DA-MCM: A Dynamic Application-Aware Mechanism for MPI Communications in Cloud Environments

Laura Espínola[1], **Daniel Franco**[1], **and Emilio Luque**[1]
[1]Computer Architecture and Operating System Department
Universidad Autónoma de Barcelona (UAB)
Barcelona - Spain

Abstract— *Cloud computing offers interesting characteristics for scientific parallel applications, which require high performance computing (HPC). Most of these kinds of applications are implemented using Message Passing Interface (MPI). The major drawback of MPI applications on cloud is the loss of performance due to virtualized networks, affecting the messages communication latency time. Current available optimizations based on network topology information for MPI applications are not suitable for cloud architectures. This paper presents a Dynamic Application-Aware mechanism for MPI Communications Management (DA-MCM) in cloud. DA-MCM improves the communications latency time avoiding congested paths through the dynamic selection of alternative paths; and helps users to detect mapping and parallel implementation issues. An evaluation of the proposed solution is carried out in public and private clouds.*

Keywords: Cloud environments, MPI Communications, User mitigation processes, parallel applications

1. Introduction

Scientific parallel applications usually require a lot of compute resources to resolve complex problems. To the users of these applications, cloud offers convenient access to a large amount of resources by renting the computing power, instead of acquiring and maintaining physical clusters [1]. Currently, there are several cloud providers that offer characteristics such as the pay-as-you-go accounting, elastic and flexible resource provisioning, along with ubiquitous computing and storage to fulfil different users' customization needs for their parallel application execution in cloud.

The migration of HPC parallel applications to cloud environments comes with several advantages, but despite considerable research efforts to improve application execution in this kind of environment [2], virtualization introduced in cloud still affects its performance. Currently, many HPC scientific parallel applications use Message Passing Interface (MPI) standard protocol to perform their communications. In cloud environments, latency of MPI communications between nodes is increased, resulting in a severely degraded system performance [3]. A key challenge of HPC in cloud is to consider the lack of efficient communication support in virtualized networks [4].

HPC application communications are characterized by intensive periods of message interchange, as opposed to a randomly constant packet injection [5]. This bursty traffic can produce hot-spot situations, where some network resources are quite congested while others remains idle. An efficient distribution of the communications is an issue that needs to be dealt with executing in cloud. The increase in the application's messages latency time in virtualized networks and the bursty traffic can not be afforded using traditional optimization methods, like topology aware algorithms, due to lack of information about the underlying layers. Furthermore, as a cloud virtualization system uses a dynamic network flow scheduling, even static topology information is not sufficient to represent the network [6].

In this paper, a Dynamic Application-Aware MPI Communication Management (DA-MCM) mechanism is presented. It improves the communications latency time for MPI applications running in public and private clouds by the selection of alternative paths in case of congestion. The mechanism also presents a user mitigation process that helps users to detect when an improvement in the application mapping or parallelization can be made.

The selection of alternative paths is performed taking into account network topology characterization, which obtains information about the underlying network configuration; and by monitoring the application processes synchronization along with its communication pattern. This information allows the proposed mechanism to detect link usage of the virtual network, differentiating when a network or process synchronization problem occurs. The main goal is to achieve a dynamic distribution of the application messages avoiding congestions.

DA-MCM is transparent for user applications. It intercepts the messages, then verifies the link usage and processes synchronization in order to dynamically select a less congested path for each sent message, using the updated topology characterization information and based on communication locality of MPI applications running in clouds. The evaluation of the technique is presented, using benchmarks and real applications, in the Amazon EC2 public cloud environment and in a private cloud built with OpenStack.

This paper is organized as follows: in Section 2, a study of MPI Communications in cloud is presented, then in Section 3, a description of the designed DA-MCM mechanism is detailed. The experimental evaluation of the technique is carried out in Section 4, along with a discussion of the results. Finally, conclusions and future work are stated in Section 5.

