Universitat Autònoma
de Barcelona

Escola d'Enginyeria.

Departament d'Arquitectura de Computadors i Sistemes Operatius

Efficient Communication Management in Cloud Environments

Thesis submitted by ***Laura María Espínola Brítez*** for the degree of Philosophiæ Doctor by the Universitat Autònoma de Barcelona, under the supervision of the ***Dr. Daniel Franco Puntes***, done at the Computer Architecture and Operating Systems Department, PhD. in Computer Science.

Bellaterra, July, 2018

Efficient Communication Management in Cloud Environments

Thesis submitted by ***Laura María Espínola Brítez*** for the degree of Philosophiæ Doctor by the Universitat Autònoma de Barcelona, under the supervision of the ***Dr. Daniel Franco Puntes***, done at the Computer Architecture and Operating Systems Department, PhD. in Computer Science.

Barcelona, July, 2018

Dr. Daniel Franco Puntes
Thesis Advisor

Laura María Espínola Brítez
Thesis Author

I would like to dedicate this thesis to my loving husband Jorge Villamayor, who has always helped me and believed that I could do it. To my mom and dad, examples of perseverance, and to my family from Paraguay that support me from the distance.

Thanks God for giving me the strength to keep going!! ⌣̈

Acknowledgements

I sincerely would like to thank my supervisor Dr. Daniel Franco Puntes, not only for his guidance, advice and confidence in me, but also for his genuine support. I cannot adequately express my thanks for his help and interest in seeing me succeed in all my endeavors. Thanks Dani!, for your accurate words when I needed.

I also want to thanks Nokia Bell Labs - Ireland for giving me the opportunity to participate in an industrial research lab. Specially to Diego Lugones and his research team, who adopt me as one more in his team. This experience show me how science and industry are linked to create revolutionary ideas. Ireland is an unforgettable country, I will never forget the kindness and cordiality of its people.

To the people at the Department of Computer Architecture and Operating Systems (CAOS), for always being helpful. Also to the people of my house of studies, Universidad Nacional de Asunción, for giving me a strong knowledge cements.

To my husband for the love and support of always, for joining me in this trip, which seemed something very crazy at the beginning. Thanks for sharing the same dreams and being part of this great adventure. Because when we are together, the impossible becomes possible.

Thanks to my family for the support received since I started this long journey. Thanks to my parents Juan Alberto and María Dolores, for simply being there always, even when it seemed impossible. Thanks to my brothers Claudia and Albert for making me laugh at times when everything seemed so difficult. Thanks to my grandparents Brigida and Anastasio for support me with their prayers, and also to those who accompany me from heaven grandma Blanca and grandpa Ercilio.

To my colleges and closer friends, for their actions and words of support throughout this years. It is impossible to name everyone, because it was a long journey, and no one crosses your path by chance and you do not enter anyone's life without any reason.

The merit of successfully finishing a career is a privilege that only those who struggle with faith and love understand.

Thanks!

Abstract

Scientific applications with High Performance Computing (HPC) requirements are migrating to cloud environments due to the facilities that it offers. Cloud computing plays a major role considering the compute power that it provides, avoiding the cost of a physical cluster maintenance. With features like elasticity and pay-per-use, it helps to reduce the researchers procurement risk.

Most of HPC applications are implemented using Message Passing Interface (MPI), which is a key component in common and distributed computing tasks. However, for this kind of applications on cloud environments, the major drawback is the lost of execution performance, due to the virtualized network that affects the communications latency and bandwidth.

To use a cloud environment with scientific applications of this kind, low latency communication mechanisms are required. The network topology detail is not available for users in virtualized environments, making difficult to use the existing optimizations based on network topology information done in bare-metal cluster environments. In some cases, cloud providers can migrate virtual machines, which impacts the efficiency of routing optimizations and placement algorithms. Moreover, if resource isolation is not guaranteed, resource sharing can lead to variable bandwidth and unstable performance.

In this thesis a Dynamic MPI Communication Balance and Management (DMCBM) is presented, to overcome the communication challenge of HPC applications in cloud. DMCBM is implemented as a middle-ware between the users application and the execution environment. It improves message communication latency times in cloud-based systems, and helps users to detect mapping and parallel implementation issues. Our solution dynamically rebalances communication flows at higher levels of the virtualized HPC stack, e.g. over MPI communications layer, to dynamically remove communication hot-spots and congestion in the underlying layers.

DMCBM abstracts the communications state between application processes based on latency measurements. This middleware characterizes the underlying network topology and analyzes parallel applications behavior in the cloud. This allows for detecting network congestion and optimizing communications by either selecting alternative

communication paths between processes, or leveraging live migration of virtual machines in cloud environments. These options are analyzed in real-time and selected according to the type of congestion (link or destination). DMCBM achieves lower application execution time in case of congestion, obtaining better performance in clouds.

Finally, experiments that verify the functionality and improvements of DMCBM with MPI Applications in public and private clouds are presented. The experiments where done by measuring execution and communication times. NAS Parallel Benchmarks and a real application of dynamic particles simulation NBody are used, obtaining an improvement of up to 10% in the execution time and a communication time reduction of about 40% in congestion scenarios.

Keywords: Cloud Computing, High Performance Computing, MPI Communications, Virtual Networks, Communication Management.

Resumen

Las aplicaciones científicas con requisitos de High Performance Computing (HPC) están migrando a entornos cloud debido a las ventajas que ofrece. El cloud computing juega un papel importante teniendo en cuenta la potencia de computo que proporciona, debido a que evita el costo de mantenimiento asociado a un clúster físico. Con características como la elasticidad y el pago por uso, el cloud ayuda a los investigadores a reducir el riesgo de adquisición de recursos físicos.

La mayoría de las aplicaciones de HPC se implementan mediante el standard Message Passing Interface (MPI), siendo este un componente clave para tareas de computo distribuido. Sin embargo, ejecutar aplicaciones MPI en entornos cloud tienen como principal desventaja la pérdida de rendimiento durante la ejecución, debido a la virtualización de la red, que afecta la latencia de las comunicaciones y el ancho de banda.

Para usar un entorno cloud con aplicaciones científicas de este tipo, se requieren mecanismos de comunicación que permitan baja latencia. La topología de red no está disponible para los usuarios en entornos virtualizados, lo que dificulta el uso de las optimizaciones existentes para bare-metal clusters, basadas en la información de topología de red. En algunos casos, los proveedores de cloud pueden migrar máquinas virtuales, lo que afecta la eficacia de las optimizaciones de enrutamiento y los algoritmos de ubicación. Además, si no se garantiza el aislamiento de los recursos, el intercambio de recursos puede generar un ancho de banda variable y un rendimiento inestable.

