as we assume that all n parallel sequences start and finish execution simultaneously, the total execution time of the system, T, given w instructions, can be defined as follows:

$$T = Rs(n) \tag{13}$$

We can compute the average response time needed to finish a single life cycle (R), given the service times and visit ratios. In our proposed model, a numerical solution is used to obtain R when the workload is run in parallel using n processes. Given the values of service time and visit ratio for all queues, we use Mean Value Analysis (MVA) algorithm [31] to compute R.

IV. MODEL PARAMETERS ACQUISITION

Fig. 2 illustrates the model parameters acquisition process; our main aim is to predict the execution time of a workload running on the cloud, using a given configuration. By configuration we mean which underlying resources to be used (how many cores/processing nodes and which network), the number of running processes in parallel to carry on the execution of the workload and the distribution of these processes over the used processing resources.

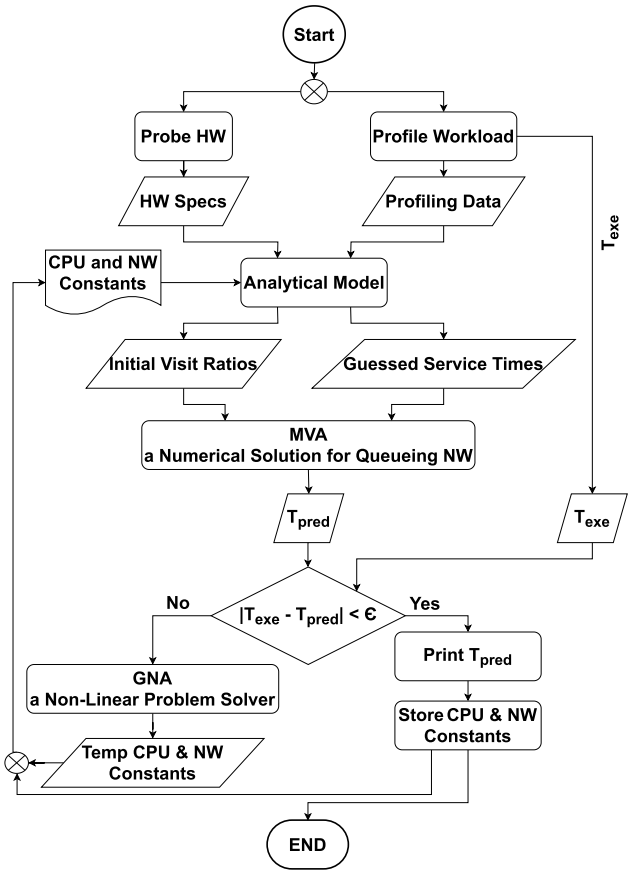

FIGURE 2. Proposed model parameters acquisition flowchart.

The first step is to profile the workload more than two times with different number of running processes for each run, using a parallel profiler, such as Vampirtrace [32] which

is an open-source library from TU Dresden, to obtain the communication pattern of the workload. At the same time, the hardware resources used (processing and networking) are probed to gather needed information about the available processing resources (processor's speed and number of available cores) and the current state of the network via a network tomographer [29]. The parallel profiler can supply us with the following information about the workload for each configuration (number of running processes) it profiled:

- number of running processes in parallel (n),
- the waiting time of each process for performing communication (MPI_Wait),
- number of send/receive operations per process ($s(n)$),
- the average message size of the communication operations performed between the running processes ($m(n)$),
- the execution time of the parallel workload (T_{exe}).

Whereas hardware probing extracts the following information:

- number of available cores for each processing node (c),
- the speed of each processing node,
- per word network transfer time (T_w),

Initially, for training purpose, we use the same configuration as one of the configurations used in the profiling. Then, through the analytical model equations, the visit ratios and the guessed service times are calculated using the workload profiling data and the hardware specifications. During these calculations, the CPU_Constant and Net_Constant initially get default values, e.g. they are both assumed to be 1. After that, the mean value analysis technique uses the initial visit ratios and service times for each queue in the queueing network to numerically calculate the system response time for the given configuration.

The predicted execution time then can be calculated using Equation 13. At this moment, we have to decide if this prediction is close enough to the ground truth measured execution time during the profiling process (difference has to be less than some constant error ϵ). If yes, then the values of CPU_Constant and Net_Constant are correct and we can consider their current values for any further not profiled configurations. If no, this means that there is imperfection in the hardware probing and workload profiling processes due to the two flaws in these processes we previously mentioned in §III-C. To handle this imperfection, the Gauss-Newton algorithm is used as a non-linear problem solver to adjust the values of both CPU_Constant and Net_Constant. Then, to test the new values of CPU_Constant and Net_Constant, we repeat the steps starting with applying these new values on the analytical model until the error between the prediction and the actual execution time becomes less than ϵ.

V. EXPERIMENTS, RESULTS AND ANALYSIS

In this section, the experiments setup, the workload used and the experiments description, results and analysis are illustrated.

A. EXPERIMENTS SETUP

To examine the accuracy of the prediction based on the proposed model and to illustrate the model capabilities, a cluster of virtual machines hosted on a private cloud and a heterogeneous HPC-cluster have been built sharing the same hardware resources. The configurations of both the cluster of virtual machines and the HPC-cluster are shown in Table 1 and Table 2, respectively. The HPC-cluster's servers have three different models of processors with different number of cores per model and different speeds. In addition, the cluster has two servers with 48 GB of memory and the rest of servers have 24 GB of memory.

TABLE 1. The Specifications of the virtual machines cluster.

Cloud Hypervisor	OpenNebula 5.4.1
Number of VM	42
Cores per VM	4
VM's memory size	4 GB
OS	CentOS Linux release 7.6.1810
Network interfaces	1 Gigabit Ethernet NIC

TABLE 2. HPC-clusters specifications.

Number of processing nodes	8
Processors' models	Intel(R) Xeon(R) CPU {E5620, E5645 and E5-2450}
Total number of cores	88
Total memory size	240 GB
OS	CentOS Linux release 7.4.1708
Network interfaces	1 Gigabit Ethernet NIC

Ten benchmarks of two different benchmark suites have been used as the parallel workload during the experiments. The first benchmark suite is the SPEC-MPI. Five benchmarks of this benchmark suite are used for assessing accuracy of the proposed model, which are: Lammps, Lu, Pop2, Socorro and Zeusmp2. The second benchmark suite is the NASA Parallel Benchmark. Also, five benchmark kernels are used to compare the accuracy of the proposed model with the prediction accuracy of one of the state of the art models. A description of the used benchmarks is given in Section V-B. Both the cloud and the cluster run MPICH 3.1 and gcc toolchain.

B. BENCHMARKS DESCRIPTION

1) SPEC-MPI

SPEC, the Standard Performance Evaluation Corporation, is a non-profit corporation established to make and maintain benchmarks to evaluate performance and energy efficiency of the modern computing systems. SPEC released many benchmark suites such as SPEC CPU, SPEC Cloud, SPEC OMP and SPEC MPI. SPEC-MPI is a benchmark suite released in 2007 and is considered a member of the high

performance group of benchmarks. This suite is designed to evaluate the performance of MPI-parallel, floating point, compute intensive native programs derived from MPI parallel end-user applications. The main performance metrics emphasised using this benchmark suite are:

- processing nodes' type and number,
- inter and intra communication interface between processing nodes, and
- the memory architecture of the processing nodes.

