Performance Modeling of MPI-based Applications on Cloud Multicore Servers*

Abdallah Saad†
Egypt-Japan University of Science and Technology
New Burj El-Arab, Alexandria
abdallah.saad@ejust.edu.eg

Ahmed El-Mahdy‡
Egypt-Japan University of Science and Technology
New Burj El-Arab, Alexandria, Egypt
ahmed.elmahdy@ejust.edu.eg

Hisham El-Shishiny§
shishiny@eg.ibm.com

ABSTRACT

While cloud computing is widely adopted in many application domains, it is not yet the case for the high performance computing (HPC) domain. HPC traditionally runs on homogeneous, high-cost servers with fast networking providing for predictable performance; while bare-metal cloud offerings is promising, the underlying hardware is heterogeneous, with slower network connection, making it difficult to predict performance and hence tune applications. In this paper we consider performance modelling message passing interface (MPI)-based applications, being a major class of HPC applications. In particular, we present a queueing network performance model to account for computation and communication contentions on the underlying heterogeneous, relatively slow-interconnect architecture of the cloud bare-metal servers. The proposed model uses a non-linear problem solver to enhance the parameters acquired by profiling. We utilise our model to conduct an initial study of the performance of two benchmarks from SPECMPI-2007 suite and two NASA Parallel kernels, executing on a small cluster with varying number of multicore servers ranging from 2 to 8. Comparing the predicted and actual execution times of workloads with different number of processes shows 86% average accuracy for the benchmarks used.

CCS CONCEPTS

• **Computer systems organization → Multicore architectures**; • **Networks → Network performance modeling**; *Network performance analysis*; • **Computing methodologies → Modeling methodologies**; • **Mathematics of computing →** *Nonlinear equations*;

*Produces the permission block, and copyright information
†On leaving from Benha University, abdallah.mohamed@feng.bu.edu.eg.
‡On leaving from Alexandria University.
§Former Manager of Advanced Technology and IBM Center for Advanced Studies in Cairo.

KEYWORDS

Performance Modelling, Message Passing Applications, Cloud Computing, High Performance Computing, Bare Metal Service

ACM Reference Format:
Abdallah Saad, Ahmed El-Mahdy, and Hisham El-Shishiny. 2019. Performance Modeling of MPI-based Applications on Cloud Multicore Servers. In *Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '19), January 21–23, 2019, Valencia, Spain.* ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3300189.3300194

1 INTRODUCTION

Cloud computing has demonstrated its efficiency and benefit for several applications due to its elasticity, easy of usage, scalability, availability and its 'pay as you go' business model. Also, the cloud is able to produce a better turnaround time than the on-premise HPC clusters, considering the waiting time by cluster management systems [8].

However, its utility for HPC applications is still questionable as the HPC users face two main challenges [9]: the relatively higher network latency than the on-premise clusters' network that limits the scalability of the tightly coupled parallel applications, and the difficulty to estimate the cost of running HPC applications on the cloud unpredictable environment. Egwutuoha et al. [4] recommended to use bare-metal servers instead of the cloud instances when running the tightly coupled parallel applications. Currently many cloud providers which include IBM, Amazon, Alibaba, offer dedicated bare-metal cloud machines for HPC applications. However, bare-metal servers still suffer from being heterogeneous, and of relatively lower networking speeds [10] that increases the contention on the network queues. Such characteristics make performance and cost prediction more complicated, making it more difficult to performance/cost optimise parallel applications for such cloud environment.

In this paper we propose an analytical performance model based on queueing networks to capture the underlying resource contention for the underlying heterogeneous architecture. The model targets the important class of MPI applications, which is typical for the HPC domain. The proposed performance model is used to predict the execution time of parallel MPI-based applications running on multicore servers. The model parameters are acquired using a non-linear solver, allowing for general applicability. Then, the prediction procedure can be done in a very short time for different configurations of parameters allowing for a better exploration for the design space. The model helps in answering several questions such as what is the turning point where increasing the scalability of the parallel program is worthless from the performance/cost view point?; what is the best grouping of servers that fits the workload needs given the communication distances between the servers and the processing powers of each?; and how the distribution of processes on servers affects performance, especially that the number of cores per server can vary?