En esta tesis se presenta Dynamic MPI Communication Balance and Management (DMCBM), un middleware para resolver la pérdida de performance en comunicaciones de aplicaciones HPC en cloud. DMCBM se implementa como un software intermedio entre la aplicación de los usuarios y el entorno de ejecución. Mejora los tiempos de latencia de comunicaciones en sistemas cloud y ayuda a los usuarios a detectar problemas de mapping y ejecución paralela. Nuestra solución re-equilibra dinámicamente los flujos de comunicación a niveles superiores de la pila de HPC virtualizada, sobre la capa de comunicaciones MPI, para eliminar dinámicamente los hot-spots de comunicación y la congestión en las capas subyacentes.

DMCBM abstrae el estado de las comunicaciones entre los procesos de la aplicación en función de las mediciones de latencia. Este middleware caracteriza la topología de red subyacente y analiza el comportamiento de las aplicaciones paralelas en cloud. Esto permite detectar la congestión de la red y optimizar las comunicaciones seleccionando rutas de comunicación alternativas entre procesos o aprovechando la migración de máquinas virtuales en entornos cloud. Estas opciones se analizan en tiempo real

y se seleccionan según el tipo de congestión (enlace o destino). DMCBM logra un menor tiempo de ejecución de la aplicación en caso de congestión, obteniendo un mejor rendimiento en el cloud.

Finalmente, se presentan experimentos que verifican la funcionalidad y las mejoras de DMCBM con aplicaciones MPI en cloud públicos y privados. Los experimentos se realizaron midiendo los tiempos de ejecución y comunicación. Para los experimentos se utilizan dos aplicaciones: NAS Parallel Benchmarks y una aplicación real de simulación dinámica de partículas NBody, obteniendo una mejora de hasta 10% en el tiempo de ejecución y una reducción del tiempo de comunicación de aproximadamente 40% en escenarios de congestión.

Keywords: Cloud Computing, High Performance Computing, Comunicaciones MPI, Virtual Networks, Gestión de Comunicaciones.

Resum

Les aplicacions científiques amb requisits d'Informàtica d'alt rendiment (HPC) estan migrant als entorns clouds a causa de les instal · lacions que ofereix. El cloud computing juga un paper important tenint en compte el poder de computar que proporciona, evitant el cost d'un manteniment de clústers físics. Amb característiques com l'elasticitat i el pagament per ús, ajuda a reduir el risc d'adquisició dels investigadors.

La majoria d'aplicacions HPC s'implementen utilitzant Message Passing Interface (MPI), que és un component clau en tasques informàtiques comunes i distribuïdes. No obstant això, per a aquest tipus d'aplicacions en entorns en el núvol, el principal inconvenient és el rendiment perdut de l'execució, a causa de la xarxa virtualitzada que afecta la latència de les comunicacions i l'ample de banda.

Per utilitzar un entorn en el núvol amb aplicacions científiques d'aquest tipus, es requereixen mecanismes de comunicació de baixa latència. El detall de topologia de la xarxa no està disponible per als usuaris en entorns virtualitzats, dificultant l'ús de les optimitzacions existents en funció de la informació de topologia de xarxa realitzada en entorns de clúster de metall sense format. En alguns casos, els proveïdors de cloud poden migrar màquines virtuals que afecten l'eficiència de les optimitzacions d'encaminament i els algorismes d'ubicació. A més, si l'aïllament de recursos no està garantit, l'ús compartit de recursos pot provocar un ample de banda variable i un rendiment inestable.

En aquesta tesi es presenta un Dynamic MPI Communication Balance and Management (DMCBM), per superar el desafiament comunicatiu de les aplicacions HPC en el núvol. DMCBM s'implementa com un mitjà entre la aplicació dels usuaris i l'entorn d'execució. Millora el temps de latència de la comunicació de missatges en sistemes basats en núvol i ajuda als usuaris a detectar problemes de mapeig i implementació paral · lels. La nostra solució reequilibra dinàmicament els fluxos de comunicació a nivells superiors de la pila HPC virtualitzada, p. Ex. sobre la capa de comunicacions MPI, per eliminar dinàmicament les puntes de comunicació i la congestió a les capes subjacents.

DMCBM resumeix l'estat de comunicacions entre processos d'aplicació basats en mesuraments de latència. Aquest middleware caracteritza la topologia de la xarxa subjacent i analitza el comportament d'aplicacions paral · leles al cloud. Això permet detectar la congestió de la xarxa i optimitzar les comunicacions, ja sigui seleccionant camins de comunicació alternatius entre processos o aprofitant la migració directa de màquines virtuals en entorns de cloud. Aquestes opcions s'analitzen en temps real i es seleccionen segons el tipus de congestió (enllaç o destinació). DMCBM aconsegueix

un menor temps d'execució de l'aplicació en cas de congestió, aconseguint un millor rendiment en núvols.

Finalment, es presenten experiments que verifiquen la funcionalitat i millores de DMCBM amb aplicacions MPI en clouds públics i privats. Els experiments que realitzem mesurant els temps d'execució i comunicació. NAS Parallel Benchmarks i una aplicació real de la simulació de partícules dinàmiques NBody s'utilitzen , obtenint una millora de fins a 10% en el temps d'execució i una reducció de temps de comunicació d'aproximadament 40% en escenaris de congestió.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

HPC	High-Performance Computing
IT	Information Technologies
MPI	Message Passing Interface

Chapter 1

Introduction

> *"I started my life with a single absolute: that the world was mine to shape in the image of my highest values and never to be given up to a lesser standard, no matter how long or hard the struggle."*
>
> – Ayn Rand, *Atlas Shrugged, 1957*

High Performance Computing (HPC) is used to obtain much higher performance than traditional computers. Most of the HPC systems are used to resolve computational intensive application problems in many topics, such as: science, engineering or business, among others. This kind of applications needs an efficient and maintainable execution environment to obtain good results.

Nowadays Cloud environments present many attractive characteristics for HPC applications, but also have many challenges relative to the performance obtained in this kind of environments. This thesis addresses one of the issues of HPC applications in cloud. We are assuming as hypothesis that one of the main problems is the loss of performance due to the communications carried out through virtual networks and causing an increase of communications latency.

This work analyzes how HPC communications are carried out in cloud and proposes a method to improve the communications latency time. The solution allows the users to achieve better performance for HPC applications in cloud environments.

1.1 HPC in Cloud Computing

HPC needs a lot of compute power and this comes with the demand of many resources. It is used in many scientific, academic and commercial applications. Although, HPC users are limited by the cost, access to supercomputers is highly expensive due to

the startup and maintenance cost. In addition, Cloud provides many features that avoid risks caused by under-provisioning of resources or wastage of resources (including energy) resulting from under-utilization of computing power [19].