In our experiments, we use the following five benchmarks of SPEC-MPI benchmark suite:

1) lammps: is a parallel open source C++ molecular dynamics simulation program. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. The main communication functions between the running processes are the blocking send and the non-blocking receive. Lammps allows for up to 140 parallel running processes.
2) lu: is a parallel FORTRAN 90 program developed in NASA Ames Research Center for the use in computational fluid dynamics and computational physics fields. The program uses the blocking send and receive as the main MPI communication functions between its running processes. For the medium reference workload used in this work, lu can use at most 512 parallel processes.
3) pop2: is a parallel ocean program (pop) developed in Los Alamos National Laboratory using FORTRAN 90. The program uses the collective communication functions allreduce and waitall to communicate between the running processes.
4) socorro: is a FORTRAN 90 and C code performing self-consistent electronic-structure calculations. This code was developed as a collaboration between Sandia National Labs, Vanderbilt University, and Wake Forest University. Socorro is like lammps that uses blocking send and the non-blocking receive MPI communication functions.
5) zeusmp2: is a FORTRAN 90 computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (University of Illinois at Urbana-Champaign). The code can run using maximum 512 processes for the medium reference workload used in this work. The main communication operations performed are the non-blocking send and receive MPI functions beside the MPI-allreduce collective operation.

All the previously mentioned benchmarks used the medium reference workload during the experiments.

2) NASA PARALLEL BENCHMARK (NPB)

NPB is a computational fluid dynamics based set of applications used to evaluate the performance of parallel systems. The basic benchmark suite consists of five kernels and three pseudo applications. In this work we used the five kernels in a comparison with one of the state of the art work [15]. The five kernels are described as follows:

1) CG: use the Conjugate Gradient method to solve an unstructured sparse linear system, the blocking send and non-blocking receive are the main MPI communication operations performed during the benchmark runtime,
2) EP: Embarrassingly Parallel; a benchmark that measures the performance of parallelism with a tiny amount of communication operations,
3) FT: a numerical solver for certain partial differential equations using Fast Fourier Transform (FFT). The benchmark uses a considerable amount of communication operations to perform the FFT steps. All-to-all operation is the main MPI communication operation performed by this benchmark,
4) IS: large Integer Sort; the running processes of this benchmark communicate use the all-to-all communication operation as the main MPI communication function between them,
5) MG: A simplified 3d Multi-Grid; a memory intensive benchmark that uses the following MPI communication operations during its running time: blocking send, non blocking receive, and all-to-all communication operations.

Only IS benchmark is written in C while the other benchmarks are FORTRAN programs. Moreover, the workload used with these benchmarks are all of class-C except for FT and MG that both use class-B. Where a class designates the size of the test program; there are three classes: A, B and C; of increasing size order. Each class is approximately four times larger than the predecessor.

C. PROPOSED MODEL EXPERIMENTS

1) MODEL PREDICTION ACCURACY

Initially, the workloads used in these experiments are profiled by running 3 different configurations for each workload. For example, we profile each workload where n is 2, 4 and 8. The profiling is performed for at least 3 different configurations because it is the minimum number of data points needed to accurately model the workload communication behaviour. It is worth mentioning that once the workload has been profiled, its profiling data remains constant for any coming execution over any underlying hardware. Similarly, once the hardware has been probed, the hardware information remains constant for any future execution using the same hardware to execute any workload. In other words, the workload profiling and the hardware probing processes are performed once for any given workload or hardware used. Then, the information results from these processes is considered constant for any possible variation of their configurations.

Each profiling run is hosted on a single server that has available cores equals to the number of running processes to avoid contention on the CPU queue. From these runs we get the execution time for each and profile the performed communication operations to help modelling them for any

given number of running processes. In this communication model, for different values of n the average message size and the average number of send operations per process $s(n)$ are modelled. We use the information from the profiling process as input to the GNA nonlinear problem solver to calculate both the CPU_Constant and Net_Constant. After that, the service time and the visit ratio for each queue in our closed queuing network model are calculated. Then, the service times and visit ratios for the queues in the queuing network are inputted to the MVA algorithm to numerically calculate the response time R. Finally, the model prediction to the execution time of the parallel application for a given number of running processes n is worked out using Equation 13.

Fig. 3, Fig. 4, Fig. 5, Fig. 6 and Fig. 7 show the predicted execution time of the parallel SPEC-MPI benchmarks based on the proposed model for different number of processes (in dashed line) versus the actual measurements of the execution time of these benchmarks (in solid line). Each figure contains the data for running a benchmark on both bare-metal servers (in blue line) and virtual machines (in red line). The X-axis represents the number of running processes while the Y-axis shows the execution time in seconds for the given number of running processes. Further, the secondary Y-axis (if any) shows the execution time also in seconds for the benchmark running on the virtual machines. If the secondary Y-axis exists, the primary Y-axis shows only the execution time in seconds for the benchmark running on the bare-metal servers. The target execution time represents the ground-truth measurements for running a specific benchmark using different values of n.

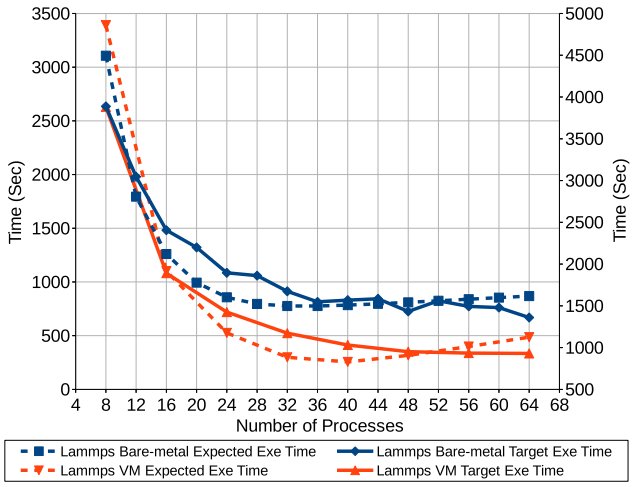

FIGURE 3. Prediction vs real measurements of Lammpas benchmark running on different number of processes.

The ground truth curves for all figures show a common behaviour related to the response of the benchmarks to increasing the number of running processes that the benchmark's workload runs on. This common behaviour can be stated as: a benchmark's execution time decreases as the number of running processes increases by a factor inversely

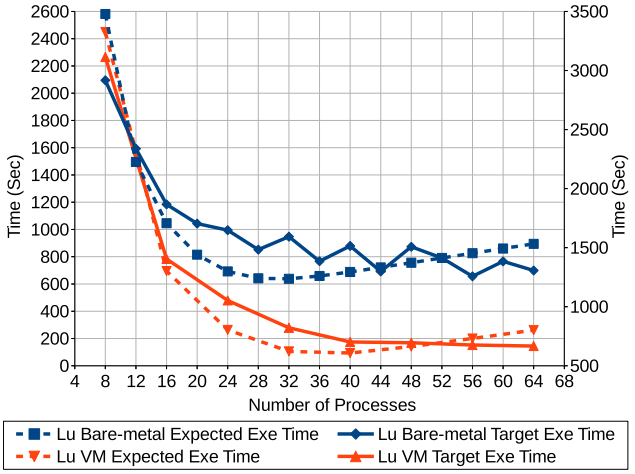

FIGURE 4. Prediction vs real measurements of Lu benchmark running on different number of processes.

