Network Topology Identification for Cloud Instances

Abdallah Saad
Computer Science and Engineering Department
Egypt-Japan University of Science and Technology (E-JUST)
Alexandria, Egypt
abdallah.saad@ejust.edu.eg

Ahmed El-Mahdy
Computer Science and Engineering Department
Egypt-Japan University of Science and Technology (E-JUST),
and on leave from Alexandria University
Alexandria, Egypt
ahmed.elmahdy@ejust.edu.eg

***Abstract*— High performance computing (HPC) on the cloud is an emerging approach that can potentially provide a significantly cheaper alternative to supercomputers. However, clouds are largely oriented towards multiprogramming workloads with no significant intercommunications. The placement of tightly coupled HPC virtual machines is thus not guaranteed to be physically affine, resulting in unpredictable communication times. This paper proposes a new cloud analytical model that describes the physical placement of virtual machines in the communication hierarchy. The model is constructed through a set of automated experiments that measure virtual machines point-to-point communication speed parameters; the parameters are then clustered, and the topology of the cloud network seen by the virtual machines is identified. As a case study, the paper applies the model to the Amazon Cloud; the obtained hierarchical model is used to select a fast communicating subset of instances and discarding the other instances. For a message-passing all-to-all communication operation such selection resulted in 4.1 to 5.5 speedup enhancement in performance when randomly executing on a similarly sized subset.**

I. INTRODUCTION

Cloud computing providers exploit economy of scale and consolidation of resources [1] to significantly cut down administration and computation costs resulting on cheap computation. Cloud computing has a potential to provide for a cheaper alternative to high performance computation systems. However, a major hurdle is the intercommunication variability among communicating nodes or instances [2].

A cloud system interconnect is typically organized as a collection of hierarchically interconnected computing nodes [3]; a typical system communication hierarchy starts by multicore processors sharing the same computing node; then computing nodes sharing the same switch forming a rack; and lastly, a group of racks sharing a core switch. A group of communicating virtual machine (VMs) is not guaranteed to reside on the computing node due to allocation fragmentation; thereby the intercommunication varies with every group creation.

Cloud computing relies on virtualization to abstract the underlying physical computing resources, allowing for lower cost of management and better sharing of the resources. However, virtualization hides the mapping between the instance and the underlying physical machines; thus the nodes are not aware of their physical placement.

In this paper, we propose a new method to identify the physical placement of VMs on the cloud for an arbitrary group of VMs. The method computes point-to-point communication speeds among the VMs. The obtained values are then clustered so that VMs in the same cluster share similar communication speeds among all cluster members. This potentially allows for achieving more efficient and predictable execution of intensively communicating HPC application on the cloud.

The main contributions of this paper are:

- Modeling and obtaining the parameters of the point-to-point communication latencies among VMs.
- Identifying a suitable clustering method that identifies the underlying physical interconnection layout on the cloud.
- A case study of all-to-all broadcast group communication operation optimization on the Amazon Web Services (AWS) cloud.

Initial results on the application of our method on the all-to-all group communication operation achieve 4.1 to 5.5 speedup enhancement in performance, compared to executing the same benchmarks on randomly chosen groups of VMs. In particular, we choose a smaller group of VMs that have fast inter-communication speeds and discards the other VMS. We thereby select the best VMs for the given problem.

The rest of this paper is organized as follows: Section II presents a detailed description of the proposed method. Section III presents the experiments, results and analysis of our method. Section IV gives a quick overview of the related work. And, finally Section V concludes the paper.

II. VM TOPOLOGY IDENTIFICATION METHOD

The main objective of the method is to generate a communication tree that approximates the underlying communication structure among the VMs. The VMs are the leaves of the tree, and the internal nodes represent a network switch.

We model the point-to-point communication between nodes by the following simplified formula [4]:

$$T_{comm} = T_s + m\,T_w \qquad (1)$$

Where T_{comm} is total time to transfer a message of m units between two particular nodes. Ts is the setup time to start the communication operation between the participating nodes. This time is required once per a single message transfer operation. Tw is the time to transfer a unit message between the two nodes.

We chose such simplified model as it abstracts many of the details that would be difficult to obtain their parameters from the cloud (such as number of hops, etc).

We measure the *Tcomm* via a sequence of blocking ping-pong message transfer operations similar to the osu_latency benchmark [15]. We choose to measure latency as it is a more accurate measure of physical node distance than bandwidth.