Cloud computing is a paradigm currently used by many distributed and parallel applications, especially in High Performance Computing (HPC) field [30, 42]. There are many reason why users of HPC are migrating their applications to this environment. Some of them are:

- Pay-as-you-go accounting: HPC users only need to pay for the resources that they use, and without need of maintenance them.
- The inherent elasticity and flexibility: Dynamically acquire new resources according to the applications demand. In cloud it is possible to use virtualization and high availability of resources to fulfill the requirements of the application. For consumers, computing resources become immediate, they can use them to scale up whenever they want, and release them once they finish to scale down.
- Energy efficient: Resources are pooled and shared by multiple consumers avoiding the waste of compute power.

Cloud environment is the synergistic combination of distributed computing and communication to provide computing capacity as a service on demand. It has potential to transform a large part of IT industry, changing the way IT hardware is designed and purchased [3]. In order to leverage economy of scale, as well as reducing in-house infrastructure and accelerating time to market.

Currently, cloud is growing beyond cost benefits as providers are offering high-performance resource types including compute or memory-optimized instances, as well as GPU or FPGA options [1] and even bare-metal environments [35, 38]. All these factors, make cloud computing an attractive paradigm for High Performance Computing [18].

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 [29]. Currently, there are several cloud providers that offer characteristics to fulfill different user's customization needs for their parallel application execution in cloud.

Virtualization techniques are needed in cloud providers infrastructures, in order to offer IT services. Consequently, bare-metal configurations are not visible from the

users layer. In this context, optimizations to improve parallel application executions are quiet difficult to design.

In shared environment environments like clouds, users can acquire resources with certain QoS. Although, the bare-metal resources are probably been shared among other user clients. It is expected to have different behaviors in the execution of users applications when running on local bare-metal clusters compared to cloud environment. For instance, even two systems (bare-metal cluster and cloud) with similar configurations in processors, memory and cluster node sizes, can perform differently [23].

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 [4, 27, 40], virtualization introduced in cloud still affects its performance. A key challenge of HPC in cloud is to consider the lack of efficient communication support in virtualized networks [11].

1.2 The Challenge of HPC in Cloud

HPC applications require an acceptable execution performance, and one of the main factor that can affect its execution is how communications are carried out [33, 36]. Many HPC applications are highly sensitive to message communication, due to the repetitiveness of their communication pattern during the execution. This repetitive

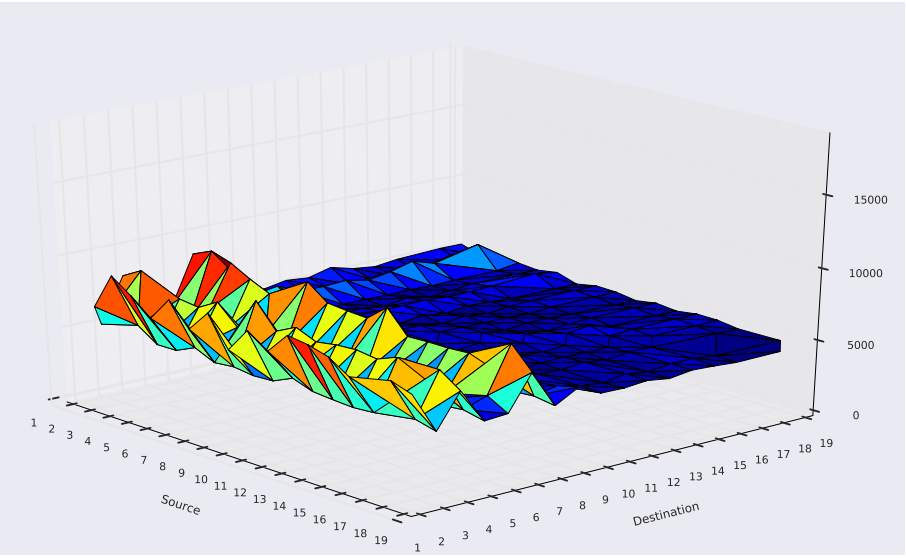


Fig. 1.1 MPI *Ping Pong* without congestion

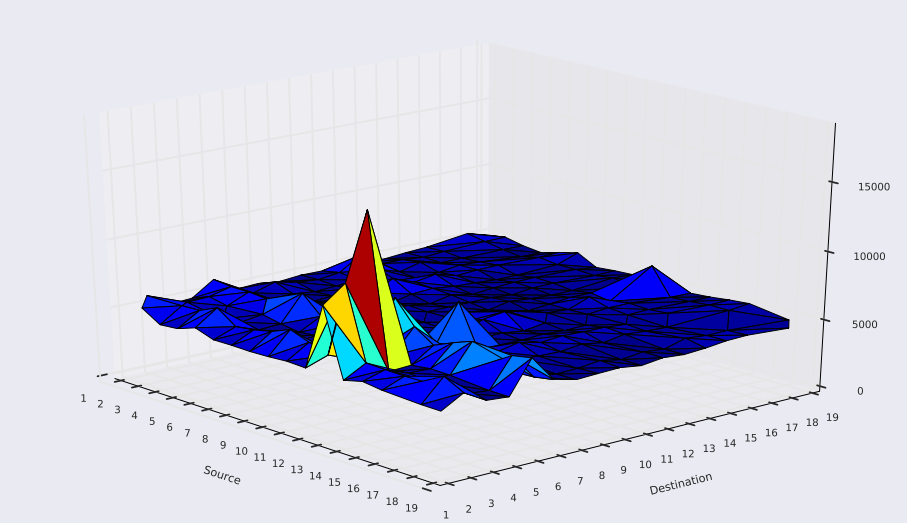


Fig. 1.2 MPI *Ping Pong* with congestion

behavior on the entire application can lead to situations in which the scalability depends on the efficient support of network communications [18].

In cloud computing, HPC applications are executed in a entire virtualized environment. The virtual networks are one of the main characteristics of this kind of environments. For this reason a key challenge of HPC in Cloud is to consider the lack of efficiency in communication support on virtualized networks. This problem inhibit parallel applications to take advantage from high performance networks causing a degradation of their messages communication [11, 23].

Currently many distributed and parallel applications in HPC use the standard protocol Message Passing Interface (MPI) to perform its communications [15]. MPI applications typically require low latency and high bandwidth inter-processor communication to achieve better performance. In case of Cloud, the effect of network virtualization results on a bottleneck for HPC applications implemented with MPI [19]. This trouble with MPI communications can even cause some processes become idle. If this issue is not efficiently controlled, message latency time on HPC applications is increased and system performance is severely degraded [5, 23].

There are many studies in this area that verify the degradation of performance in MPI applications communications caused by network virtualization in cloud. In Q. He et al. [23] for example, they compare the end-to-end delay of MPI communications on clouds with a NCSA cluster: the message latency is below to 50 microseconds for the cluster and above 100 microseconds in cloud systems.