2. Cloud Communications Analysis

The mechanism presented is based on network topology characterization and in the existence of communication patterns during application execution. Both affect communication performance of parallel applications in cloud, due to repetitive usage of the same links and the bursty traffic created. As network topology is hidden from the user's application, DA-MCM has to find out which pair of virtual instances that are used during the application execution can communicate faster than others.

2.1 Network Characterization

A main procedure for the proposed mechanism is to characterize the underlying network topology in order to obtain alternative paths if congestions appear. A previous analysis of virtual nodes interconnection performance is made by executing the MPI Ping Pong application without any disturbance from another application in order to evaluate the distance between virtual nodes. This information is useful for corroborating the existence of alternative paths.

A matrix of latency distances is created using the MPI Ping Pong application's message latency time. The matrix is filled with the base latency time between virtual nodes. For N virtual nodes, we need $N^2 - 1$ latency evaluations of ping pong messages between them, in order to get all pair-to-pair performances.

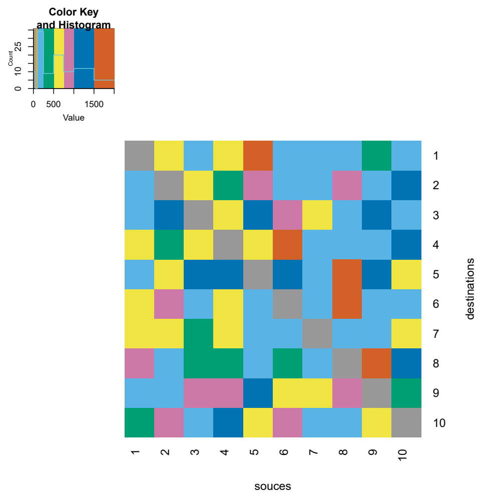


Fig. 1: Heat map of network status

We evaluate latency using ping pong messages of 1 megabyte (MB) size, and 20 instances t2.medium of Amazon EC2. The latency results between nodes are shown in the heat map (Fig. 1). It represents the current status of links between virtual machines and it was obtained with the ping pong of MPI messages between each source and destination, without another application running in the system.

Alternative paths can be selected using the results shown in the heat map. This analysis shows that alternative paths can be used to avoid congestion situations when parallel applications are running in a cloud environment.

2.2 HPC Applications Communication Pattern

Recent studies about scientific parallel applications observe a repetitive behavior in HPC applications. This repetitive behavior is based on computing and communications phases found during application execution [7]. When the MPI processes of an application are launched, they perform compute and communications following a strong periodic pattern. Generally, the processes communicate in a non-uniform way, causing an imbalance of link usage. This problem decreases available bandwidth between nodes and generates performance degradation of MPI applications [8].

The MPI communication locality behavior is represented by fundamental phases of communication during the application execution (e.g. a set of source / destination pairs of communications). An example of this assumption occurs with the NAS CG benchmark. It has relevant phases that contains communication between nodes, which conform to a pattern, as depicted in Fig. 2. The repetitive behavior of these phases represents a significant time of the application execution. Any congestion that affects the communications within the application phases can importantly disturb the application execution time.

The repetitive communication pattern of parallel applications presents an unbalanced use of the network links. Most of the optimized algorithms for MPI operations implemented for a cloud environment consider groups of virtual machines, the distance between them and the process mapping into

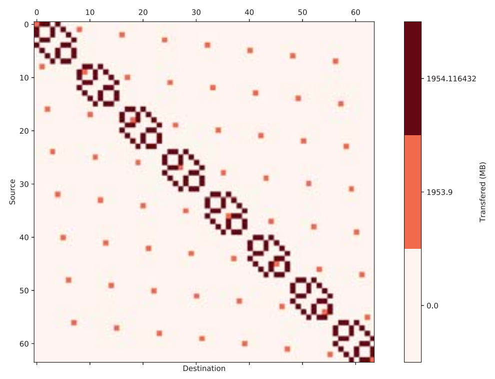


Fig. 2: NAS-CG Communication Pattern

nodes to solve the problem [6]. However, none of them perform MPI message management to obtain a balanced distribution of the traffic or a better application process synchronization.