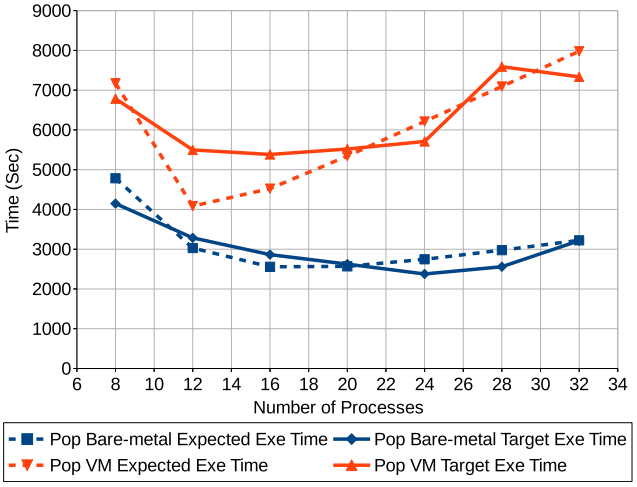

FIGURE 5. Prediction vs real measurements of Pop2 benchmark running on different number of processes.

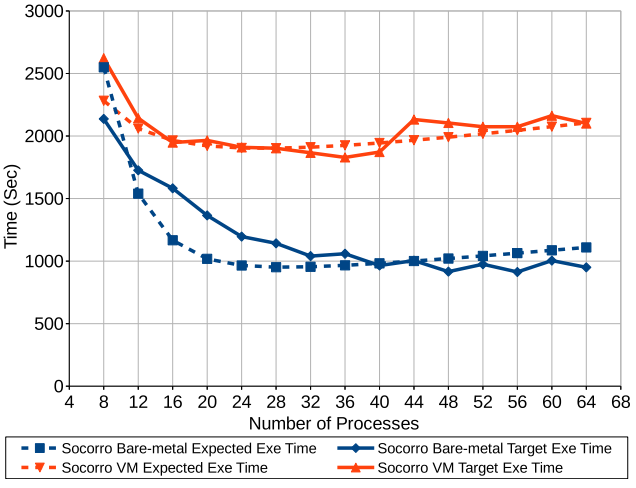

FIGURE 6. Prediction vs real measurements of Socorro benchmark running on different number of processes.

proportional to this increment. For example, doubling the number of processes for Zuesmp benchmark running on bare-metal servers from 8 to 16 processes, halves the execution

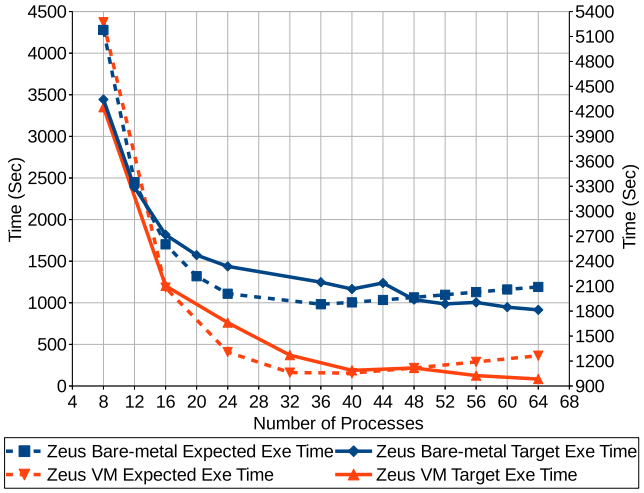

FIGURE 7. Prediction vs real measurements of Zeusmp benchmark running on different number of processes.

time from around 3400 to 1700 seconds. However, this factor decreases as the number of processes continue increasing. For example, doubling the number of processes for also Zuesmp running on bare-metal servers from 32 to 64 processes, only decreases the execution time from around 1300 to 900 seconds. The reason for this behaviour is the correlation between processing demands and communication demands. As the number of processes increases, the communication between these processes increase in a non-linear form, prohibiting the total execution time from gaining the full expected time decrements caused by the distribution of the workload among more processes.

In these experiments the evaluated metric is the model-based prediction for the parallel benchmarks execution time for a given number of running processes. In order to measure the model accuracy, two statistics are used: coefficient of variation (CV) and mean absolute percentage error (MAPE). We consider the accuracy as 100% - MAPE. The accuracy of the SPEC-MPI benchmarks are as follows; Lammps is 86% accurate while running on the bare-metal servers and 84% while running on the virtual machines, Lu is 82% on bare-metal servers and 86.5% on VM, Pop2 is 90.5% on bare-metal and 88% on VM, Socorro is 87% on bare-metal and 96.4% on VM, and finally, Zeusmp is 84.4% accurate on bare-metal servers and 86% on VM. For further details, Table 3 shows the coefficient of variation and the mean absolute percentage error for each benchmark on both bare-metal servers and VM.

The results show that the accuracy of the predictions based on the proposed model is not less than 80% for any of the five benchmarks used, either when running on bare-metal servers or on a group of virtual machines. In addition to that, from the figures we can determine the point where it is not cost worthy to increase the number of running processes/processing cores from the performance perspective. For instance, increasing the number of running processes for the benchmark Lammps running on a cluster of bare-metal servers from 8 processes

TABLE 3. Model based prediction accuracy details.

Benchmark Name	Bare-Metal Servers		Virtual Machines	
	CV	MAPE %	CV	MAPE %
Lammps	0.18	13.7	0.31	15.8
Lu	0.22	18.01	0.13	13.5
Pop2	0.11	9.5	0.14	11.7
Socorro	0.18	13.1	0.06	3.6
Zeusmp	0.2	15.6	0.24	13.8

to 36 processes, reduces the running time from 2635 seconds to 815 seconds (3.23 $\times$ speedup with baseline $=8$ and efficiency $=0.09$). Whereas increasing the number of running processes from 8 to 64 reduces the benchmark running time to 669 seconds (3.94 $\times$ speedup with baseline $=8$ and efficiency $=0.06$). From the previous calculations of the speedup and efficiency, we can conclude that increasing the number of running processes above a certain point is not cost efficient. The proposed model prediction helps in determining this point and recommends not adding more processing resources as it is not cost worthy.

2) MODEL COMPARISON WITH THE STATE-OF-THE ART

The closest research from the literature [14] to the proposed work is the work done by Shi *et al.* [15]. In their work, they propose a performance prediction methodology using Amdahl's law and Gustafson's scaled speedup formulation. Their work is measured on both an HPC-cluster and on a private cloud. The workload used is five kernels of the NASA Parallel benchmark suite. Shi *et al.'s* work and our proposed model consider the cloud infrastructure and the parallel workloads running on it, then uses a model to predict the running time of these workloads on the cloud. In this subsection a comparison between their work and our proposed model's prediction is introduced. For the sake of fairness, their model has been re-implemented on our experiments test-bed. Also, the comparison uses the same benchmark suite they used.

Figure 8, Figure 9, Figure 10, Figure 11, and Figure 12 show the results of the comparison between the running time prediction based on the proposed model (blue line) versus the prediction based on Shi *et al.* model (scalability analysis model; red line) versus the measured execution time (ground truth; black line) while running the benchmarks: CG, EP, FT, IS and MG, respectively, on a cluster of bare-metal servers. Further Figure 13, Figure 14, Figure 15, Figure 16, and Figure 17 show the results of the same comparison but with the benchmarks running on a group of virtual machines.