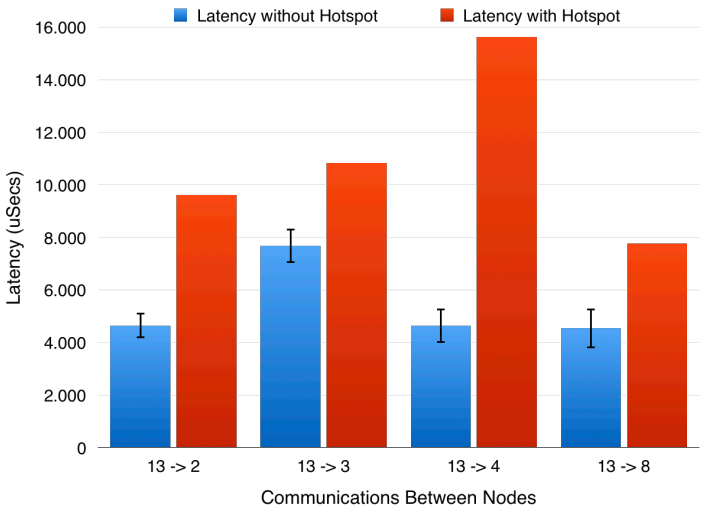


Fig. 1.3 MPI *Ping Pong* with congestion

To exhibit the problem, we run a MPI parallel application called *Ping Pong*, which exchange messages between all processes of the application. The base latency of application's messages, running in cloud environment (Amazon EC2) without any disturbance from another application is shown in Fig. 1.1. Then, we measure process communication message latency of the same application, although this time we add a constant traffic between two nodes of the cloud: node 1 and 17. The generated traffic creates congestions in other processes communications that are in other nodes of the virtual cluster, causing an effect in terms of message latency time. Fig. 1.2 shows the disturbance caused on processes communications in nodes: 2, 3, 4, 8 and 13, when the constant traffic is created. We illustrate latency variation on affected nodes in Fig. 1.3.

The increase of message latency time in virtualized networks can not be afforded with traditional optimization methods like topology aware algorithms because this information is in underlying layers, and hidden from users. Furthermore, as cloud system virtualization uses a dynamic network flow scheduling, even static topology information is not sufficient to represent the network [16].

The loss of communications performance for HPC in cloud environments may not produce the results as planned, for example the expected execution time, causing even more cost to the users. These concerns justify the research of solutions in this area.

1.3 Objetives

1.3.1 General Objetive

In this thesis the main objective is to reduce the communication latency of MPI applications in cloud environments, providing a method to manage communications.

1.3.2 Specific Objectives

Several specific objectives are elaborated to cover the main goal of this thesis. The objectives are selected in order to understand the basic nature of HPC communications in cloud and how they can be improved. These specific objectives are the following:

- Analyze the factors that affect the execution of parallel applications in virtualized networks on cloud environments.
- Designing a mechanism that improve MPI communications in cloud.
- Evaluate the designed mechanism in terms of functionality and scalability.

1.4 Methodology

The research methodology followed in this thesis is oriented to design, implement and analyses experimental scenarios in order to improve MPI communications in cloud. The experimental methodology is integrated by a set of methodical and technical activities that are carried out to collect information and necessary data on the subject to be investigated and the problem to be solved. The research in this thesis is framed in the academic program of the Universitat Autònoma de Barcelona.

This methodology will be divided into stages described below:

- Bibliographical analysis of the state of the art: At this stage, current approaches on the subject to be investigated are searched, it is sought to delve into the challenges and obstacles in the field. The topics on which the research would be carried out are: cloud environments, virtualized networks, SDN and techniques that improve the performance in the management of MPI messages in the network.
- Data collection and analysis: This point refers to a comparative study of existing techniques. Data analysis consists of organizing and processing information so that it can be described, analyzed and interpreted. With the knowledge obtained from this phase we have a guide to create our solution.

- Design and propose a method that improves the management of HPC communications in Cloud environments.
- Evaluation and verification of the proposed solution compared to the existing proposals: An evaluation will be carried out through benchmarks of the results obtained.

1.5 Contributions

The work developed for the thesis has been published in the following papers:

- **MPI Communications Management in Cloud** - Laura Espínola, Daniel Franco, and Emilio Luque in ***Jornadas de Paralelismo Sarteco, Cordoba-Spain, 2015***. [6]

  This paper presents an study of the feasibility of how to improve latency and network traffic for parallel MPI applications running on cloud. In addition an study of the behavior of MPI communications in Amazon EC2 public cloud is also presented.

- **MPI Communications Management in Cloud** - Laura Espínola, Daniel Franco, and Emilio Luque in ***The International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas-USA, 2015***. [7]

  The paper presents a methodology that provide dynamic management of MPI communications in cloud environments. It proposes an alternative path distribution considering the latency and possible congestion but keeping in mind the characteristics of the cloud.

- **Improving MPI Communications in Cloud** - Laura Espínola, Daniel Franco, and Emilio Luque in ***ACM-W Europe WomENcourage 2016 Celebration of Women in Computing, Linz-Austria, 2016***. [8]

  In this article we present the architecture structure of a communications management method for MPI applications, based in the study of communication latencies between processes.

- **MCM: A new MPI Communication Management for Cloud Environments** - Laura Espínola, Daniel Franco, and Emilio Luque in ***International***

Conference on Computational Science (ICCS), Zurich-Switzerland, 2017. [9]

In this paper a method named MPI Communication Management (MCM) is presented. This method characterizes the underlying network topology and analyzes parallel applications behavior in the cloud, improving the application's messages latency time.

- **DA-MCM: A Dynamic Application-Aware Mechanism for MPI Communications in Cloud Environments** - Laura Espínola, Daniel Franco, and Emilio Luque in ***The International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas-USA, 2018***. [10]

  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. It introduces a user mitigation process that helps users detect issues regarding the application mapping and parallel implementation.

1.6 Thesis Outline

As mentioned previously, the main aim of this thesis is to analyze one of major issues of HPC applications in cloud, the loss of communication performance in MPI, and proposed a solution to obtain better latency communication time in cloud environments. This thesis is organized as follows:

Chapter 2: Thesis Background

It presents the main Cloud computing characteristics and concepts, as well as tools and technologies to build clouds. Also introduces MPI applications, how they are implemented, and how they are actually carried out in cloud.

Chapter 3: Dynamic Communication Balance and Management in Cloud (DMCBM)

In this chapter the proposed solution is described. The details of the presented method DMCBM are expose, with it modules and main processes. Also, DMCBM implementation details and its software architecture are explained.

Chapter 4: Evaluation

The experimental scenarios are described with the applications selected and the softwares that were used. The main area of this chapter is the analysis of results, that verify the functionality of the proposed solution.

Chapter 5: Conclusions and Future Works

Present the thesis conclusions and summarizes selected research challenges that could be addressed as future work.

Chapter 2

Thesis Background

"Forms disappear, words remain, to signify the impossible."
– Augusto Roa Bastos, *I, the Supreme*

In this chapter, we give a global vision of cloud computing and HPC applications in order to frame the thesis.