One particular issue in measuring *Tcomm* is that the CPU sharing in the cloud can result in any of the communicating parties to be preempted. This would increase the perceived communication time. To help negate that effect, we repeat the point-to-point transfer operation many times and select the minimum communication time. The rational here is that the minimum is likely to capture the situation where both nodes are no preempted.

The *Tw* parameter characterizes the propagation delay among nodes; we therefore choose it to characterize communication switches; in other words, nodes with similar *Tw* are clustered together.

In the following subsections we describe in detail our method steps depicted in Fig. 1.

A. VMs Generator

This is the stage where the instances are automatically hired from the service provider with the given specifications and user credentials as inputs. The specifications of the instances are the number of required instances, instances type, security group of the instances and the image that those instances are created from. The output of this stage is the required instances and their private IPs.

B. Hardware Probes

At this stage all the measurements are done. Each instance probes its neighbors using point-to-point communications with different message sizes. The probe operations of all the instances are made in parallel. The private IPs of the instances are used to automatically create the MPI run command which is used to execute all the measurements. The measurements are repeated n times (where *n* is a user input).

Those measurements are communication time matrix, measuring the communication time between every pair of instances for different communicated message sizes; the metrics measured are average communication time, min/max, standard deviation, and coefficient of variation of the communication times.

The minimum communication time matrix is three-dimensional. The minimum communication time is measured for different message sizes from a range that starts from 1 byte to a given maximum size with a given step size. For example: the range starts from 1 byte to maximum of 500 KB with a step size of 10 KB; this gives a range of 50+1 different sizes {1, 10000, 20000, … , 500000}.

C. Curve Fitting

The given minimum communication time matrix from stage #2 are plotted versus the message sizes for each pair of nodes. This stage fits the output curves using the following simplified formula (1) presented earlier.

The fitting results in extracting two features *Ts* - emphasizing the latency- and T_w -indicating the minimum communication time for each pair of nodes. The output of this stage is two 2D matrices one for the setup time, *Ts*, between each pair of instances in microseconds, and the second matrix is T_w between each pair of instances also in microseconds. Both of the matrices are square symmetric matrices of size (*np* x *np*) with zero trace, where *np* is the number of instances hired.

D. Topology Identifier

Here the instances are grouped and clustered based on the differences of T_w among them. The applied clustering technique is the average linkage clustering [5]. This clustering technique initially considers each instance as a single cluster and then based on a given threshold, the clusters can be merged if and only if the average of differences between each instance in each cluster and the other cluster instances is less than or equal to the given threshold.

The output of this stage is a hierarchical cluster in the shape of a dendrogram. This diagram represents the communication costs between the hired instances in microseconds. Moreover, the diagram identifies the network topology between the physical machines hosting the hired instances. This gives a clue to the closest instances in groups with minimal communication costs. This information benefits HPC applications to minimize their execution time and performance.

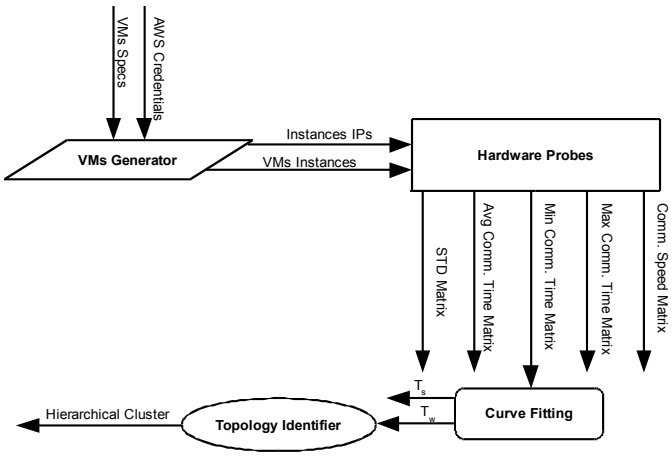

Fig. 1. The proposed methodology for identifying VMs network topology

III. EXPERIMENTS AND RESULTS

Following the methodology described above, the experiment begins with automatically creating *np* instances, with a given specification on Amazon EC2. Then using the resulting instances' IPs to create an MPI run command to the program that probes the instances that are hosted on physical machines with only one process per instance. The program results in number of matrices mentioned previously. The curve-fitting program starts fitting the communication time matrix with equation (1). The curve fitting phase produces Ts and Tw matrices.