2.3 Justification

The analysis of the underlying network characterization and the repetitive MPI parallel application communication pattern allow us to create a mechanism that manages MPI communications of parallel applications in cloud, providing a dynamic distribution of the messages and a better synchronization between processes following the application communication pattern. The mechanism captures MPI communications and forwards them through alternative paths if required, taking into account the network topology characterization performed and it tries to balance the application communications. The mechanism proposed achieves better performance of MPI parallel applications in cloud without adding significant overhead.

3. Dynamic Application-Aware Mechanism for MPI Communications in Cloud

In cloud environments, MPI applications experience loss of performance caused by the increase in communications latency time in virtual networks. The Dynamic Application-Aware mechanism for MPI communications in cloud (DA-MCM) improves communications latency time by managing messages sent by the application's processes. The mechanism uses alternative paths to avoid congestions.

DA-MCM has modules in charge of the System Characterization, which is performed prior the application execution. The modules obtain the application communication pattern, and with it, characterize the underling network topology. The characterization obtains an approximation of the underlying network, measuring the distance between nodes, based on the study of communication latencies between processes, in order to recognize possible alternative paths for messages sent by the application processes. During the application Runtime, alternative path are constantly updated and when congestion situations are detected, alternative paths are selected to avoid congested links, achieving better distribution of MPI communications in cloud.

During the application Runtime, the DA-MCM mechanism introduces a user mitigation process that helps users detect issues regarding the application mapping and parallel implementation, leveraging the methodology of MPI Communication Management (MCM) [9]. In Fig. 3, it is possible to observe the summary of processes performed by DA-MCM, both prior to and during the application execution.

3.1 System Characterization

System characterization includes processes that are executed prior application execution, such as profiling operations. They are in charge of the ***Application communication pattern*** recognition, the ***Network Topology Characterization*** and the ***Alternative Paths Table Creation***. DA-MCM uses two matrixes to create the alternative path table; the Latency Distance and the Sensibility Matrixes. The matrices are obtained based on the message sizes of the application's communication pattern with more repetitiveness during its execution, improving the viability of the alternative paths. The Fig. 4 shows the system characterization process.

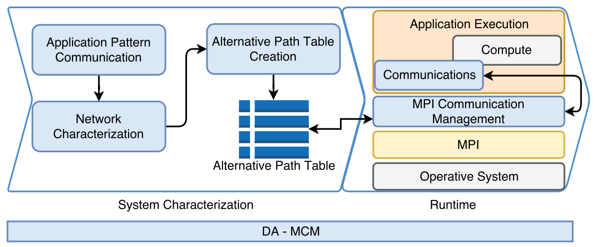


Fig. 3: Dynamic Application-Aware Mechanism

The application communication pattern is obtained building a histogram with the sizes of messages sent between application's processes. The representative phases of the parallel applications are used to build the histogram, by executing the applications with the performance prediction tool PAS2P [7], in order to avoid a complete application execution. With application communication pattern recognition, only relevant phases are analyzed. A large number of messages over time, or frequent communication operations, may also indicate that the application execution will be affected by network contention in cloud environments.

Once the application communication pattern has been obtained, DA-MCM uses the message size from communications phases with more repetitiveness in order to characterize the underlying network topology. This characterization is performed taking into account that network performance of MPI application processes are related to the network performance of their corresponding virtual machines location. A MPI Ping Pong application is executed using the message sizes obtained from the application communication pattern, without disturbance from another application.