Overall, the results show a prediction accuracy of 74% for the scalability analysis model-based prediction (80% for bare-metal servers and 68% for virtual machines), and 88.8% for the prediction based on our proposed model (87.4% for bare-metal servers and 90.2% for virtual machines). The details of these results are presented in Table 4. Also Figure 18 visualizes the accuracy of predicting the running time of each benchmark of NPB on both bare-metal

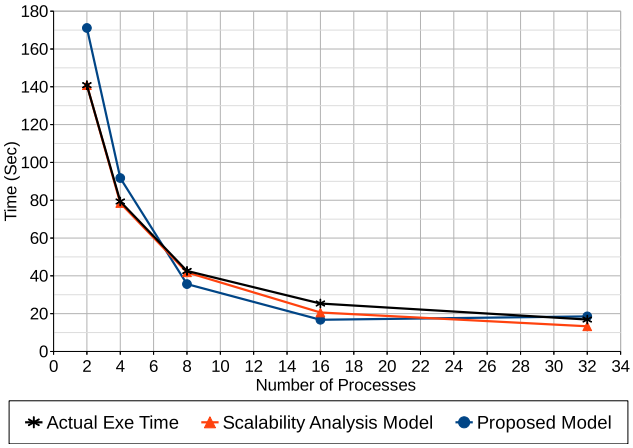

FIGURE 8. Proposed model prediction vs scalability analysis model vs real measurements of CG class-C benchmark running on a cluster of bare-metal servers.

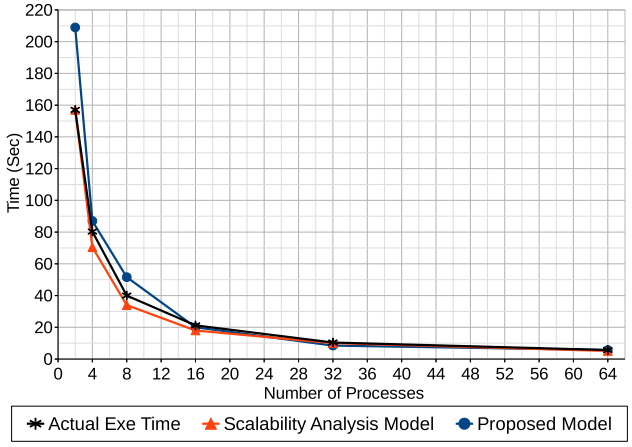

FIGURE 9. Proposed model prediction vs scalability analysis model vs real measurements of EP class-C benchmark running on a cluster of bare-metal servers.

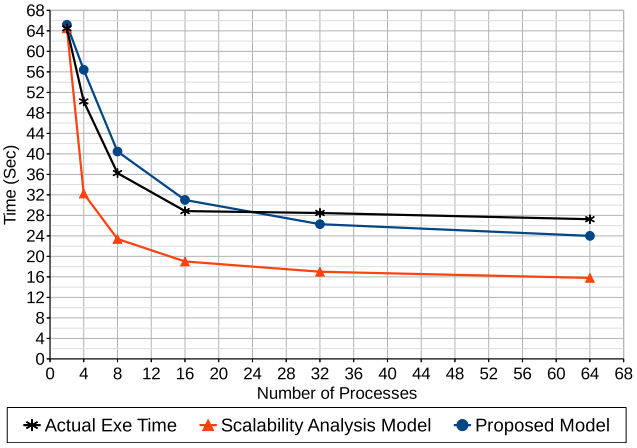

FIGURE 10. Proposed model prediction vs scalability analysis model vs real measurements of FT class-B benchmark running on a cluster of bare-metal servers.

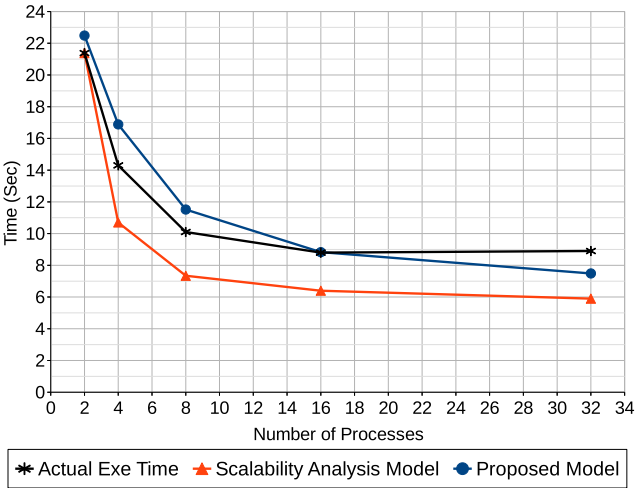

FIGURE 11. Proposed model prediction vs scalability analysis model vs real measurements of IS class-C benchmark running on a cluster of bare-metal servers.

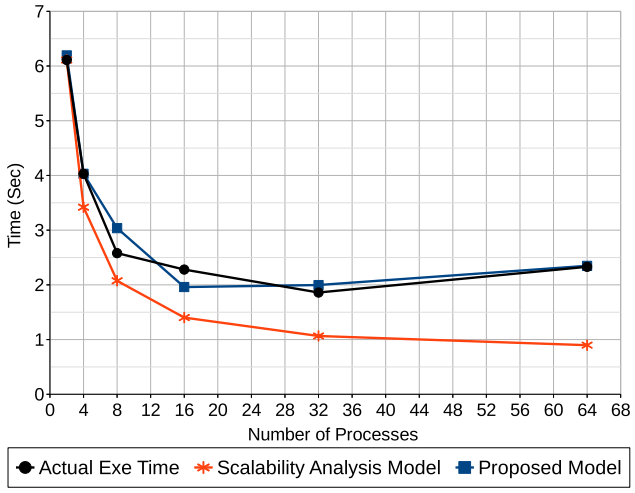

FIGURE 12. Proposed model prediction vs scalability analysis model vs real measurements of MG class-B benchmark running on a cluster of bare-metal servers.

TABLE 4. Comparison between prediction accuracy based on the proposed model and scalability analysis model.

Benchmark	Proposed Model				Scalability Analysis Model			
	Bare-metal		VM		Bare-metal		VM	
	CV	MAPE	CV	MAPE	CV	MAPE	CV	MAPE
CG	0.27	20.7	0.17	11.9	0.04	8.5	0.22	36
EP	0.38	15.3	0.19	13.1	0.09	9.5	0.16	17.3
FT	0.09	8.7	0.07	8.4	0.3	31.3	0.35	37.7
IS	0.1	10.7	0.08	8.3	0.2	22.7	0.5	52.6
MG	0.07	6.9	0.07	7	0.26	29.5	0.15	14.7

servers (solid bars) and virtual machines (dashed bars). The figure shows the prediction based on our proposed model in blue colour as well as scalability analysis model but in red colour. As can be seen from the results, the accuracy of the prediction based on both our proposed model and the scalability analysis model is very close for running the benchmarks on a maximum of 8 processes, which is the number of available bare-metal servers we used during these experiments. Using more than 8 processes to run the workload causes the accuracy of the prediction based on the

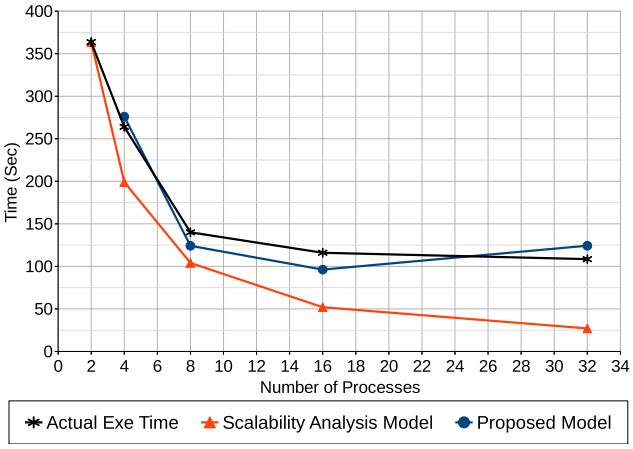

FIGURE 13. Proposed model prediction vs scalability analysis model vs real measurements of CG class-C benchmark running on a group of virtual machines.