To verify the proposed model, an initial experiment is conducted on three different clusters sizes of cloud physical machines of two, four and eight nodes. Two benchmarks of the SPECMPI-2007 benchmark suite, and two NASA parallel kernels are used as the HPC MPI-based workloads. Results show an average accuracy of 86% when comparing predicted with actual execution times on different configurations. Also, this work is compared with a closely related work [14] using two NASA parallel benchmarks [3] running on the different size clusters. For configurations with homogeneous network topology (single process per node) both models achieve similar accuracy; however, for heterogeneous topology (multiple processes per node) the accuracy of the other model degrades significantly.

The rest of this paper is organised as follows: related work is presented in Section 2. Section 3 describes the proposed performance model and its corresponding parameters acquisition. The methodology used in the modelling process is depicted in Section 4. Section 5 presents and discusses the model validation experiments. Finally, Section 6 concludes the paper and discusses future work.

2 RELATED WORK

Several work is done to evaluate the usage of the cloud as an alternative to ordinary HPC systems. Jackson et al. [7, 11] and Zhai et al. [18] examined the usefulness of cloud computing for e-Science and HPC applications, and Wang et al. [16] studied the impact of virtualisation on network performance of Amazon EC2 data centre. Gupta et al. [6] evaluated the performance of HPC applications on different execution platforms; cluster, grid and a private cloud. Their work shows the performance bottleneck caused by communication on the cloud even with dedicated 10 Gigabit Ethernet network medium. Also, they demonstrate how the performance instability and unpredictability affect the cost of long running applications. Marathe et al. [8] investigate the cloud and HPC clusters using different metrics of comparison; turnaround time and cost. their results show the inefficiency of communication intensive applications on Amazon EC2.

Another trend is to model the cloud system for performance prediction of running HPC applications. Shi et al. [14] introduce an instrumentation assisted complexity analysis methodology for program scalability analysis. Their methodology is based on Amdahl's and Gustafson's laws. They manage to predict the execution time for parallel programs on cloud and HPC cluster. Their results show 71% average accuracy between actual and predicted execution times of five NAS benchmark 'kernels' on HPC cluster, where our proposed model achieves 82.5% average accuracy of predicting execution of two SPECMPI benchmarks. Our model prediction seems better on the overall results. However, the clusters size used in experimentation of the proposed model is smaller than the size of the used cluster in their experiments.

Beyond the cloud virtualised environment, Egwutuoha et al. [4] recommends to run the tightly coupled parallel applications on a bare-metal cluster of servers on the cloud to gain the benefit of the cloud availability—small queue waiting time to start the requested service—and to avoid the cloud relatively slow network compared to HPC clusters.

In order to predict the scalability of parallel programs, Barnes et al. [1] use a modified regression model. In their work, they use several program executions on different number of working processing nodes as a training test to extrapolate the performance of running the parallel program with untrained configuration of processing nodes' count. Their model shows accurate best-fit predictions. However, their work is not flexible to explore and explain the effect of changes in the parameters of interest, e.g. number of cores used per server, allocation of servers in the underlying network, the current network state and the amount of currently available processing power per server. Also, Bridges et al. [2] provide a work in progress to model the MPI main performance characteristics of communication operations using a simple closed queuing network model. They validate their work using simple communication benchmarks running on a tiny cluster of two multicore machines. Their work focuses on modelling the communication operations and does not include modelling neither the hardware characteristics, nor the computation operations of the parallel programs.

3 PERFORMANCE MODEL OVERVIEW

The heterogeneity of the underlying hardware and the underlying relatively slow network would potentially result in contention on both the CPUs and network; the model therefore considers queueing networks for modelling such contentions. Moreover, to generalise the applicability of our model, we decompose/separate 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 larger design space exploration.

This section is organized as follows: Section 3.1 discusses the workload model; Section 3.2 describes the underlying system and its interaction with the workload model; Sections 3.3, 3.4 and 3.5 discuss model parameters acquisition.

3.1 Workload modelling

We consider the workload application as a long sequence of instructions, w, running on n processes. Each process is repeatedly invoked to perform a number of instructions ϕ followed by a send and receive operation. Every single invocation of the process is called a job[1]. The number of jobs invoked during the program running time equals the number of sends per process s. Accordingly, we have the following equation holding:

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

Where s and w are functions of n, depending on the current benchmark.

We consider the MPI program to have a fixed number of cycles;

[1]This is not to be confused with an MPI job; a job here refers to a queueing network job, which is a process cycle

each cycle is a sequence of computations ϕ that ends with a communication operation. When n processes are running, the cycles are distributed uniformly over the processes as jobs.