The main idea is to build a Latency Distance Matrix using ping pong communications, with message sizes obtained from the application communication pattern. The ping pong

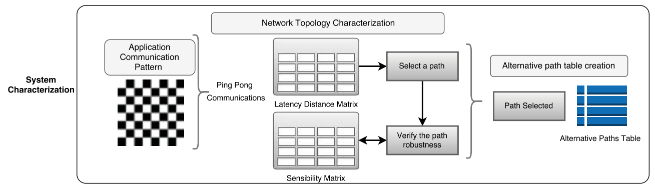


Fig. 4: DA-MCM: System Characterization

is performed between processes placed in different virtual machines of the virtual cluster. Each intersection of rows and columns represents a path between one pair of source and destination nodes, obtaining a basic detail of distances between virtual machines in terms of latency time. A process is located on each virtual machine, so for N virtual machine, N iterations of ping pong latency evaluation are needed in order to obtain all pair-to-pair latencies.

The Sensibility matrix is also created with the MPI Ping Pong application execution results. This matrix stores as a weight, in the intersection of rows and columns, the number of times in which a path between each virtual node is affected by communication congestions between any other pair of nodes, representing its robustness.

Routes for the Alternative Paths Table are selected, considering the lowest latency value obtained from a source to a destination, through an intermediate node. The robustness of the routes are also evaluated during the alternative path selection process. For example, the path with the lowest latency distance is obtained from the Latency Distances Matrix, then its robustness is verified using the Sensibility Matrix. If the path is efficient, it is stored into the alternative paths table. The alternative path table is then updated during the application execution to avoid its information becoming obsolete due to the cloud usage variability.

3.2 Runtime Dynamic Communication Management

DA-MCM includes modules that are executed during the application Runtime, in order to monitor the virtual network status and detect congestion situations. A $DA-MCM$ daemon is launched together with the application's processes. There is a daemon for each virtual machine in the execution environment, and it is in charge of the application's processes communication executed in the virtual node. DA-MCM also provides Notification and User Mitigation capabilities, which help the users detect application mapping and parallel implementation issues.

The Monitoring process checks all messages sent by the application processes, but only acts when source and destination processes are located in different virtual machines, without disturbing application. It measures message latency time and maintains the virtual network link status between each virtual machine updated.

When a message arrives at its destination, a Notification process is activated, if congestions occurs in the message path. It sends an Acknowledge (ACK) notification to the source, with the path latency measured during the monitoring process. Hence, the ACK notification contains the current state of each path used in terms of latency. With this information, the alternative path table is updated, thus constantly maintaining the actual status of each virtual link. Using this strategy, the technique tries to avoid the loss of information relative to the paths due to cloud variability.

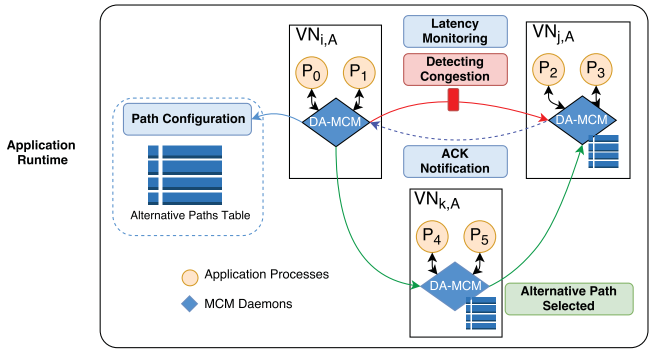


Fig. 5: Application Runtime

Initially, messages are sent through regular paths, but if congestion scenarios are detected, alternative paths are selected. This procedure is called Path Configuration. It involves the configuration of new alternative paths according to latency values and it is performed at source nodes. When several alternative paths are found for a pair of source and destination virtual nodes, a selection of the best alternative path has to be carried out using the alternative path table. Therefore, new messages are sent through the selected alternative path avoiding congestion. This selection procedure distributes messages among the configured paths in previous tasks. Figure 5 shows the whole process.

3.2.1 ACK Notification Process

The dynamic communication management mechanism sends an ACK notification only when it is really necessary. A strategy is introduced to determine the effective situation when an ACK has to be sent or not, in order to avoid causing an imbalance to the application execution. The strategy consist of considering two different cases, the time when a intercepted message sent arrives to its destination daemon, and the time when the application destination process actually requests the message.