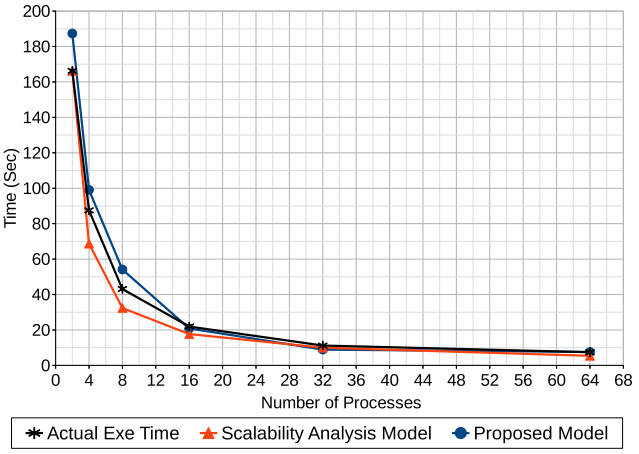

FIGURE 14. Proposed model prediction vs scalability analysis model vs real measurements of EP class-C benchmark running on a group of virtual machines.

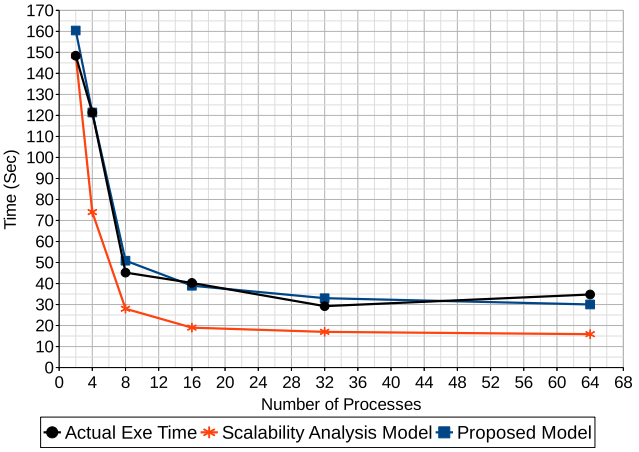

FIGURE 15. Proposed model prediction vs scalability analysis model vs real measurements of FT class-B benchmark running on a group of virtual machines.

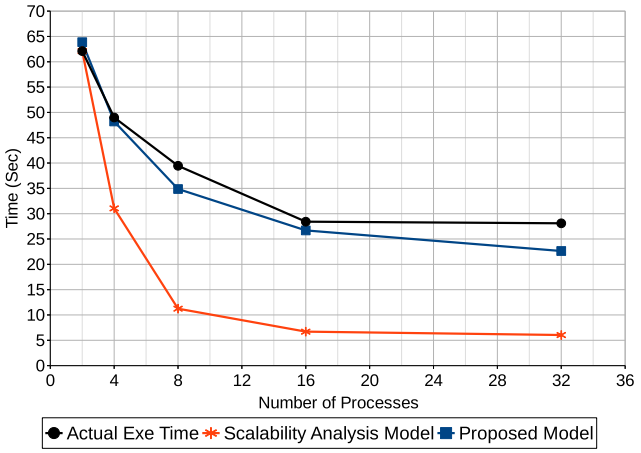

FIGURE 16. Proposed model prediction vs scalability analysis model vs real measurements of IS class-C benchmark running on a group of virtual machines.

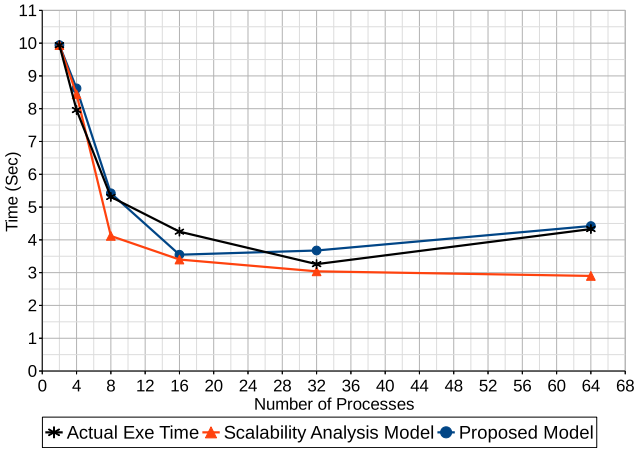

FIGURE 17. Proposed model prediction vs scalability analysis model vs real measurements of MG class-B benchmark running on a group of virtual machines.

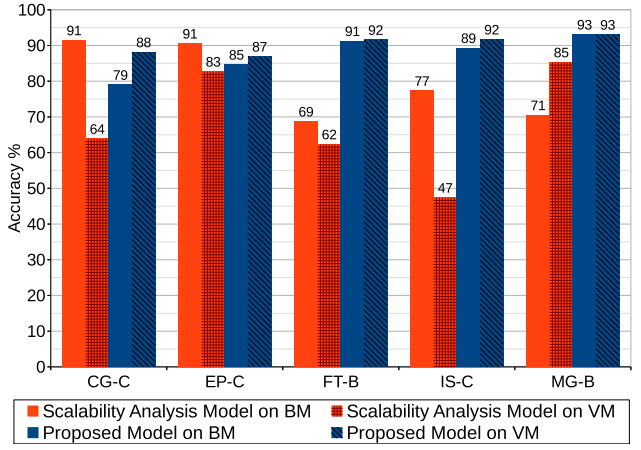

FIGURE 18. Prediction accuracy for the running time of NASA Parallel Benchmarks on both bare-metal servers and virtual machines based on the proposed model and the scalability analysis model.

scalability analysis model to degrade while the prediction based on our proposed model keeps the same accuracy level. This is because the scalability analysis model counts only for the inter-communication between servers and do not consider the intra-communication within the processes located on the same server (processes are assigned to cores on the same

server). In their work, they assume that each server is assigned a single process to run on, and this is not the typical scenario. Usually, the parallel application seeks to make use of all the available computing resources, i.e. the available cores on the server's processor. In contrast, our proposed model counts for both the inter and intra communication between the running processes. Another notice is that the accuracy of the predictions based on the scalability analysis model for the workloads running on the cluster of bare-metal severs is better than the same predictions of the workloads running on a group of virtual machines, for most of the benchmarks. Contrarily, our proposed model copes with the effect of virtualization on the infrastructure hosting the parallel programs, and keeps the same accuracy level.

3) ASSESSING THE VIRTUALIZATION OVERHEAD

In this subsection, the effect of the virtualization overhead on the running time of the parallel benchmarks of SPEC and NPB benchmarks suites is shown through examining the performance difference between running these benchmarks on bare-metal servers and running them on virtual machines. Also, we analyse the causes of this difference. Then, we examine the effect of the virtualization overhead on the CPU and network parameters of our proposed model.