3.2 Queueing Network and process life cycle

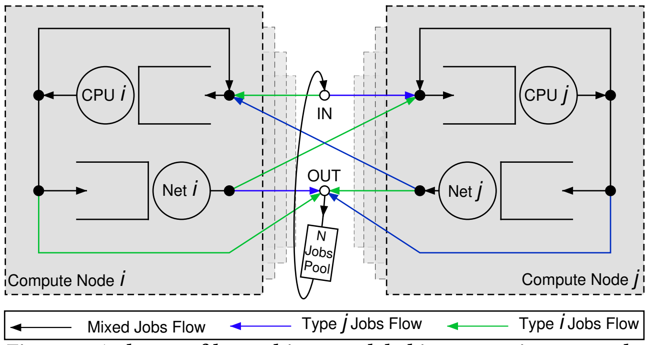


Figure 1: A cluster of k-machines modeled in a queueing network.

Fig. 1 presents our closed-queueing network system for modelling the execution of an MPI program on a cluster of k multicore machines. The figure shows the queueing network for two arbitrary nodes i and j. All other nodes on the cluster have the same structure; all nodes share a centralised job pool. A job represents a single invocation of a process (a cycle). Initially, jobs reside in an MPI jobs pool, which is a collection of jobs ready for execution.

A job enters the system through the 'IN' port, and eventually exits through the 'OUT' port and returns back to the pool, completing an execution cycle. Then, a job is scheduled immediately for another cycle, and this process repeats.

During the job life cycle, the job is distributed over one of the k processing nodes (node i). The job starts using the processing powers of node i through CPU i queue to perform its ϕ computation sequence. After that, the job begins a communication operation with another corresponding job located either locally on node i or remotely on a different processing node (node j). The local communication operations are performed on the CPU queue as a communication overhead quanta, whereas the remote communication operations are done through the processing node's network queues of the two communicating nodes, Net i and Net j.

3.3 Service time calculation

The service time is the average time required to serve a job visiting a queue. Thus, for the case of the CPU queues, the service time is calculated such that it is the time required to process the ϕ instructions of a job. Thus, the service time of the CPU can be computed as: $s_{\text{cpu}} = k\phi$, Where k is a given constant representing the time to execute a single instruction.

Now, assuming that the CPU has c cores, the service time will improve with increasing number of jobs inside the CPU queue, n, until reaching the server's capacity. Thus:

$$s_{\text{cpu}} = \frac{k\phi}{\min(n, c)} \tag{2}$$

Assuming a fixed workload and substituting for $\phi = w/ns(n)$, we get:

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

Now, consider the network part where the service time of the Net queue is the time required to communicate an average size message between two nodes. We define the average message size, m, as a function of n; and the number of total messages is given by $s(n)n$. Therefore, the mean service time of the network can be represented as the communication time of the average message size for a given number of processes. The simplified cost model for communicating messages [5] is used to compute the communication time as follows:

$$s_{\text{net}} = T_s + m(n)T_w \tag{4}$$

Where, T_s is the start-up time to handle a message at the two communicating nodes, and T_w is the per word transfer time. To simplify the model, we assume $T_s = 0$, and $m(n) = m_a/n + m_b$. Thus,

$$s_{\text{net}} = (m_a/n + m_b)T_w \tag{5}$$

m_a and m_b are parameters of modelling the message size variation with respect to n. These parameters can be calculated by fitting the workload profiling data that contains the workload's communication behaviour with respect to n. Where, T_w can be obtained using the methodology described in [13]. In order to calculate the constants values in the CPU service time equation, k and w (for a fixed size workload), we give these constants initial values and consider the outcome service time as an initial guess to a non-linear problem solver, e.g. Gauss-Newton Algorithm [17], to get the best fit for these constants. Thus, $s_{\text{CPU}_{\text{guess}}}$ and $s_{\text{Net}_{\text{guess}}}$ can be defined respectively as;

$$s_{\text{CPU\_guess}} = \frac{\text{CPU\_Constant}}{s(n)n\min(n, c)} \tag{6}$$

$$s_{\text{Net\_guess}} = s_{\text{net}} \times \text{Net\_Constant} \tag{7}$$

Where, CPU_Constant = kw and Net_Constant represents the imperfection in probing the network state.