If the intercepted message arrives before the application destination process requests the message, the message arrives in time. In this case, the ACK notification process is not needed, due to the fact that the path is actually without congestion or the application destination process does not have to wait for the message. In the Fig. 6, it is possible to observe this case where $t_r \approx t_{rq}$, in which t_r is the message reception time in the destination daemon and t_{rq} is the application destination process request time. Another scenario in which the ACK notification is avoided happens when $t_r < t_{rq}$, in this case the message arrives before the application process destination requests it.

Assuming the previous scenario, an ACK notification is only needed when $t_r > t_{rq}$, hence an ACK is sent to the source daemon, with the latency path information to maintain the path status. In this case, the message arrives

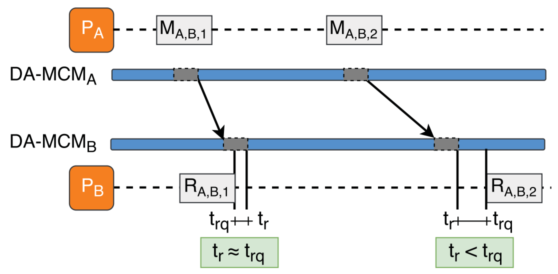


Fig. 6: Avoiding ACK notification

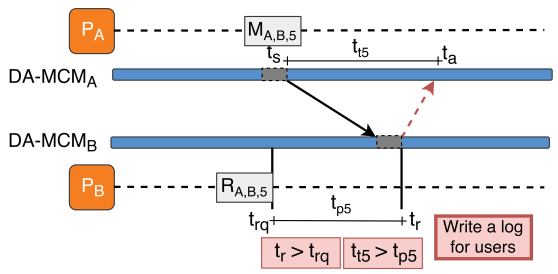


Fig. 8: DA-MCM Mitigation processes

after the application process destination requests it, denoting a scenario that causes the application process destination having to wait for the message, involving a possible congestion scenario that has to be analyzed in case another message uses the same path. This scenario is shown in Fig. 7.

3.2.2 User Mitigation Process

DA-MCM includes an user mitigation process that helps users detect issues in the application implementation; and in the mapping applied to the application processes in cloud. When application processes take more time for communications between them compared to the computation they perform, there is a high probability of finding a better parallel design to implement the application or improve the application processes mapping configuration. Using DA-MCM, it is possible to obtain values for message transmission and processing time of every process in the application, which can be used to notify users about possible issues.

The technique can calculate each message transmission time t_t, using t_s and t_a. It includes the sending and waiting time for the ACK notification from the destination daemon $DA - MCM$. The period of time that it takes for the destination daemon $DA - MCM$ to deliver the message to the application destination process is also measured using $t_r and t_{rq}$, and it denotes the processing time t_p. Both measures are shown in the Fig. 7.

It is possible to observe that when $t_t \approx t_p$, the application behavior is correct. The same happens when $t_t < t_p$, the message transmission process takes less time than the message processing time. In both cases, no problem of process desynchronization in the application execution exists, and it is possible that a congestion is detected for the DA-MCM technique.

The User mitigation process is only activated when $t_r > t_{rq}$. Figure 8 shows an example of $DA-MCM$ performing the mitigation. If the transmission time is greater than the processing time of the message ($t_t > t_p$), a possible desynchronization between application processes exists, and it may be caused by issues in the application implementation or mapping mapping configuration. In this case, DA-MCM writes a log to inform the application user.