For the SPEC benchmarks, Figure 19 shows the difference between the running time of Lammps (in red colour), Socorro (in green colour), Zeusmp (in blue colour), while running on a cluster of bare-metal servers (in solid lines) and while running on a group of virtual machines (in dashed lines). The X-axis represents the number of running processes; the primary Y-axis represents the running time in seconds for both Lammps and Zeusmp benchmarks and the secondary Y-axis represents the running time in seconds for Socorro benchmark. Further, Figure 20 shows the same comparison but for Lu and Pop2 benchmarks. Lu is represented in green colour and Pop2 in blue colour. The running time for both of them is represented on the primary Y-axis. Also, the running time on the bare-metal servers is represented in solid line while the running time on the virtual machines is represented in dashed line.

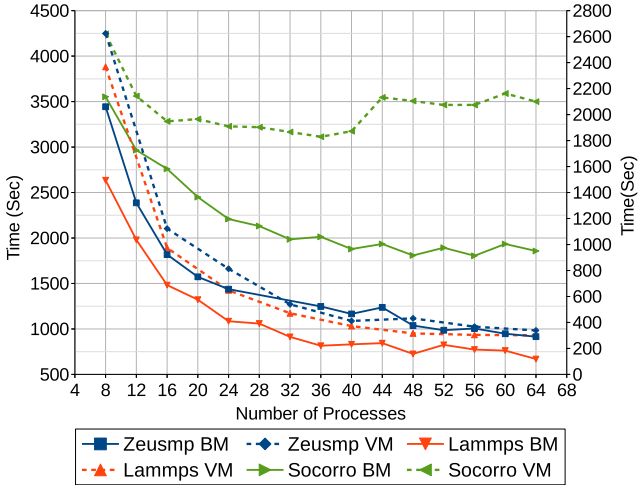

FIGURE 19. Comparison between running Lammps, Socorro and Zeusmp benchmarks on bare-metal machines and running them on virtual machines.

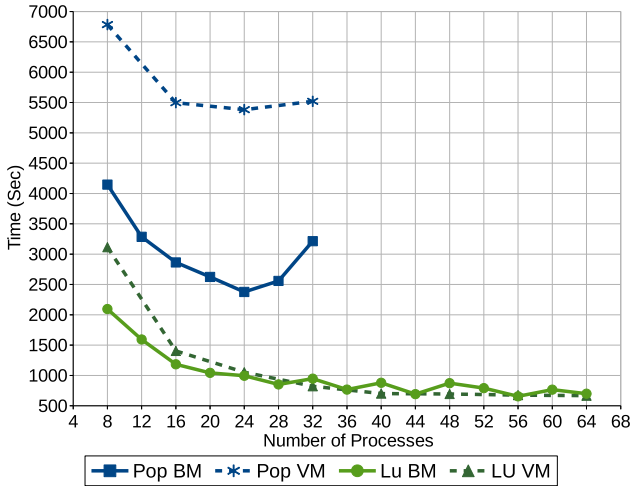

FIGURE 20. Comparison between running Lu and Pop2 benchmarks on bare-metal machines and running them on virtual machines.

Regarding the NASA parallel benchmarks, Figure 21 shows the difference between the running time of the following benchmarks: FT (in red colour), IS (in black colour) and MG (in blue colour), on both the bare-metal servers (in solid lines) and the virtual machines (in dashed lines). Also, Figure 22 shows the same comparison for the benchmarks; EP (in blue colour) and CG (in green colour).

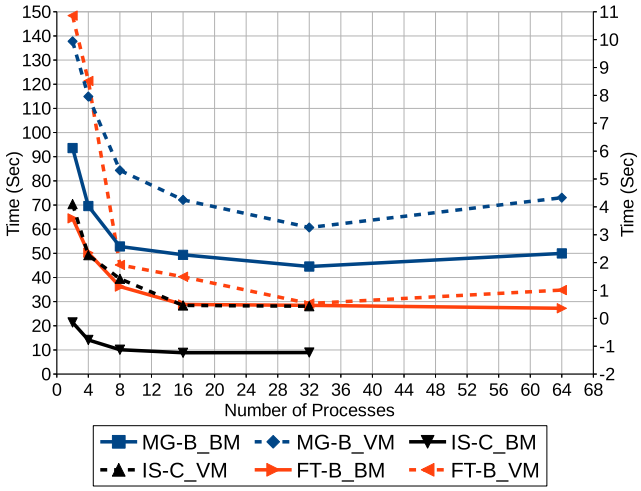

FIGURE 21. Comparison between running FT, IS and MG benchmarks on bare-metal machines and running them on virtual machines.

The outcomes of the comparison between the running times of the benchmarks on both environments, physical (bare-metal) and virtual environments, show a higher running time for the benchmarks while running on the virtual machines than while running on the bare-metal servers. Moreover, the gap between the running times on bare-metal servers and the running times on virtual machines decreases

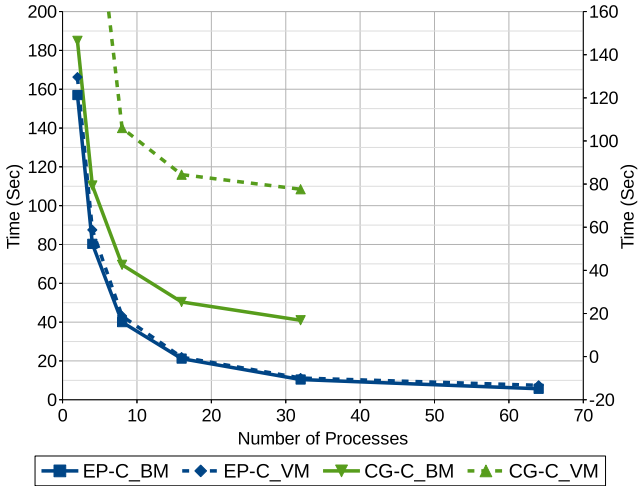

FIGURE 22. Comparison between running CG and EP benchmarks on bare-metal machines and running them on virtual machines.

as the number of the running processes increases. A simple explanation to that is the effect of the virtualization layer on the communication between processes. This is due to the extra time spent by processes queueing for the virtualiztion hypervisor in each communication operation. Besides, as the number of running processes increases, the number of communication operations between them (to transfer data, exchange results or perform synchronization) increases as well. With this increase in the number of processes and consequently the number of communication operations executing the same workload, the communication overhead increases and the network becomes the bottleneck. Thus due to this increase in the number of communication operations and the networking time, both the running times on bare-metal servers and on virtual machines get close to each other as these communication overheads cover the queueing time on the hypervisor for the benchmarks running on virtual machines.

To conclude, the factors that affect the gap between the running time curves of the parallel benchmarks on bare-metal servers and on virtual machines are: the number of running processes executing the benchmarks, the rate of communication between these processes, and whether there is a bottleneck on the network or not. For example, for EP (embarrassingly parallel, NPB benchmark), as it has the least amount of communication data and rate, the curve of executing it on a cluster of bare-metal servers is almost the same as its curve of execution on a group of virtual machines as shown in Figure 22.

Another example to illustrate the effect of the communication rate is the difference between the gaps in case of Socorro and Zeusmp benchmarks. As Socorro demands a higher communication rate between its running processes, the gap between its running time on bare-metal servers and on virtual machines is much wider than the corresponding gap of Zeusmp. Figure 23 and Figure 24 show the communication rate among 32 processes of Zeusmo and Socorro benchmarks, respectively. Each figure shows a grid of coloured cells that visualize the data communicated between the running processes, where each cell's colour represents the amount of data transferred between the processes intersecting at this cell. The colour code at the bottom of the figure shows the relationship between each colour and the amount of data transferred, starting from light grey (least amount of transferred data) to red (most amount of transferred data). It is obvious from Figure 23 that only a small portion of the processes running Zeusmp benchmark communicate with each other. And, even when they communicate, the communication rate is small in most of the cases. On the other hand, Figure 24 shows apparently that the communication rate between the processes running Socorro benchmark is intensively high between each process and the other processes.