3.4 Visit ratio calculations

The visit ratio of a queue is the average number of times a job will visit the queue during its life cycle. In other words, it is the number of quanta of processing/communication required by a job assuming that each visit takes a certain time quantum.

In order to calculate a visit ratio, we need to consider the job's average CPU quanta spent during both its computation and the communication overhead processing. We start by calculating the communication overhead by profiling the parallel application on a physical machine under no contention condition; where there is no network communication time and all jobs are communicating locally.

Now, let us consider the time required to finish a job life cycle (Cycle_time); it consists of the time to execute the job's instructions (T_{comp}), the network time (T_{nw}) the job needs to communicate remotely, and the communication overhead time (T_{oh}) that mimics the communication waiting time the job spends waiting for its adjacent job to prepare the response of the communicated message. Thus, we can define: Cycle_time = $T_{\text{comp}} + T_{\text{nw}} + T_{\text{oh}}$. And thus, the cycle time for each process (Process_cycle_time) is defined as: Process_cycle_time = Cycle_time/n.

By profiling the parallel application using a number of running processes that is less than or equal to the available processing units (no CPU contention) we get the execution time of the running application (T_{exe}), and the wait time for all the MPI communication operations (MPI_Wait) performed by all processes. For that parallel execution, we assume all processes to start and end execution together. According to that, the application execution time represents the running time of a single process. Thus, the per process cycle time is defined as Process_cycle_time = T_{exe}/s.

Since the profiled run was on a single machine, the T_{nw} is neglected. And T_{oh} from profiling can be represented as; T_{oh} = MPI_Wait$/s$. Thus, in this case T_{comp} is defined as $T_{\text{comp}} = T_{\text{exe}} - T_{\text{oh}}$.

By knowing T_{exe}, T_{comp} and T_{oh}, we can compute the sub-visit ratio of computation time spent during the job cycle time to the total cycle execution time (V_{comp}). And similarly, the sub-visit ratio of communication overhead time of a job cycle to the cycle's total execution time (V_{comm}). Where, $V_{\text{comp}} = T_{\text{comp}}/T_{\text{exe}}$ and $V_{\text{comm}} = T_{\text{oh}}/T_{\text{exe}}$. After calculating the sub-visit ratios needed, V_{comp} and V_{comm}, we can deduce the job visit ratio to each processing node's CPU queue. Let's assume a compute node i; the CPU queue of node i is visited by all the jobs located originally on node i to perform their computation. Also, the jobs located on node i would re-enter the CPU queue again for doing the communication overhead if these jobs are going to communicate locally. Finally, the CPU queue of node i is possibly visited by jobs located originally on different compute nodes to make remote communication with adjacent jobs on node i. Thus, the visit ratio of the CPU queue of node i is defined as:

$$V_{\text{CPU}_i} = \frac{n_i}{n} V_{\text{comp}} + \frac{n_i}{n} \frac{n_i - 1}{n} V_{\text{comm}} + \frac{n - n_i}{n} \frac{n_i}{n} V_{\text{comm}} \tag{8}$$

Where $\frac{n_i}{n}$ is the ratio of node i jobs to the total number of running jobs, $\frac{n_i}{n}\frac{n_i-1}{n}$ is the ratio of jobs of node i that possibly could communicate locally with jobs of node i as well, and $\frac{n-n_i}{n}\frac{n_i}{n}$ is the ratio of jobs originally located outside node i and performing remote communication operations with jobs of node i.

Similarly, the visit ratio for the network device queue (V_{net_i}) can be deduced. The jobs of node i that are communicating remotely visit node i's network queue (net$_i$). Also, jobs located outside node i but communicating remotely with jobs of node i are going to visit net$_i$ in their return journey. Thus, we can define V_{net_i} to be:

$$V_{\text{net}_i} = \frac{n_i}{n} \frac{n - n_i}{n} + \frac{n - n_i}{n} \frac{n_i}{n} \tag{9}$$

Where $\frac{n_i}{n}\frac{n-n_i}{n}$ represents the ratio of jobs of node i doing remote communication with jobs of other nodes, and $\frac{n-n_i}{n}\frac{n_i}{n}$ represents the ratio of jobs of nodes other than i that are communicating remotely with jobs of node i.