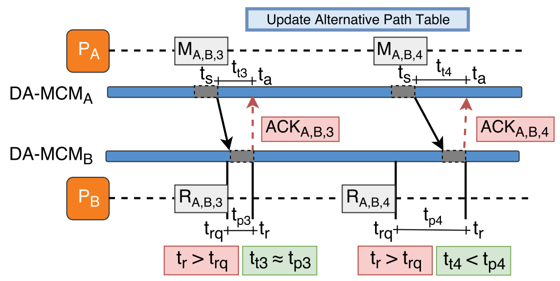


Fig. 7: ACK notification

4. DA-MCM Evaluation

To analyze the functionality of the proposed approach, a series of experiments evaluating latency of MPI communications and the total execution time of applications running on cloud are conducted. Table 1 describes main system components and configurations used. The experiments where performed in a public and private cloud.

Experiments are performed with different sets of software, from the lowest layer to the top. In Amazon EC2 public cloud, instances are launched using StarCluster 0.95.6 [10]. This is an open source cluster-computing toolkit for Amazon EC2. On the other hand, for the private cloud, the instances are built with OpenStack Newton. For this cloud the corresponding version of the OpenStack projects used to build the cloud environment are Nova 14.0.6, Neutron 6.0.0, Glance 2.5.0 and Heat 1.5.1. The instances use Debian 8 as its operating system.

Table 1: Experimental Setup

Component	Amazon EC2		OpenStack
Instance	t2.medium	c3.2xlarge	m1.medium
Memory	4 GiB	15 GiB	4 GiB
Storage(GB)	EBS	EBS	40GB
Network	Low to Moderate	Low to Moderate	10Gbps
Physical Processor	Intel Xeon 2.5GHz	Intel Xeon 2.5GHz	Intel Xeon 2.5GHz
vCPU	4 vCPU	8 vCPU	2vCPU

4.1 Applications

a) NAS parallel Benchmark Conjugate Gradient (CG): It is configured with classes B, C and D. The NAS-CG is selected due to its repetitive pattern of communications between processes. NAS-CG Class B is used with 75000 rows, 1000 iterations, 60 Eigenvalue shifts and 13 non-zeros. Then NAS-CG Class C is configured with 150000 rows, 1000 iterations, 110 Eigenvalue shifts and 15 non-zeros. Hence, the native NAS-CG Class D (Fortran) implementation of the kernel have been compiled with 1500000 rows, 100 iterations, 500 Eigenvalue shifts and 21 non-zeros using the GNU compiler.

b) NBody Simulation: It represents a dynamical system of particles, usually under the influence of physical forces. It is implemented as a circular pipeline with a communication pattern. The application was configured with 100000 particles and 50 iterations.

4.2 Results

NAS-CG and NBody are executed in 16 nodes in each kind of cloud. The t2.medium and c3.2xlarge instances are used in Amazon EC2; and in the private cloud mounted with OpenStack, m1.medium instances are employed. The experiments expose the benefits of the DA-MCM method for the application execution time, by improving the processes messages latency time when congestions are detected. The results also show the overhead added and they evaluate the scalability of DA-MCM, by testing it with different problem sizes of NAS-CG.

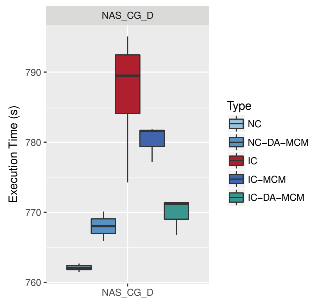
Evaluations are carried out to test the functionality of DA-MCM and to measure the added overhead, in scenarios without congestion (NC), without congestion and applying DA-MCM (NC-DA-MCM), with congestion (IC) and with congestion applying DA-MCM (IC-DA-MCM). The congestion is created with the bursty traffic generated by the application itself and by delays in links that connect the virtual nodes. Finally, an improvement is presented with the DA-MCM mechanism compared to a well-know state of the art technique, MCM (IC-MCM) [9].