As a proof of concept before testing the proposed method on the cloud, an experiment was applied in a local machine to verify the used model. Then the difference between the real measurements (T_{comm} vs. M) and the line resulted from the curve fitting stage, is measured. The local Machine specificationsare listed in Table 1, below.

TABLE I. LOCAL MACHINE SPECS

Item	Description
Processor	Intel® Core™ i7-2670QM CPU @ 2.20GHz × 8
Memory	8 GB
Caches	6 MB
Architecture	64-bit
O/S	Ubuntu ™ 12.04 precise LTS

Fig 2 shows the results.

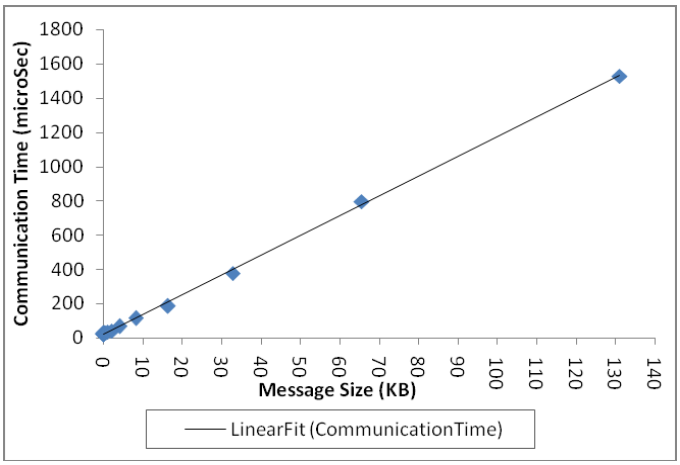

Fig. 2. Communication time between two cores on local machine versus different workloads

The linear-fitting results in the following equation:

$$T_{comm} = 0.011\ M + 20.19 \qquad (2)$$

With coefficient of determination (R^2) = 0.999.

Fig.2 shows that the fitted line is clearly close to the real measurements. This result encourages us to proceed in the experiments.

The next step is to verify the model on the cloud and analyses the measurements. So, on AWS we begin with only two instances with specifications listed in Table 2.

The purpose of this experiment is to model the communication time between two virtualized instances on the cloud to check the validity of the proposed model on a virtual environment. The communication process works as follows: Communicate different message sizes that ranges from zero byte to max size of 128 KB per message by doubling the message size every step. The measurements are taken 10 times for each message size; each time the sender sends the message 100 times, and the average communication time of those communication operations is calculated. Fig. 3 shows the output of this experiment with regular fitting.

TABLE II. VIRTUAL MACHINE SPECS

Item	Description
Type	M1. large
Processor	2 cores each with 2 EC2 compute units
Memory	7.5 GB
I/O Performance	500 Mbps
Architecture	64-bit
O/S	Ubuntu ™ 12.04 precise LTS

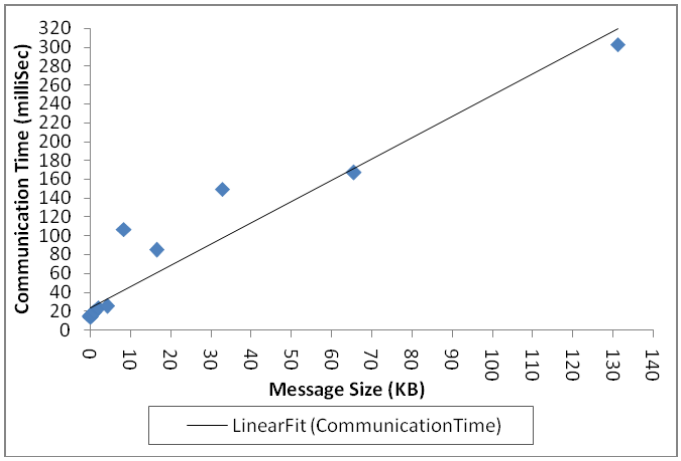

Fig. 3. Communication time between two virtual instances on the cloud

Equation 3 is the resulted fitting-line equationwith R^2 = 0.918.

$$T_{comm} = 2.261\ M + 23590 \qquad (3)$$

These results on the cloud areless accurate than the measurements on the local machine due to the virtualization environment and the variation of the workloads on the

physical machines hosting the VMs on the cloud. However these results are acceptable.