3.5 Job response time calculations

Now let us call the system response time to be R; it represents the execution time for a ϕ sequence. Since every process has $s(n)$ sequence of ϕ to execute, each process response time is $s(n)\phi$. And since we assume that all n parallel sequences enter the system at the same time and finish execution at the same time, the system total execution time, T, for the w instructions is:

$$T = Rs(n) \tag{10}$$

We profile the workload for different number of uniformly distributed running processes, to get the execution time for each and to profile the communication operations to help modelling them for any given number of running processes. For example, modelling the average size of messages for different n, $m(n)$ as An^{-B} and the average number of sends per process $s(n)$ as $C\ln(n) + D$. Where $A, B, C$and D are constants calculated using the gathered profiling data. Using these profiling data, the CPU_Constant and Net_Constant are calculated using GNA nonlinear problem solver.

Using the service time and visit ratio, the average response time of jobs (R) to complete one life cycle is calculated. In our model, we use a numerical solution to get R while the workload is parallelised over n processes. Mean value analysis (MVA) algorithm [12] is used to calculate R, knowing the values of service time and visit ratio for all queues in our queueing network model.

4 PROPOSED SYSTEM DESIGN

The proposed system shown in Fig. 2 is used to model both the multicore system available resources and the MPI-based application required resources as well. The system starts with two parallel steps:

- the application profiler runs the MPI-based application (more than two times) with test input data to profile the program communication behaviour with respect to the variation in number of working processes,
- the hardware prober gathers information about the available resources and its current state.

If these information are known a priori, then those parallel steps are skipped. Then, the output from the application profiler and the hardware prober move to the initial model for calculating the guess service time and visit ratio for each CPU queue and NW queue for each compute node in the system. At that instant, the system has two paths to go based on the availability of the final CPU and NW constants:

- if available; the mathematical model uses them to calculate the service time and visit ratio for each queue in the queuing network. Afterwards, the mean value analysis (MVA) technique is used to numerically estimate the response time of the system for any given number of processes, then, print it to the user and the system exit.
- if not available; the non-linear problem solver, Gauss-Newton Algorithm (GNA), is used to find the values of the constants that minimise the error between the expected response time and the execution times previously measured during profiling. To do that, the GNA uses the proposed model equations with the initial service times and sub-visit ratios to get the temporal service times and visit ratios for each queue in the queuing network. Then, it passes them to the MVA technique to calculate the predicted response time and send it as a feedback to the GNA. If the error is less than a predefined threshold ϵ, these constants' values (for this workload on this multicore system at this state) are stored and the proposed system finishes. Otherwise, the system repeats that path until the system finds the constants that minimise the error to be less than ϵ.