For the NAS-CG Class D execution, a study of the improvement in terms of execution time and communication time is carried out using virtual nodes with c3.2xlarge as instance type. In Fig. 9(a), it is possible to observe there is no significant added overhead in the application execution time, comparing no congestion scenario (NC) with the scenario applying DA-MCM without congestion (NC-DA-MCM). On the other hand, when congestion occurs, a reduction of 16% of the delay in the application execution time is observed, comparing scenario with congestion (IC) and the scenario with congestion and applying DA-MCM (IC-DA-MCM). The results show the advantage of avoiding congestion during the application execution taking into account the total execution time.

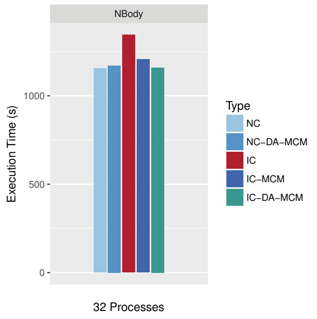
The same behavior is shown for the NBody application executed using m1.medium instances in the OpenStack private cloud. In Fig. 9(b), NBody simulation executed with DA-MCM reveals approximately 2% of overhead compared to execution without DA-MCM. Hence, in a scenario in which congestion appears, a reduction of approximately 9% is obtained applying DA-MCM. The introduced DA-MCM mechanism improves the previous MCM approach, as observed in the presented results.

Figure 10 shows the communication time of the NAS-CG application processes. It is possible to observe how the technique improves communications in the case of congestion scenario compared to the execution without applying DA-MCM. The figure depicts that high latency hot-spots (red) are turned into lower latency ones (blue), denoting the reduction of communication time between processes, when the proposed technique is applied.

Mechanism scalability is evaluated, testing it with different problem sizes of the NAS-CG benchmark with different



(a) Amazon EC2



(b) OpenStack

Fig. 9: DA-MCM analysis

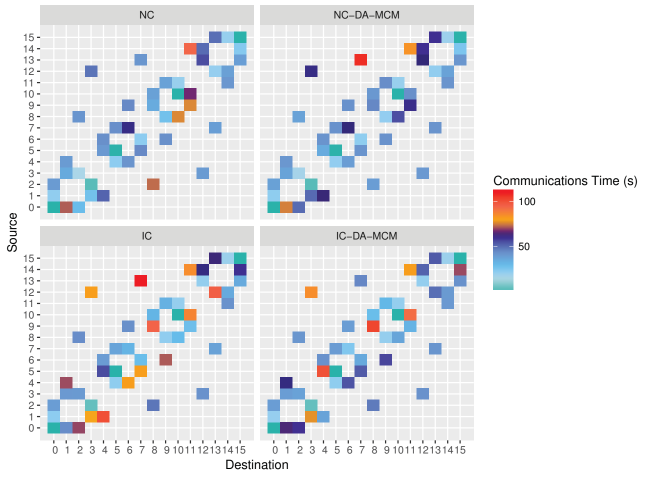


Fig. 10: Communications between processes

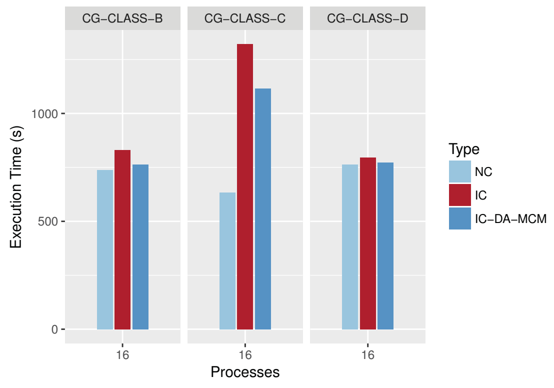


Fig. 11: DA-MCM scalability

instance types. For Class B execution, using t2.medium instances without congestion gives an execution time of 252.74 seconds. Meanwhile, the same application executed with congestion has an execution time of 782.33 seconds. When the DA-MCM is applied to avoid a congestion situation, an execution time of 655.15 seconds is obtained (Fig. 11). Class C execution obtains results with similar behavior. The DA-MCM method reduces the NAS-CG Class B execution time by 16.2%, compared to execution with congestion. For Class C execution, the mechanism reduces the execution time by 12.7%. A similar pattern of the mechanism is observed in the execution of the same application with a larger problem size (Class D) and using instances with more resources(c3.2xlarge), denoting the benefits of DA-MCM in multiple scenarios.