After proofing the concept that our model can represent the communication between virtualized instances on the cloud, we start some experiments that pass through the whole proposed method's stages.

The experiments were done over 16 virtualized instances with specifications listed in Table 2. The "VMs generator" module automatically hired the required instances, and then the "Hardware prober" probes the physical machines hosting the virtualized instances and outputs the communication matrix to the "Curve fitting" stage.This stage automatically fits the communication time between each pair of instances and passes the T_w matrix to the "Topology identifier" stage. This stage applies the average linkage clustering on the given matrix and results in the hierarchal cluster in Fig. 4 based on T_w and in Fig. 5 based on T_s. We consider that the cluster based on T_w represents the real topology of the hired VMs because T_s is the absolute term for the communication equationthat is affected by the state of the communicating machines more than the communication channel or distance.

As a proof of accuracy of our proposed topology, we use a message passing all-to-all broadcasting communication operation that is executed on different groups of VMs. Each group consists of 8 VMs. These groups are as follows: a group of the closest machines according to the proposed topology based on T_w, a group of the closest machines according to the proposed topology based on T_s, a group of the farthest VMs according to the proposed topology based on T_w, a group of random nodes with predefined numbers (from 0 to 7) and another group of random VMs (from 8 to 15). Then we compare the execution times of the all-to-all benchmark with different message sizes {0, 100, 512, 1K, 10K, 50K}Bytes. This comparison is presented in Table 3 and Fig. 6.

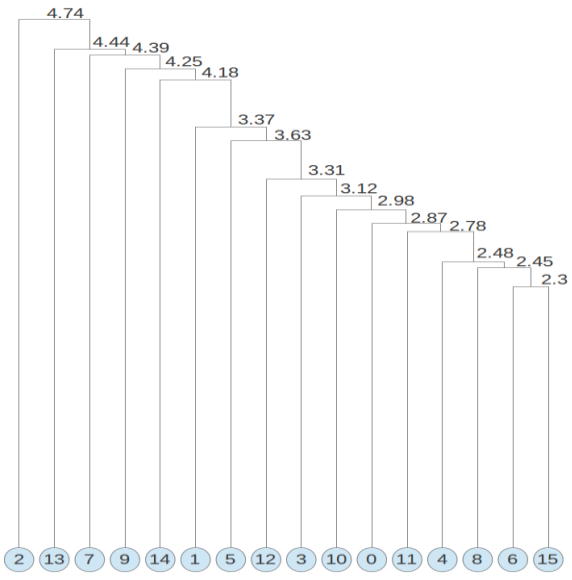

Fig. 4. T_w based Hierarchal cluster represents the communication distances between the virtual instances. The vertices represent the instances and the weighted links are the communication distance between them in microseconds.

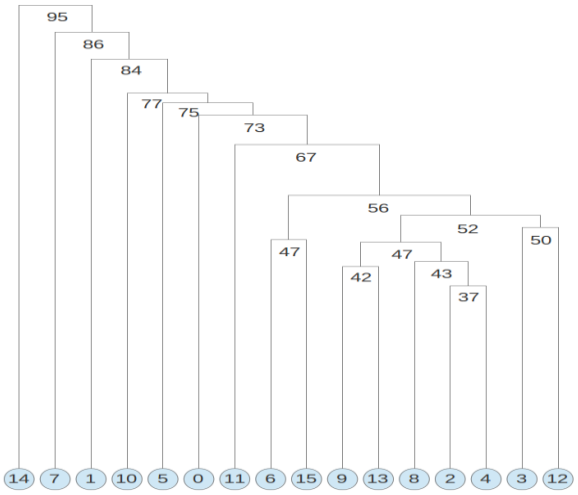

Fig. 5. T_s based Hierarchal cluster represents the communication distances between the virtual instances. The vertices represent the instances and the weighted links are the communication distance between them in milliseconds.

TABLE III. EXECUTION TIMES IN MICROSECONDS OF THE BENCHMARK ON DIFFERENT GROUPS OF INSTANCES WITH DIFFERENT WORKLOADS

Message Size (Byte) / Group name	0	100	512	1K	10 K	50 K
T_w Based Closest VMs	638	1258	1353	1313	1889	7349
T_w Based Furthest VMs	12175	24570	23804	25305	31783	97485
T_s Based Closest VMs	3563	7416	7394	7933	11906	39393
Random Group 1 (0:7)	3510	6335	6607	6428	9283	32393
Random Group 2 (8:15)	2813	5980	5665	6098	8304	30140

The enhancement in the execution time of the benchmark is obvious in Table 3. The closest VMs group according to T_w clustering is better than the other groups by speedups listed in table 4.