First the main characteristics of cloud are explained, then cloud implementations models are summarized. Following by some examples of cloud infrastructures, in order to put in context the infrastructures in which our solution is applied. Finally the HPC applications and how they are migrating to cloud is explained, and also the state of the are of improving the execution of this type of applications in cloud are described.

2.1 A Global Vision of Cloud Computing

Cloud computing is a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [32].

2.1.1 Characteristics and Services

Cloud environments have some essential characteristics like:

- On-demand self-service: Services of the cloud are ready for the user whenever the user demands, without having to have human interference.

- Broad network access: Users of the cloud services are able to have access to the services over a broad network and by using their workstations, mobile or tablets, laptops, etc).
- Resource Pooling: Cloud services are pooled to serve several users in a multi-tenant environment. Resources are dynamically assigned and reassigned and there is only a high level of abstraction for the location of the resources.
- Rapid Elasticity: Resources on the cloud appears to be unlimited for the user. According to the user demand or even automatically the resources can scale up or down.
- Measured Service: There is a metering system in the cloud that monitors, controls and reports service usages for both users and providers.

2.1.2 Implementation Models

Some of the implementations models for cloud computing defined in National Institute of Standards and Technology (NIST) definition of cloud computing [32] and use in general are the cited here:

Public implementation model: This implementation model is often treated as the real definition of cloud computing, but this assumption is not true, because all the implementations models are considered valid by the NIST definition of cloud computing. The resources always belong to a provider and they are shared within its various users. The provider defines its policies and prices and the users need to agree with them to have access to the resources [37]. The user of this model can be an entire organization or just a single person, because it is possible to allocate just one small, and cheap, machine.

This model is adherent with all the essential characteristics. Mainly the pay-per-use is totally fulfilled, because there are no upfront costs to the user. The elasticity is granted by the provider infrastructure and, for the user, it seems to be unlimited. The access to the cloud is made through an Internet connection. The user control is defined by the provider, and the user needs to use his credentials to have access to the resources. Usually each user has unique credentials, even when they are users of an organization with a corporate account.

Private implementation model: A private cloud is a model where the physical resources are controlled by the user, a single user or an organization. The most common scenarios of using this model is an organization that decides to virtualize its data center (on-site configuration), or if an organization wants to rent all the resources from a provider (outsourced scenario). In both scenarios, the user has exclusivity over the physical resources.

In terms of cost, this model is the most expensive of all. This occurs because the user needs to buy and maintain the resources by itself or rent exclusive resources from a provider. The resources are dedicated to the organization, then there are no other users accessing it. This model achieves the same level of isolation and security than the user of traditional computational resources. Those characteristics make this model the most secure of all.

Hybrid Cloud: Having critical information and applications in the private cloud within the corporate and having shared applications and non-critical data on a public cloud helps businesses to take advantage of a secure private cloud as well as cost benefits of a public cloud. This approach is called a Hybrid Cloud.

Community Cloud: This model can be considered as a private cloud shared between several organizations with the same policy. Data will still be private and secure within the community but the organizations can benefit from sharing the cost between each other.

2.2 Cloud Infrastructures

In this section some examples of cloud infrastructures are detailed. These infrastructures represents examples of cloud environments in which our solution is applicable. They can be classified as infrastructures for public and private clouds.

2.2.1 An overview of Amazon EC2 cloud infrastructure

One of the main target platform for numerous academic and commercial applications is Amazon Web Service (AWS). It introduces Elastic Compute Cloud (EC2) that offers a public cloud platform with powerful compute and storage resources on demand through hardware level virtualization [22, 23].

There are several kinds of virtual machine instances to lease in Amazon EC2. The instances provides to users a highly customizable operating system environment. The

users have complete control of their leased instances. These characteristics allows users run distributed data analysis applications, scientific simulations and HPC applications. Some large physics experiments such as STAR [46], or weather research and Forecasting (WRF) model [13] simulations, have experimented with building virtual-machine-based clusters using Amazon EC2.

Amazon provides to users the choice of multiple instance types, operating systems, and software packages. Amazon EC2 allows to select a specific configuration in relation to memory, CPU, instance storage, and the boot partition size optimal for their choice of operating system and applications. These characteristics allows users run distributed data analysis applications, scientific simulations and HPC applications [1] .

The relative processing performance is given in Elastic Compute Units (ECU) in Amazon EC2. One ECU corresponds roughly to the equivalent CPU capacity of a 1.0 - 1.2 GHz 2007 Opteron or 2007 Xeon processor. The smallest machine can be instantiated with 1.7 GB RAM and Gbit Ethernet. It has a single virtual core with a compute power of 1 ECU. The largest machine offers 60 GB RAM, 10 Gbit Ethernet and 88 ECUs, based on the actual dual eight-core Intel Xeon CPU systems [31].

These features allows to select Amazon EC2 as case of study to understand network performance in the public cloud environment. In this work, the instance for general purpose, t2.micro and t2.medium are used, along with the c3.2xlarge instances, dedicated to HPC exclusive. The general and HPC purpose instances are used to probe that our proposed solution is capable to improve the applications performance in both kind of instances. The selected instances offer balance of compute, memory, and network resources to test the proposed solution cost and benefits.

2.2.2 OpenStack cloud infrastructure

In private clouds a platform to offer infrastructure as a service (IaaS) is also presented. A representative project that build this kind of architecture is OpenStack, although there are also others like OpenNebula. OpenStack is powering many of the world's most notable science and research organizations [27]. It is notable that many research and science disciplines are the main use cases in OpenStack cloud architecture, it provides compelling solutions for many of the challenges of delivering flexible infrastructure for research computing.

HPC requires high compute power and cluster networking; storage, networking access to large volumes of data and infrastructure manageability. OpenStack supports these needs and it is constantly updated to expand services in order to fill gaps. Many researchers are able to use this architecture which fulfill their requirements. The

dynamic, automated nature of software-defined infrastructure cuts away time wasted on the distractions of setup and enables researchers to maximize the time they spend on research itself [45].

OpenStack allows the segregation between cloud users and projects that requires isolation due to its multi-tenancy support, which is integrated in the architecture. Despite the isolation that the architecture offers, the hardware is shared among the virtualized resources that are available to the users. HPC requires high compute power and cluster networking; storage, networking access to large volumes of data and infrastructure manageability. OpenStack supports these needs and it is constantly updated to expand services in order to fill gaps. Many researchers are able to use this architecture which fulfill their requirements. The dynamic, automated nature of software-defined infrastructure cuts away time wasted on the setup distractions and enables researchers to maximize the time they spend on research itself [45]. OpenStack allows the segregation between cloud users and projects that requires isolation due to its multi-tenancy support, which is integrated in the architecture. Despite the isolation that this architecture offers, hardware is shared among the virtualized resources that are available to users.