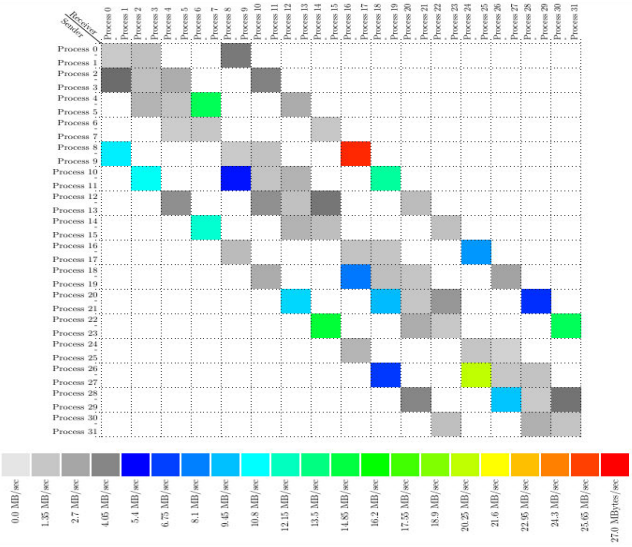

FIGURE 23. Communication rate among 32 processes running Zeusmp benchmark.

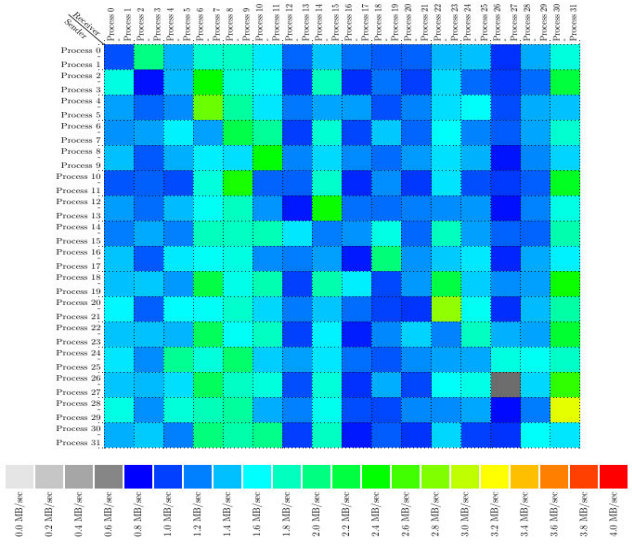

FIGURE 24. Communication rate among 32 processes running Socorro benchmark.

We also study of the effect of the virtualization layer on the model parameters. The main effect is more clear on the CPU_Constant and Net_Constant. These parameters acquired via the non linear problem solver; Gauss-Newton algorithm, represent the inaccuracy in probing the hardware resources' characteristics. We measure the variation that happened on both parameters while running the SPEC-MPI benchmarks on bare-metal servers and on virtual machines. On average, the CPU_Constant is slightly higher (1.25×) while running on virtual machines than while running on the bare-metal servers. On the other hand, the difference is much more larger for the Net_Constant (43.6×). This means that according to the proposed model, the network was the most affected resource by the virtualization on the cloud, which is consistent with the previously mentioned analysis.

VI. CONCLUSION AND FUTURE WORK

In this paper we propose a predictor of the execution time of the tightly coupled HPC applications on the cloud. This predictor is based on an analytical performance model for the MPI-based applications on both cloud bare-metal servers and cloud virtual machines. The model considers the cloud processing and communication resources as a closed queueing network and the HPC-applications running on them as a closed group of tiny jobs competing for these resources/queues. The accuracy of the prediction based on the proposed model is measured on a dedicated cluster of bare-metal servers and also on a group of virtual machines on a private cloud. During the experiments conducted to measure the accuracy of the prediction, 10 different benchmarks are used; 5 from SPEC-MPI benchmark suite and 5 from NPB benchmark suite. The average accuracy of the model's prediction for all benchmarks achieved is 88%. The experiments also show a steady level of accuracy either while running on bare-metal servers or running on virtual machines. In addition to that, the model copes with the effect of the increase in the degree of parallelism of the running workload. Further, we studied the difference between the execution of the parallel programs on the bare-metal servers and on the virtual environment. Through this study, we analyzed the difference gap between the execution on both environments and the effect of virtualization on the proposed model, and reconfirm that communication is severely degraded, in comparison with bare-metal, as identified elsewhere. Future work will consider developing a scheduler for this application domain based on the proposed model's prediction to enhance its performance while running on the cloud. This could be through choosing the best configuration of resources that achieves the best predicted performance with lowest possible cost.

REFERENCES

[1] A. Marathe, R. Harris, D. K. Lowenthal, B. R. de Supinski, B. Rountree, M. Schulz, and X. Yuan, "A comparative study of high-performance computing on the cloud," in *Proc. 22nd Int. Symp. High-Perform. Parallel Distrib. Comput. (HPDC)*, New York, NY, USA, Jun. 2013, pp. 239–250. [Online]. Available: http://doi.acm.org/10.1145/2462902.2462919

[2] S. Kayum and M. Rogowski, "High-performance computing applications transition to the cloud in upstream," in *Proc. 4th EAGE Workshop High Perform. Comput. Upstream*, vol. 2019, no. 1. Bogota, Colombia: European Association of Geoscientists and Engineers, 2019, pp. 1–5. [Online]. Available: https://www.earthdoc.org/content/papers/10.3997/2214-4609.201903299

[3] U. Karneyenka, K. Mohta, and M. Moh, "Location and mobility aware resource management for 5G cloud radio access networks," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2017, pp. 168–175.

[4] M. A. S. Netto, R. L. F. Cunha, and N. Sultanum, "Deciding when and how to move HPC jobs to the cloud," *Computer*, vol. 48, no. 11, pp. 86–89, Nov. 2015. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MC.2015.351

[5] I. P. Egwutuoha, S. Chen, D. Levy, and R. Calvo, "Cost-effective cloud services for HPC in the cloud: The IaaS or the HaaS?" in *Proc. Int. Conf. Parallel Distrib. Process. Techn. Appl. (PDPTA)*, 2013, p. 217.

[6] L. Ramakrishnan, K. R. Jackson, S. Canon, S. Cholia, and J. Shalf, "Defining future platform requirements for e-Science clouds," in *Proc. 1st ACM Symp. Cloud Comput. (SoCC)*, New York, NY, USA, 2010, pp. 101–106. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807145

[7] J. Li, M. Humphrey, C. van Ingen, D. Agarwal, K. Jackson, and Y. Ryu, "EScience in the cloud: A MODIS satellite data reprojection and reduction pipeline in the windows azure platform," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, 2010, pp. 1–10.

[8] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen, "Cloud versus in-house cluster: Evaluating Amazon cluster compute instances for running MPI applications," in *Proc. SC*, Nov. 2011, pp. 1–10.

[9] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon EC2 data center," in *Proc. 29th Conf. Inf. Commun. (INFOCOM)*, Piscataway, NJ, USA, 2010, pp. 1163–1171. [Online]. Available: http://dl.acm.org/citation.cfm?id=1833515.1833691

[10] P. Rad, A. T. Chronopoulos, P. Lama, P. Madduri, and C. Loader, "Benchmarking bare metal cloud servers for HPC applications," in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, Nov. 2015, pp. 153–159.