The enhancement varies in the range between 4.1 and5.5 speedup for the random chosen instances, where the enhancements are much bigger comparing to the farthest chosen group where the enhancement speedup varies from 13.3 speedupto19.5 speedup according to the applied workload on the benchmark.The execution time of the

benchmark in the previous experiment isaveraged from 1000 times of execution for every message size.

TABLE IV. ENHANCEMENT IN PERFORMANCE IN SPEEDUP FOR THE RANDOM AND WORST GROUPS COMPARING TO THE CLOSEST GROUP BASED ON BOTH T_W AND T_S WITH DIFFERENT WORKLOADS

Message Size (Byte) / Group name	0	100	512	1K	10 K	50 K
T_w Based Closest Vs. Farthest	19.1	19.5	17.6	19.3	16.8	13.3
T_s Based Closest Vs. Farthest	3.4	3.3	3.2	3.2	2.7	2.5
T_w Based Closest Vs. Random1	5.5	5.0	4.9	4.9	4.9	4.4
T_s Based Closest Vs. Random1	0.99	0.85	0.89	0.81	0.78	0.82
T_w Based Closest Vs. Random2	4.4	4.8	4.2	4.6	4.4	4.1
T_s Based Closest Vs. Random2	0.79	0.81	0.77	0.77	0.70	0.77

From the previous results, it is apparent that our model represents the communication time on the virtualized environment on the cloud, and identifies the topology of the used instances accurately, as the execution time of the all-to-all broadcast group communication varies according to the distances between instances in our hierarchical cluster. Also, as we assumed T_s cannot represents the cloud's internal topology as it is the absolute term in the assumed communication model that is affected by many factors that is not related to the nature of communication itself.

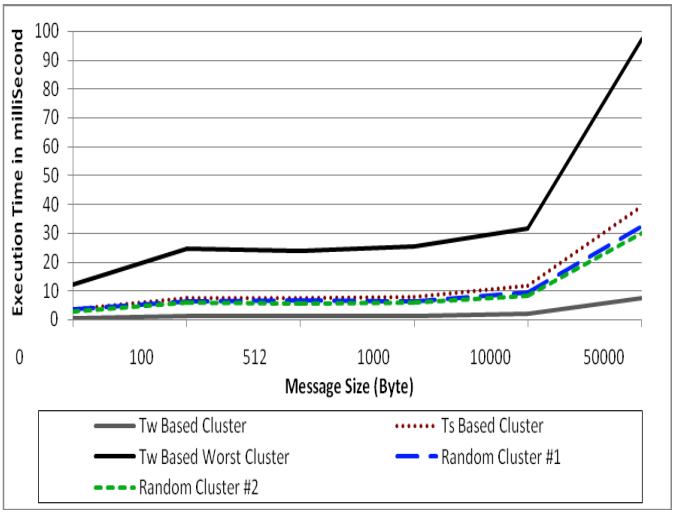

Fig. 6. The difference between execution times for various groups of instances.

IV. RELATED WORK

There have been many attempts in the literature to address HPC on the cloud. The closest approach to this work is the network tomography inference approach. In this approach two main probing mechanisms are usually utilized; the first is round-trip probe (as in our work), and the second is sandwich probing [17]. The latter relies on a sending node transmitting two packets to a destination node separated by a larger packet addressed to an alternate node; the delay perceived by the first destination is correlated with similarity of the two destination nodes. Such information is then used to construct a hierarchical network topology structure.

Battré et al.[16]investigate the use of this approach on a mid-sized experimental cloud setup. They confirmed that the round-trip time probingis more accurate than sandwich probing in the virtualized environment, mainly due to large timing variation introduced by the virtual machine monitors. However, the straightforward application of round-trip probing still suffered significant inaccuracies. Our work proposes an enhanced round-trip probing method, by integrating a simple two-parameter communication model, experimentations, curve fitting, and a generalized cluster hierarchy trees (instead of binary trees). The method runs in parallel, requiring linear-time complexity, rather than the typical underlying quadratic complexity of round-trip probing.