2.3 HPC applications in cloud

Cloud-technology are rapidly advancing, it presents significant opportunities for HPC. The cloud models can leverage benefits to the system managers for they HPC environments [29]. Cloud enables workload demand, increase system utilization, and makes high performance computing system accessible to a wider community [40], including:

- Scale to better support application and job needs with automated workload-optimized OS provisioning.
- Provide simplified self-service access for broader set of users that also reduces management and training costs.
- Accelerate collaboration or funding by extending HPC resources to community partners without their own HPC systems
- Enable pay-for-use with show-back and chargeback reporting for actual resource usage by user, group, project, or account.

Figure 2.1 represent a basic idea of how HPC is migrating to cloud from bare-metal clusters environments. One example is that HPC users can deploy or destroy instances

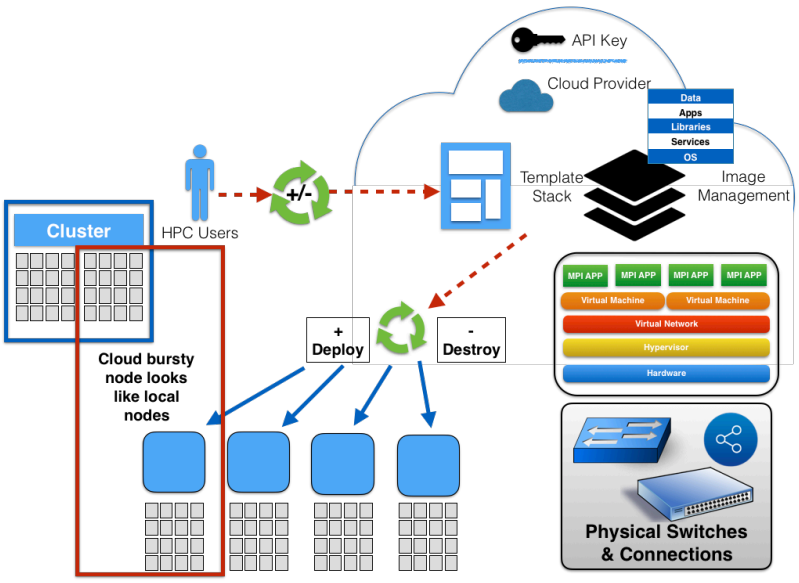


Fig. 2.1 HPC migration to cloud

of virtual machines with their corresponding OS and applications like in the figure. This dynamic characteristic gives to HPC users the ability to "burst" workloads onto the cloud when in-house computing resources run tight, enabling them to leverage the resources in public and private clouds to continue running their applications without having to invest in more infrastructure to meet peak demand [39].

2.3.1 HPC with MPI communications in cloud

HPC applications use MPI standard to communicate their messages. It have been used for around two decades. Point to point and collective communications are provided to processes by this interface. Although, the network communications are abstracted from the processes, hence applications have not control or information about the underlying network.

When the MPI processes of an application are launched, they communicate through a *Communicator*, which is part of the MPI implementation. The standard MPI message transmission model is illustrated in Fig. 2.2.

Generally these processes communicate in a not uniform way, allowing unbalance of link usage. This problem decreases available bandwidth between nodes and generate performance degradation of MPI applications [44].

Most of the optimized algorithms for MPI operations implemented for a cloud environment are specialized either for latency or throughput, considering groups of

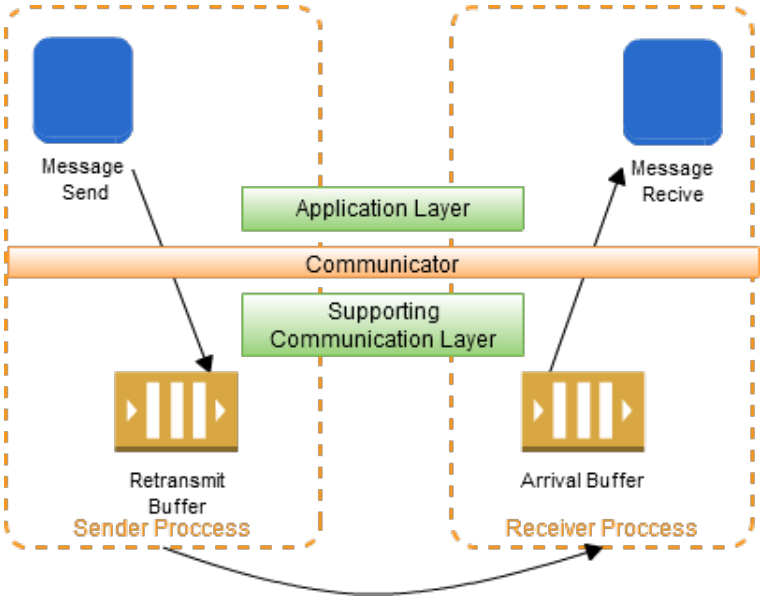


Fig. 2.2 MPI message transmission

virtual machines and the distance between them, but do not work for the manage of the message [16].

2.3.2 MPI 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 [47]. 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 [44].

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 a pattern, as depicted in Fig. 2.3. 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.

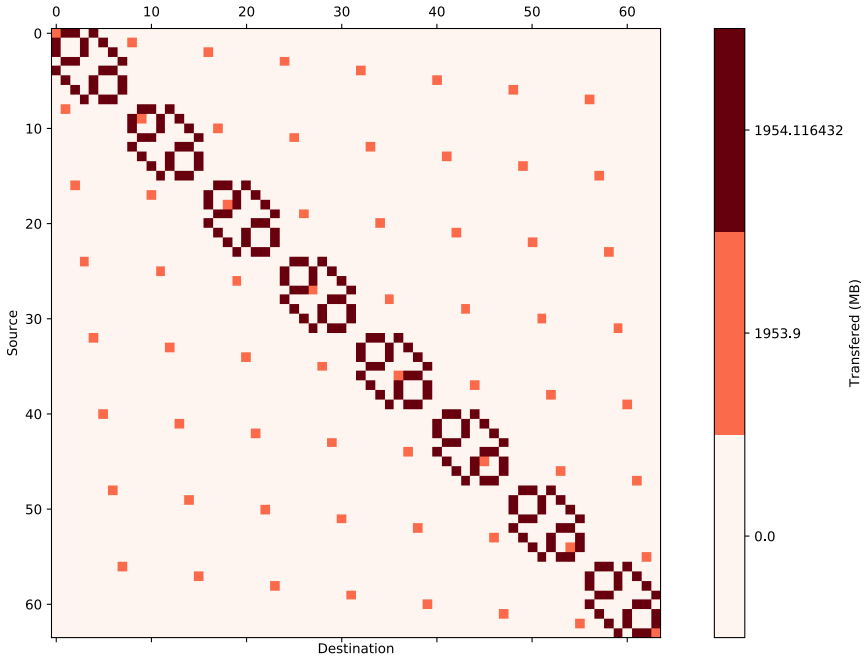


Fig. 2.3 NAS-CG Communication Pattern

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 nodes to solve the problem [16]. However, none of them perform MPI message management to obtain a balanced distribution of the traffic or a better application process synchronization.