DA-MCM improves the application execution time when congestion occurs in the virtual links of a cloud environment, providing users with better usage of virtual network resources. Lower message latency time allows us to obtain improved application execution time. The key concept is to avoid the congested links, enhancing the usage of non-congested links.

5. Conclusions and Future Work

The challenge exposed in this paper is how to improve HPC application communications for cloud environments. This is necessary due to the loss of performance that applications experience in virtualized environments. A better management of MPI communications in virtual networks must be implemented to improve performance of MPI applications in cloud.

The presented mechanism improves the performance of MPI applications in cloud and provides the user's application with a mitigation process which enables the detection of issues related to the application processes mapping; as well as parallelization implementation. DA-MCM provides the management of MPI communications in virtual network environments, without modifying the user's application implementation. Furthermore, DA-MCM characterizes the underlying network topology and looks for alternative paths under congestion situations, keeping the status of the paths between virtual machines constantly updated. Future work will look at improving the mechanism using different technologies in private clouds, such as containers and with this, reducing the costs without losing application performance.

Acknowledgments

This research has been supported by the MICINN/MINECO Spain under contracts TIN2014-53172-P and TIN2017-84875-P and it has been partially supported by the CloudMas as Government Competency of AMAZON Web Services (AWS).

References

[1] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting Cloud Performance for HPC Applications: A User-Oriented Approach," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CC-GRID)*. IEEE, may 2017, pp. 524–533. [Online]. Available: http://ieeexplore.ieee.org/document/7973739/

[2] E. Roloff, M. Diener, A. Carissimi, and P. O. A. Navaux, "High performance computing in the cloud: Deployment, performance and cost efficiency," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, Dec 2012, pp. 371–378.

[3] K. Dashdavaa, S. Date, H. Yamanaka, and E. Kawai, "Architecture of a High-Speed MPI Bcast Leveraging Software-Defined Network," *Euro-Par 2013: Parallel Processing Workshops: BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013.*, pp. 885–894, 2014.

[4] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, "Performance analysis of hpc applications in the cloud," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 218–229, 1 2013.

[5] G. Rodriguez, R. Beivide, C. Minkenberg, J. Labarta, and M. Valero, "Exploring pattern-aware routing in generalized fat tree networks," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: ACM, 2009, pp. 276–285. [Online]. Available: http://doi.acm.org/10.1145/1542275.1542316

[6] Y. Gong, B. He, and J. Zhong, "Network Performance Aware MPI Collective Communication Operations in the Cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 3079–3089, 2015.

[7] A. Wong, D. Rexachs, and E. Luque, "Parallel Application Signature for Performance Analysis and Prediction," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 2009–2019, jul 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6827943/

[8] K. Takahashi, D. Khureltulga, Y. Watashiba, Y. Kido, S. Date, and S. Shimojo, "Performance evaluation of sdn-enhanced mpi allreduce on a cluster system with fat-tree interconnect," *2014 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 784–792, 7 2014.

[9] L. Espínola, D. Franco, and E. Luque, "ScienceDirect MCM: A new MPI Communication Management for Cloud Environments," *Procedia Computer Science*, vol. 108, pp. 2303–2307, 2017. [Online]. Available: http://ac.els-cdn.com/S1877050917306075/1-s2.0-S1877050917306075-main.pdf?_tid=b082d058-628f-11e7-a9ae-00000aacb35f&acdnat=1499375674_860fa5d607d6a8e8f07110a190882fdb

[10] "Starcluster documentation (v.0.95.6)," http://star.mit.edu/cluster/index.html, accessed: 2016-11-28.