In a second approach the service provider offers high speed interconnected computing clusters, such as Amazon Cluster Computer (CC) machines [7] and Penguin Computing HPC service [8]. The former is investigated by Expósito et al. [9]; they confirmed the cost-efficiency of the Amazon EC2 CC1 and CC2 instance types with respect to computation, point-to-point communication and scalability performance. However, the trade-off here is the much higher service price than standard instances. In our approach we potentially overcome this trade-off by placing the tasks on much closer VMs with the same amount of renting cost (aside from an initial experimental cost including a short-term use of distant VMs, that are terminated afterwards)..

In another approach, the authors considered standard, non-cluster instances. Ostermann et al. [10] have evaluated and analyzed the performance of the Amazon EC2 cloud services for scientific, distributed computing. They found that the current performance is not sufficient to execute a scientific application on the cloud. However, they mainly analyzed the effect of virtualization. Jackson et al. [2] also evaluated the cloud while running HPC applications and showed that the more the HPC communicate the more the performance becomes worse, emphasizing the importance of the network. Our work thus complements these studies by considering the communication aspect on the cloud.

A number of researches have worked in improving the performance of executing the applications that demands clusters of VMs during its execution. Their work was on how to conduct a reliable service available and satisfying the user demanded resources to meet the Service Level Agreement minimizing the communication cost. One of the solutions by Jayasinghe et al. [11] is to model both the data center and the application and input them as a graph of virtual machines and a graph of data center to a placement engine which use the divide and conquer strategy to solve the problem as an optimization problem. Then the mapping of the VMs on data center's physical machines is solvable as a graph-coloring problem. Another work in the same approach is proposed by Tantawi[12]; it is a statistical method to bias the selection of

the samples while performing cross entropy. This resulted in making the cross entropy scalable and linear with the size of the cloud. This approach of enhancement is a server side one, which is contrary to our model that enhance the performance of the cluster based applications with blind view of the real cloud structure and capacityHu et al.[18] present a lightweight system-level method to discover VMs ensembles and their interactions. The information helps provide better VM management, such as by placing communicating VMs in close by physical machines.This work is server side, while our approach is form the client side.

Another approach is to simplify the process of hiring and deploying VMs in cluster structures. An example is a distributed middle-ware called Cyber aide Creative Service [13], which can create distributed VMs on demand as an interface to the cloud web services to create and manipulate the what called cyber infrastructure as a service (CaaS). Another example is Aneka [14], which is an enterprise cloud computing platform and solution to develop on demand scale applications. Vecchiola et al. [15] demonstrate two examples on performing two scientific applications on the cloud using Aneka.This approach facilitates the deployment and running of VMs on cluster structures.Our work can be integrated with one of those systems in this approach, as a future work.

V. Conclusions and Future Work

This paper considers the problem of identifying the virtual machine placement within the cloud provider network. The method does not assume any information provided by the cloud provider, thereby allowing for being largely cloud provider independent.

The method has revealed a large variation of performance for the all-to-all group communication operation, with a ratio of 19.8. Such result confirms the significance of the VM placement problem in the cloud. Moreover, our method resulted in a consistent improvement in the overall performance, ranging from 4.1 to 5.5 speedups.

The main objective of this paper is to assess the possibility of identifying the placement details, and the corresponding achievable benefits. Future work is needed to conduct more elaborate experiments using a large set of benchmarks, as well as further tuning of our model; another interesting area is to extend the model to assess the virtualization load on the computing node, thereby allowing for further tuning/mapping of the parallel tasks to the VMs.

Acknowledgement

This research is partially supported by a PhD scholarship from the Egyptian Ministry of Higher Education (MoHE), and an Amazon AWS Research Grant.

References

[1] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” J. Internet Serv. Appl., vol. 1, no. 1, pp. 7–18, Apr. 2010.

[2] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright, “Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud,” in 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010, pp. 159–168.

[3] L. A. Barroso and U. Hölzle, “The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines,” Synth. Lect. Comput. Arch., vol. 4, no. 1, pp. 1–108, Jan. 2009.

[4] A. Grama, G. Karypis, V. Kumar, and A. Gupta, Introduction to Parallel Computing, 2nd ed. Addison-Wesley, 2003.