2.3.3 Current research improving MPI communications in cloud

This section introduces the main topics in which the proposed method is based. First the performance of HPC applications running in cloud is studied. Then available solutions to gain more performance in cloud are described, with their main advantages and disadvantages.

HPC applications had been successfully executed in cloud environments, though with performance drawback, mainly due to the lack of network communication performance, limiting applications scalability [11, 24]. Research effort is being made to overcome this issue, by proposing techniques to group, balance and map parallel tasks among virtual clusters.

In order to determine which kind of HPC applications are well suitable to run in cloud platforms, Gupta et al. [19] and Jackson et al. [25] have performed evaluations. Results, shown the performance-cost trade-offs of scaling taking in to account the virtualized architecture on Amazon EC2 cloud. Furthermore, there are studies of HPC applications on Microsoft Azure cloud under different configurations and cluster sizes. One of this studies is presented by Hassan et al. [21], where they measure the performance in terms of MOPS, speedup, latency and bandwidth. The performance loss due to the poor latency of communications in virtualized networks on the cloud is a common issue reported in all the studies.

There are many topology-aware algorithms to improve HPC communications in bare-metal cluster environments [26, 43, 48, 49]. For example in Nuñez et al. [34], an effective method to control network efficiency is presented. The method controls network congestion based on paths expansion, traffic distribution, application patterns and speculative adaptive routing. However, inefficiency in virtualized networks on cloud can not afford with traditional optimization methods, like topology aware algorithms because this information is in the underlying layers, ratter hidden from the users.

Regarding the improvement of HPC communications in cloud environments, Gupta et al. [20], propose the load balancing of parallel tasks distributed among virtual machines. In this research, they develop a methodology to dynamically perform a distribution of parallel tasks implemented in Charm++, by measuring CPU loads, task loads, and idle times from the execution environment. The proposed solution focuses on parallel applications that are implemented using MPI library and depends on a message-driven object-oriented parallel programming system (Charm++).

Another approach to improve MPI communications in cloud are based in network performance-aware algorithms, for a series of collective communication operations including broadcast, reduce, gather and scatter [16]. In this research, they model the network performance among different VMs pairs in the cloud environment, creating an hierarchy of grouped VMs. According to this hierarchy, network performance-aware collective communications were developed, modifying the MPI library implementation. There is also, other research that aims to improve the performance of HPC in cloud, in which, the MPI library is modified: Hassani et al. [22] creates a new MPI version using parallel Radix sort. However, any of them perform the management of MPI messages to obtain better latency times, in a transparent way to the users, without modifying the MPI library.

In Zhang et al. work [50], a High Performance implementation of MPI library is designed to work with SR-IOV enabled InfiniBand virtual clusters. It dynamically

detects VM locality and coordinates data movements between SR-IOV and Inter-VM shared memory (IVShmem) channels, to meet the demand of high performance communication on virtualization environments. Our work aims to a transparent solution to the users, not requiring specific virtualization components, nor the installation of a particular MPI library.

Nowadays, there are also solutions that overcome the MPI communication degradation in cloud using virtual machine live migration [14, 51]. From Gomez-Folgar et al. [14], the main idea that we got from this works is that the communication pattern of the application plays an important role regarding communication balance between VMs. They perform a live reallocation of the application processes before its execution, by first study the MPI application via an MPI profiling tool. Although, their solution do not offer a dynamic reallocation during the application execution, in order to maintain the same performance due to the cloud interconnection network sharing characteristic. On the other hand, Zhang et al. [51], presents an innovative solution that performs VMs live migration during the application execution but still requires modifications in the MPI library and also adds to the application a delay, suspending the communication messages channel.

All the works presented in this section gives us a base line to overcome the main goal of this thesis. First, the study of MPI applications execution performance degradation in cloud denotes that its major drawback is related to communications in virtual networks. Also, it allows to understand why and where this becomes an issue. Finally, each of the possible solutions that exist for the improvement of MPI communications, with its pros and cons, allow us to create a new solution to solve the problem successfully.

2.4 DMCBM: a new method to improve MPI Communications in cloud

In this work, a transparent middleware for MPI application running in cloud is proposed. It is transparent for the application and the MPI library. The solution captures MPI communications and forwards them in congestion scenarios, or use the virtual machine live migration technique trying to balance communications between VMs. It looks for a better performance of MPI applications in cloud environments.

The proposed solution, Dynamic MPI Communication Balance and Management (DMCBM), tries to improve latency communication time for MPI applications running in public and private clouds. The analysis of the underlying network characterization and the repetitive MPI parallel application communication pattern allows the method

to create a mechanism, which is capable of manage MPI communications of parallel applications in cloud, detecting and avoiding congestion situations.

The middleware provides a dynamic distribution of the messages and a better processes synchronization following the application communication pattern. The mechanism captures MPI communications and forwards them through alternative paths if congestion scenarios appear, taking into account the network topology characterization performed. It tries to balance the application communications taking in to account process allocated on each VM. The mechanism proposed achieves better performance of MPI parallel applications in cloud without adding significant overhead. The middleware runs at user level and therefore can be applied at any cloud environment, and is explained with more details in the Chapter 3

Chapter 3

Dynamic Communication Balance and Management in Cloud

"Imagination is the Discovering Faculty, pre-eminently. It is that which penetrates into the unseen worlds around us, the worlds of Science."

– Ada Lovelace, *1841*

3.1 Introduction

HPC applications communications performance represents a challenge on cloud environments. In this kind of environments, network virtualization tend to increment the HPC communications latency. A middleware is presented to overcome HPC applications communications performance degradation on cloud, specifically for MPI applications. The proposed solution is called *Dynamic MPI Communication Balance and Management* (DMCBM). The middleware balances, distributes communications and migrates tasks of MPI applications running on cloud. In order to avoid congestion situations, reducing the communications latency.

HPC application's communications are mostly dynamic and usually relies on underlying routing mechanisms to balance and optimize the use of network resources in bare-metal HPC systems. In cloud, however, the underlying topology is either obfuscated by the cloud providers nor able to change, given the shared nature of resources which can generate inefficiencies to some other applications or services. Therefore, typical HPC routing solutions [34] are not straightforward, or even recommended, to use. At the same time, cloud operators can for example consolidate different workloads in the physical hosts dynamically, or arbitrarily migrate Virtual Machines (VM) for

maintenance [17], and other tenants can interfere with HPC application by executing different services in the shared resources. This further increase communication inefficiencies and makes difficult reduce the extent to which application owners can control performance.

DMCBM tackles the issues of the HPC communications in cloud by proposing an abstraction of the network topology and balancing communications at a higher level of the HPC execution stack. With the abstraction is possible to detect alternative paths for HPC application communications which can be used to reduce the communication's latency, when congestion situations are detected.