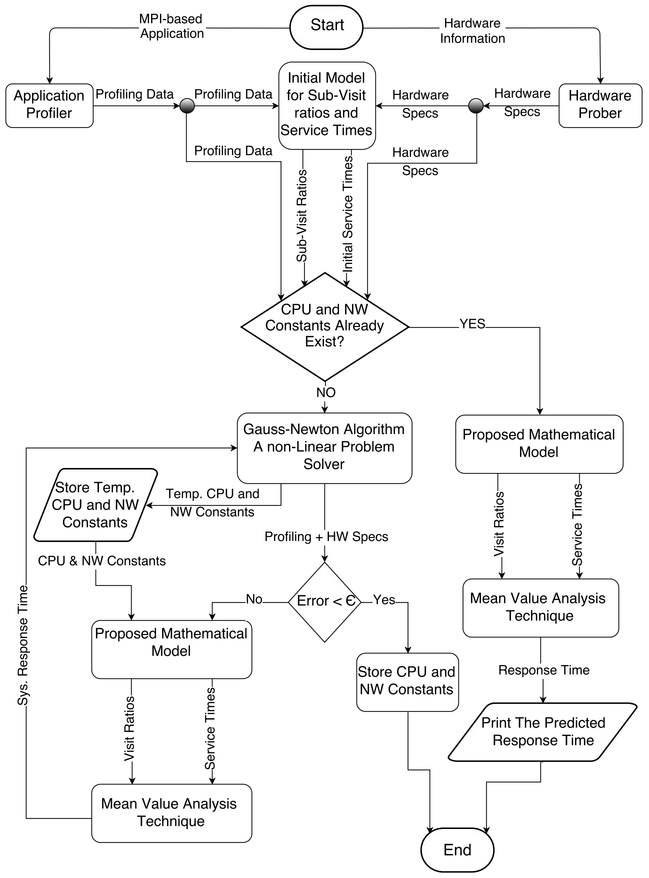

Start
MPI-based Application
Hardware Information
Application Profiler
Profiling Data
Profiling Data
Initial Model for Sub-Visit ratios and Service Times
Hardware Specs
Hardware Specs
Hardware Prober
Profiling Data
Hardware Specs
Sub-Visit Ratios
Initial Service Times
CPU and NW Constants Already Exist?
YES
NO
Gauss-Newton Algorithm A non-Linear Problem Solver
Store Temp. CPU and NW Constants
Temp. CPU and NW Constants
Profiling + HW Specs
CPU & NW Constants
Sys. Response Time
No
Error < ϵ
Yes
Proposed Mathematical Model
Store CPU and NW Constants
Visit Ratios
Service Times
Mean Value Analysis Technique
Proposed Mathematical Model
Visit Ratios
Service Times
Mean Value Analysis Technique
Response Time
Print The Predicted Response Time
End

Figure 2: Proposed system design

5 EXPERIMENTS, RESULTS AND ANALYSIS

In this work, the '126.lammps' and '130.socorro' benchmarks of SPECMPI-2007 benchmark suite [15] are used as the MPI-based workload, as well as two kernels from NASA Parallel benchmark. Clusters of 2, 4 and 8 multicore compute nodes are used as the underlying system with 1 GBit/s Ethernet network. All of the compute nodes are running CentOS-7.4 release. The clusters specifications are shown in Table 1. The system runs MPICH 3.1 and gcc 4.8.5 tool chain.

Table 1: Clusters Specifications

Num of nodes	Nodes variability	Memory Size (GB)	Processors model	Num of cores
2	homogeneous	24	E5620	16
4	homogeneous	24	E5620	32
8	4x	24	E5620	88
	2x	48	E5645	
	1x	24	E5-2450	
	1x	32	E5-2450	

After gathering and calculating all the data needed to calculate the service time and the visit ratio to the queueing network model queues, both service times and visit ratios are used as an input to the MVA algorithm to calculate the response time R numerically. Finally, from Equ. 10 the predicted execution time is calculated for the number of running processes n.

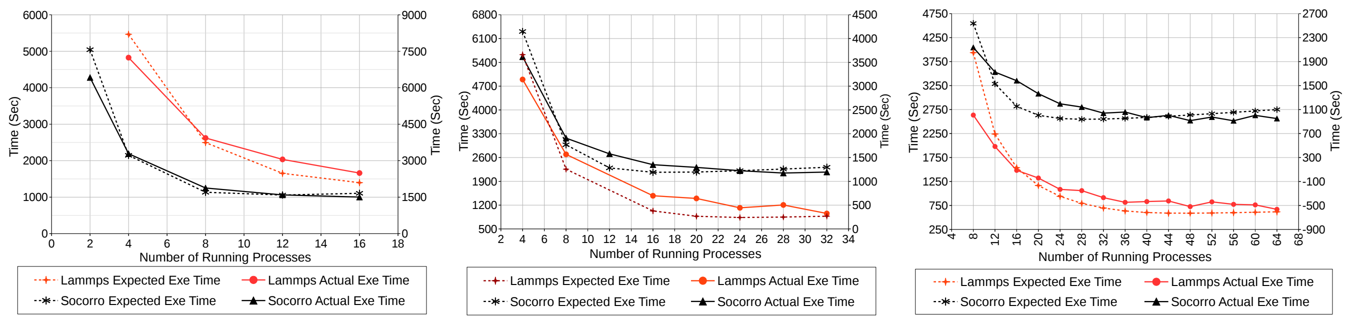
Fig. 3a, Fig. 3b and Fig. 3c show a comparison between our model prediction and the actual measurements of running 'lammps' and 'socorro' benchmarks on the cluster of two, four and eight nodes, respectively. The benchmarks are run with medium reference (mref) data set as it is more reliable workload than the test data size that has a very short execution time which is more susceptible to noise. For all sub-figures in Fig. 3, The secondary Y-axis represents the actual and predicted execution time of the 'Socorro' benchmark, where the primary Y-axis represents the execution time for 'Lammps' benchmark. The results illustrate the model prediction for different parallel workloads.