[5] G. Gan, Data Clustering in C++: An Object-Oriented Approach, Har/Cdr. Chapman and Hall/CRC, 2011.

[6] “Amazon Web Services, Cloud Computing: Compute, Storage, Database.” [Online]. Available: http://aws.amazon.com/. [Accessed: 17-May-2013].

[7] “HPC Cloud Services | Penguin Computing.” [Online]. Available: http://www.penguincomputing.com/services/hpc-cloud. [Accessed: 17-May-2013].

[8] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, “Performance analysis of HPC applications in the cloud,” Future Gener. Comput. Syst., vol. 29, no. 1, pp. 218–229, Jan. 2013.

[9] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, “A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing,” in in Cloud Computing, D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds. Springer Berlin Heidelberg, 2010, pp. 115–131.

[10] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, “Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-Aware Virtual Machine Placement,” in 2011 IEEE International Conference on Services Computing (SCC), 2011, pp. 72–79.

[11] A. N. Tantawi, “A Scalable Algorithm for Placement of Virtual Clusters in Large Data Centers,” in Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Washington, DC, USA, 2012, pp. 3–10.

[12] L. Wang, D. Chen, Y. Hu, Y. Ma, and J. Wang, “Towards enabling Cyberinfrastructure as a Service in Clouds,” Comput. Electr. Eng., vol. 39, no. 1, pp. 3–14, Jan. 2013.

[13] C. Vecchiola, X. Chu, and R. Buyya, “Aneka: A Software Platform for .NET-based Cloud Computing,” arXiv:0907.4622, Jul. 2009.

[14] C. Vecchiola, S. Pandey, and R. Buyya, “High-Performance Cloud Computing: A View of Scientific Applications,” in 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN), 2009, pp. 4–16.

[15] “Benchmarks | Network-Based Computing Laboratory.” [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/.

[16] D. Battre, N. Frejnik, S. Goel, O. Kao, and D. Warneke, “Evaluation of Network Topology Inference in Opaque Compute Clouds through End-to-End Measurements,” in Cloud Computing (CLOUD), 2011 IEEE International Conference on, 2011, pp. 17–24.

[17] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, “Maximum likelihood network topology identification from edge-based unicast measurements,” in *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, New York, NY, USA, 2002, pp. 11–20.

[18] L. Hu, K. Schwan, A. Gulati, J. Zhang, and C. Wang, “Net-cohort: detecting and managing VM ensembles in virtualized data centers,” in *Proceedings of the 9th international conference on Autonomic computing*, New York, NY, USA, 2012, pp. 3–12.

APPENDIX

```
for msg in {0,2^0,2^1, ..., maxMessageSize {
     for t = 0 : numberTestCases{
          for turn = 1 : numProcesses{
               peer = processRank xor turn;
               if processRank > peer{
                    for i = 0 : numberRepetitions{
                         receive(workload from peer);
                         send(workload to peer);
                    }
               }else{
                    start recording time;
                    for i = 0 : numberRepetitions}
                         send(workload to peer;(
                         receive (workload from peer);
                    }
                    stop recording time;
               }
               times[t][peer] = (elapsed time)/numberRepetitions;
          }
     }
     commTime[msg][processRank][peer] = min(times[all testcases][peer]);
}
```

Fig. 7. Hardware probing algorithm.

We developed our hardware-probing module to mainly measure the point-to-point communication times among *numProcesses* process in parallel. The module also measures many statistical and experimental metrics mentioned in the methodology section. The user enters a set of parameters to measure the communication time accurately;the parameters are:

- *numberTestCases*: represents the number of measures to be recorded for the given message size (set to 10),
- *numberRepetitions*: represents the number of repetitionsin each send operation (set to 100),
- *maxMessageSize*: specifies the maximum workload (message size) to stop at (set to 128kb).

The module outputsa 3D communication values matrix that provides the communication times among process pairs, '*i*' and '*j*', at message size *msg*. Such a value is addressed by *commTime*[*msg*][*i*][*j*].

The Probing Algorithm:

Fig. 7 shows the used algorithm. The algorithm probes communication time *numberTestCases* times, for every message size in the range from 0, 1 to *maxMessageSize* (bytes),in doubling step sizes. A reading time of sending and receiving a message *numberRepetitions* times.

The algorithm takes the minimum of these readingsin the curve fitting stage; finally the algorithm stores the minimum communication time in a 3D matrix called *commTime*.