[11] A. Saad, A. El-Mahdy, and H. El-Shishiny, "Performance modeling of MPI-based applications on cloud multicore servers," in *Proc. Rapid Simulation Perform. Eval., Methods Tools (RAPIDO)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–6, doi: 10.1145/3300189.3300194.

[12] Standard Performance Evaluation Corporation. *SPEC MPI 2007 Benchmark Suite*. Accessed: May 2020. [Online]. Available: https://www.spec.org/mpi2007

[13] Standard Performance Evaluation Corporation. *SPEC MPI 2007 Benchmark Suite Documentation*. Accessed: May 2020. [Online]. Available: https://www.spec.org/auto/mpi2007/Docs

[14] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Comput. Surveys*, vol. 51, no. 1, pp. 1–29, Apr. 2018, doi: 10.1145/3150224.

[15] J. Y. Shi, M. Taifi, A. Pradeep, A. Khreishah, and V. Antony, "Program scalability analysis for HPC cloud: Applying Amdahl's law to NAS benchmarks," in *Proc. SC*, Nov. 2012, pp. 1215–1225.

[16] NASA Advanced Supercomputing Division. *NASA Parallel Benchmark Suite*. Accessed: May 2020. [Online]. Available: https://www.nas.nasa.gov/publications/npb.html

[17] A. Gupta, L. V. Kalé, D. S. Milojicic, P. Faraboschi, R. Kaufmann, V. March, F. Gioachin, C. H. Suen, and B.-S. Lee, "Exploring the performance and mapping of HPC applications to platforms in the cloud," in *Proc. 21st Int. Symp. High-Perform. Parallel Distrib. Comput. (HPDC)*, New York, NY, USA, 2012, pp. 121–122. [Online]. Available: http://doi.acm.org/10.1145/2287076.2287093

[18] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz, "A regression-based approach to scalability prediction," in *Proc. 22nd Annu. Int. Conf. Supercomput. (ICS)*, New York, NY, USA, 2008, pp. 368–377. [Online]. Available: http://doi.acm.org/10.1145/1375527.1375580

[19] P. G. Bridges, M. G. F. Dosanjh, R. Grant, A. Skjellum, S. Farmer, and R. Brightwell, "Preparing for exascale: Modeling MPI for many-core systems using fine-grain queues," in *Proc. 3rd Workshop Exascale MPI (ExaMPI)*, New York, NY, USA, 2015, pp. 5:1–5:8. [Online]. Available: http://doi.acm.org/10.1145/2831129.2831134

[20] R. L. F. Cunha, E. R. Rodrigues, L. P. Tizzei, and M. A. S. Netto, "Job placement advisor based on turnaround predictions for HPC hybrid clouds," *Future Gener. Comput. Syst.*, vol. 67, pp. 35–46, Feb. 2017.

[21] D. A. Menasce, L. W. Dowdy, and V. A. F. Almeida, *Performance by Design: Computer Capacity Planning by Example*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[22] D. A. Menascé, "Virtualization: Concepts, applications, and performance modeling," in *Proc. Int. CMG Conf.*, 2005, pp. 407–414.

[23] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the Xen virtual machine environment," in *Proc. 1st ACM/USENIX Int. Conf. Virtual Execution Environ. (VEE)*, New York, NY, USA, 2005, pp. 13–23. [Online]. Available: http://doi.acm.org/10.1145/1064979.1064984

[24] D. Gupta, R. Gardner, and L. Cherkasova, "XenMon: QoS monitoring and performance profiling tool," Hewlett-Packard Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2005-187, 2005, pp. 1–13.

[25] M. N. Bennani and D. A. Menasce, "Resource allocation for autonomic data centers using analytic performance models," in *Proc. 2nd Int. Conf. Automat. Comput. (ICAC)*, Washington, DC, USA, 2005, pp. 229–240, doi: 10.1109/ICAC.2005.50.

[26] R. Hu, J. Jiang, G. Liu, and L. Wang, "Efficient resources provisioning based on load forecasting in cloud," *Sci. World J.*, vol. 2014, pp. 1–12, Feb. 2014.

[27] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches," *Cluster Comput.*, vol. 11, no. 3, pp. 213–227, Sep. 2008, doi: 10.1007/s10586-008-0060-0.

[28] A. Grama, A. Gupta, G. Karypis, and V. Kumar, "Parallel programming platforms," in *Introduction to Parallel Computing*. London, U.K.: Pearson, 2003, pp. 58–60.

[29] A. Saad and A. El-Mahdy, "Network topology identification for cloud instances," in *Proc. Int. Conf. Cloud Green Comput.*, Sep. 2013, pp. 92–98.

[30] Y. Wang, "Gauss-Newton method," *Wiley Interdiscipl. Rev., Comput. Statist.*, vol. 4, no. 4, pp. 415–420, Jul. 2012, doi: 10.1002/wics.1202.

[31] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queuing networks," *J. ACM*, vol. 27, no. 2, pp. 313–322, Apr. 1980. [Online]. Available: http://doi.acm.org/10.1145/322186.322195

[32] R. Schöne, R. Tschüter, T. Ilsche, and D. Hackenberg, "The vampirtrace plugin counter interface: Introduction and examples," in *Proc. Eur. Conf. Parallel Process.* Berlin, Germany: Springer, 2010, pp. 501–511.

ABDALLAH SAAD received the B.S. degree in computer systems engineering from the Faculty of Engineering at Shoubra, Benha University, in 2007, and the M.S. degree in computer science and engineering from the Egypt–Japan University of Science and Technology, Alexandria, Egypt, in 2014, where he is currently pursuing the Ph.D. degree in computer science and engineering. From 2007 to 2010, he was a Research and Teaching Assistant with the Computer Systems Engineering Branch, Electrical Engineering Department, Faculty of Engineering at Shoubra, Benha University. His research interest includes the performance modeling of computer systems, distributed systems, parallel programming, high performance computing, and algorithm analysis and design. He is a Former Contestant and a Coach in the National and Regional ACM Collegiate Programming Contests from 2008 to 2010. In 2015, he was invited to give a talk in the ISM HPCCON, Tachikawa, Tokyo, Japan.

AHMED EL-MAHDY received the B.Sc. and M.Sc. degrees from Alexandria University and the Ph.D. degree from the School of Computer Science, The University of Manchester, U.K., where he contributed to one of the early multicore processors (JAMAICA). He is on leave from the Computer and Systems Engineering Department, Alexandria University. He has visited the Group of Advanced Processor Technologies contributing to porting the IBM Jikes Dynamic Compiler for JAMAICA. He has also been the Visiting Scientist with IBM Centre for Advanced Studies, Cairo, where he was the First Inventor for eight US issued patents in the area of high-performance computing. He is currently a Full Professor and the Chair of the Computer Science and Engineering Department, Egypt–Japan University of Science and Technology (E-JUST). He is also the Founding Director of Parallel Computing Laboratory, E-JUST with many funded research grants/support from IBM, Amazon, ITIDA, STDF, Academy of Science and Technology in the areas of embedded compilers, high performance GPU acceleration, and high performance computation on the cloud. His research work resulted in around 60 publications including, journals, conference paper, patents, and book chapters, as well as a Startup company. Dr. El-Mahdy is a member of ACM. He is also a TPC Member of ICCD and ARCS Conferences.