In order to measure the model accuracy, two statistics are used; coefficient of variation (CV) and mean absolute percentage error (MAPE). The Accuracy is calculated by subtracting MAPE from 100. On average for the three different clusters, the predicted execution time for 'Socorro' was accurate by 89.8% with average CV = 0.16 and, 'lammps' achieves 82.3% accuracy with average CV = 0.24. Also, Fig. 3 shows the turning points at which scaling up the parallelism of the program do not have a significant impact on the performance. For a cluster of two nodes, lammps execution time starts to saturate using 16 cores while socorro saturates using 8 cores only. For four nodes the saturation happens at 24 and 16 cores for lammps and socorro respectively. Where using a cluster of eight noeds, the program scalability saturates at 36 and 40 cores for lammps and socorro respectively.

There is another close work done to model and predict the cloud system for running HPC applications without using the queueing networks; Shi et al. [14] introduce an instrumentation assisted complexity analysis methodology for program scalability analysis. Their methodology is based on Amdahl's and Gustafson's laws. Their work is re-implemented on the same experimental setup described in this paper for fair comparison. Two kernels of NASA parallel benchmark suite are used in this comparison; CG and MG with class-C problem size for both. The prediction of their model and the proposed model against the actual measured execution time are shown in Fig. 4. Their model does not count for the heterogeneity in the communication operations in parallel applications where there are inter- and intra- communication operations among the working processes. Their model counts only for intercommunication operations among the processing units not the intra-communication operations occurred inside the same processing unit. So, the accuracy for both models are almost the same for the experiments where every server in the working cluster contains only one process; up to 8 servers. The average accuracy for both kernels was 89% for the proposed model where the scalability analysis model achieves 87.3% accuracy. However, accounting for the intra-communication operations with no resources contention (using the available number of cores only), the proposed model results in 87.4% average accuracy for both kernels preserving the same level of accuracy, where the scalability analysis model's accuracy degrades to only 61%.

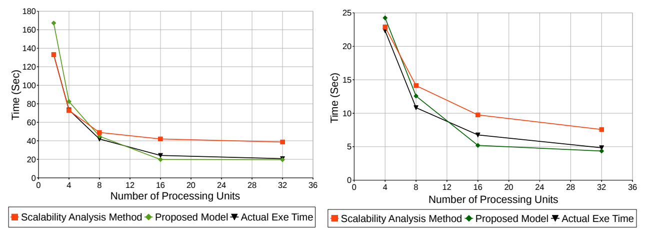
6 CONCLUSION AND FUTURE WORK

In this paper we propose an analytical performance model for predicting the performance of HPC MPI applications on the cloud bare-metal multicore servers. The model considers the contention



(a) Cluster of two machines. **(b) Cluster of four machines.** **(c) Cluster of eight machines.**

Figure 3: Running time prediction results of SPEC-MPI benchmarks; Lammps and Socorro on three different size clusters.



(a) CG benchmark. **(b) MG benchmark.**

Figure 4: Proposed model running time prediction results of NAS Parallel Benchmarks; CG and MG compared to the scalabilty analysis method prediction.

on both computation and communication resources through modelling them as a queueing network. In addition, the model accounts for the heterogeneity with the cloud bare-metal servers, where the model considers the processors speed and available number of cores (which varies among servers) as parameters as well as considering both the intra and inter communication operations between processes inside the same server or among different servers. For our experiments, we consider a cluster of multiple bare-metal cloud servers as the underlying system and two benchmarks from the SPECMPI suite as well as two kernels from NASA Parallel benchmarks suit as the workload. The results show a prediction, with 86% average accuracy, to the execution times of the two running benchmarks for different configurations of compute nodes; two, four and eight different machines.

Thus the model can potentially be used to assess the cost of resource usage on the performance of the cloud physical machines with different virtual to physical configurations, and to aid in developing better schedulers, which is an important area for future work. In particular, the queueing service times can account for the hypervisor overhead. Also, the number of working processes on each processing node would reflect the number of working virtual machines on each physical host, allowing for modelling resource sharing contention. Moreover, we need to extend the model to account for interference with other workloads running simultaneously on the same multicore machines.

REFERENCES

[1] Bradley J. Barnes, Barry Rountree, David K. Lowenthal, Jaxk Reeves, Bronis de Supinski, and Martin Schulz. 2008. A Regression-based Approach to Scalability Prediction. In *Proceedings of the 22Nd Annual International Conference on Supercomputing (ICS '08)*. ACM, New York, NY, USA, 368–377. https://doi.org/10.1145/1375527.1375580

[2] Patrick G. Bridges, Matthew G. F. Dosanjh, Ryan Grant, Anthony Skjellum, Shane Farmer, and Ron Brightwell. 2015. Preparing for Exascale: Modeling MPI for Many-core Systems Using Fine-grain Queues. In *Proceedings of the 3rd Workshop on Exascale MPI (ExaMPI '15)*. ACM, New York, NY, USA, Article 5, 8 pages. https://doi.org/10.1145/2831129.2831134

[3] NASA Advanced Supercomputing Division. [n. d.]. NASA Parallel Benchmarks Suite. https://www.nas.nasa.gov/publications/npb.html.

[4] Ifeanyi P Egwutuoha, Shiping Chen, David Levy, and Rafael Calvo. 2013. Cost-effective Cloud Services for HPC in the Cloud: The IaaS or The HaaS?. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. 217.

[5] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. 2003. *Introduction to Parallel Computing*. Pearson Education Limited, Chapter Parallel Programming Platforms, 58–60.

[6] Abhishek Gupta, Laxmikant V. Kalé, Dejan S. Milojicic, Paolo Faraboschi, Richard Kaufmann, Verdi March, Filippo Gioachin, Chun Hui Suen, and Bu-Sung Lee. 2012. Exploring the Performance and Mapping of HPC Applications to Platforms in the Cloud. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC '12)*. ACM, New York, NY, USA, 121–122. https://doi.org/10.1145/2287076.2287093

[7] J. Li, M. Humphrey, C. van Ingen, D. Agarwal, K. Jackson, and Y. Ryu. 2010. eScience in the cloud: A MODIS satellite data reprojection and reduction pipeline in the Windows Azure platform. In *IPDPS 2010*. 1–10. https://doi.org/10.1109/IPDPS.2010.5470418

[8] Aniruddha Marathe, Rachel Harris, David K. Lowenthal, Bronis R. de Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. [n. d.]. A Comparative Study of High-performance Computing on the Cloud. In *HPDC '13*.

[9] M. A. Netto, R. L. Cunha, and N. Sultanum. 2015. Deciding When and How to Move HPC Jobs to the Cloud. *Computer* 48, 11 (Nov. 2015), 86–89. https://doi.org/10.1109/MC.2015.351

[10] Paul Rad, AT Chronopoulos, P Lama, Pranitha Madduri, and Cameron Loader. 2015. Benchmarking bare metal cloud servers for HPC applications. In *Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on.* IEEE, 153–159.

[11] Lavanya Ramakrishnan, Keith R. Jackson, Shane Canon, Shreyas Cholia, and John Shalf. 2010. Defining Future Platform Requirements for e-Science Clouds. In *SoCC (SoCC '10)*. ACM, NY, USA, 101–106. https://doi.org/10.1145/1807128.1807145

[12] M. Reiser and S. S. Lavenberg. 1980. Mean-Value Analysis of Closed Multichain Queuing Networks. *J. ACM* 27, 2 (April 1980), 313–322. https://doi.org/10.1145/322186.322195

[13] A. Saad and A. El-Mahdy. 2013. Network Topology Identification for Cloud Instances. In *2013 Int. Conf. on Cloud and Green Computing*. 92–98. https://doi.org/10.1109/CGC.2013.22

[14] J. Y. Shi, M. Taifi, A. Pradeep, A. Khreishah, and V. Antony. 2012. Program Scalability Analysis for HPC Cloud: Applying Amdahl's Law to NAS Benchmarks. In *SC12*. 1215–1225. https://doi.org/10.1109/SC.Companion.2012.147

[15] The Standard Performance Evaluation Corporation (SPEC). [n. d.]. SPEC MPI 2007 Benchmark Suite Documentation. https://www.spec.org/auto/mpi2007/Docs.

[16] Guohui Wang and T. S. Eugene Ng. [n. d.]. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *INFOCOM'10*.

[17] Yong Wang. 2012. GaussâĂŞNewton method. *Wiley Interdisciplinary Reviews: Computational Statistics* 4, 4 (7 2012), 415–420. https://doi.org/10.1002/wics.1202

[18] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen. 2011. Cloud versus in-house cluster: Evaluating Amazon cluster compute instances for running MPI applications. In *SC11*. 1–10.