The middleware is implemented in the session layer and does not require changes in the user's application. Distributed components monitor the communication paths between application processes and store local information about latency, running hosts and potential alternative paths that can be used. DMCBM also leverage this information to detect congestion according to the latency levels that are acceptable by the user. DMCBM balances communication flows and avoids network hotspots by either; 1) selecting alternative paths whenever a congestion is detected at link process communication level, or 2) live-migrating virtual machines in case of congestion at destination host level.

This chapter describes the DMCBM architecture modules that fulfills the objective of this thesis. A detailed functionality description of each modules on DMCBM middleware, along with their interaction are included. There are 2 modules, the System Characterization Module executed prior the application execution; and the Application Runtime Module, that is executed along user's application. The System Characterization Module consists on the application communication pattern recognition, the network topology characterization and the alternative path table creation. The Application Runtime Module consists in the following tasks: monitoring, detection of congestion, alternative path selection, virtual machine live migration and user mitigation. The modules working together, helps HPC applications improve their performance on cloud environments by reducing MPI communication latencies time.

The rest of chapter is organized as follows: The proposal description is detailed in section 3.2. The modules that are part of DMCBM are explained in section 3.3 and 3.4. Finally, in section 3.5 the software architecture of the proposal is described.

3.2 Dynamic MPI Communication Balance and Management

DMCBM middleware gets a balance of HPC communication flows and avoids network congestion by choosing alternative paths whenever a congestion is detected at link process communication level, or live-migrating virtual machines in case of congestion at destination host level. Also, helps user to detect application mapping and parallel implementation issues. The middleware has two modules in order to improve HPC communications in cloud. Figure 3.1 depicts each part of the DMCBM modules.

The System Characterization module is executed prior the application execution (Offline). It detects communication patterns between application processes, and use the pattern to characterize the network topology based on latency measurements. This information allows the middleware to find alternative paths for the application process communications.

The Application Runtime module is executed along with the application. It monitors application communications in order to detect congestions scenarios, whenever communication latency out range the boundaries accepted by the user. Furthermore, the module implements two complementary controllers to deal with congestion: i) the *Alternative Path Selection Controller* handles congestion scenarios related to link process communications in different VMs, ii) the *Virtual Machine Live Migration Controller* takes care of live-migrating VMs when a destination host is overflowed by communications. The feature of virtual machine live migration is currently activated for private clouds. This module also introduces a *User Mitigation Controller* which helps users detect issues related to application parallelization of task or mapping by logging this kind of event.

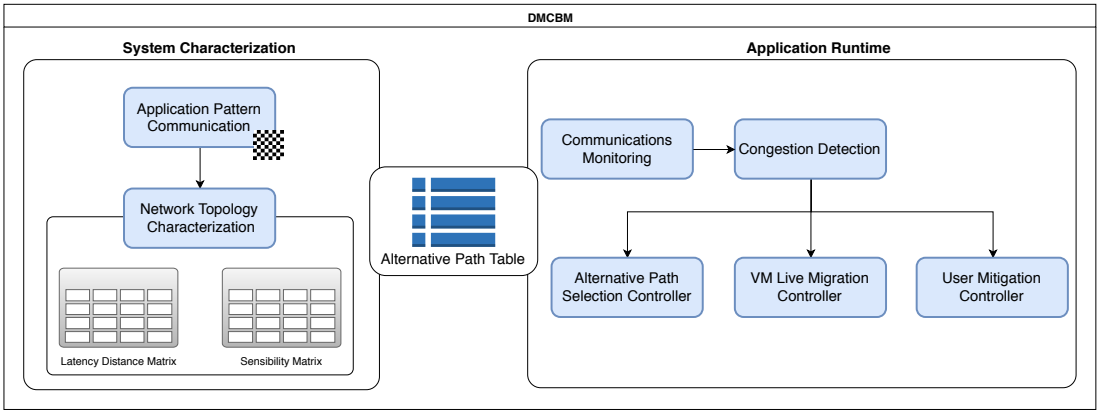


Fig. 3.1 DMCBM: Dynamic MPI Communications Balance and Management in Cloud

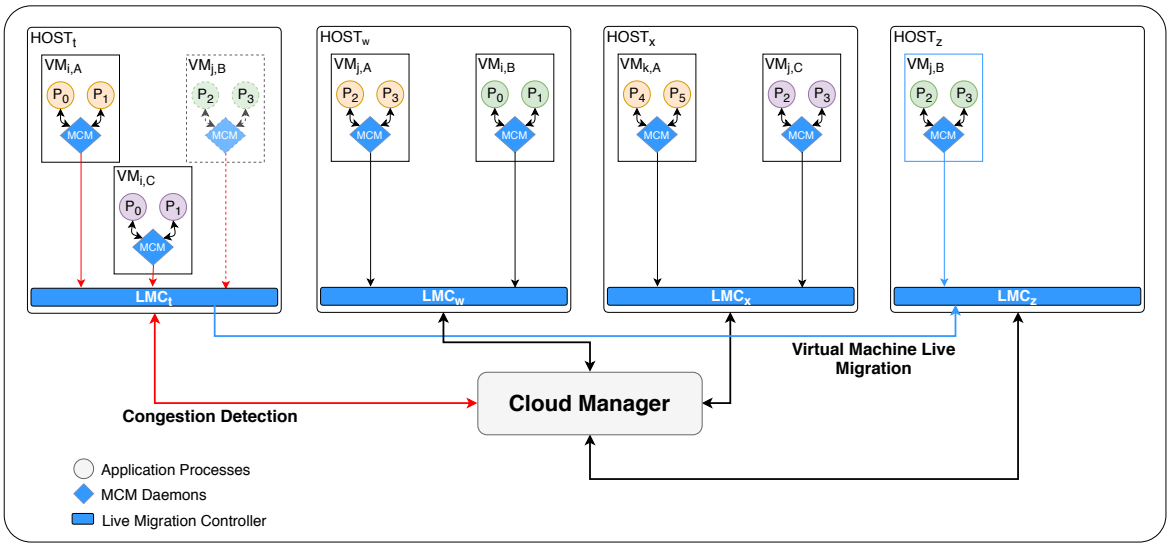


Fig. 3.2 DMCBM: MCM and the Live Migration Controller

DMCBM implements the Application Runtime module task and controllers by software components. The MPI Communication Management (MCM) daemon implements the tasks of monitoring and detection of congestion, along with the *Alternative Path Selection Controller* and the *User Mitigation Controller* functionalities. Besides a Live Migration Controller (LMC) daemon implements the functionality of the *VM Live Migration Controller*.

There is one MCM per virtual machine and one LMC per host. The MCM daemon captures sent and received messages between processes in order monitor message path and compute alternative paths avoiding congested routes. After selecting a better path, the message is forwarded through that path. The LMC is constantly updated by each MCM daemon running on the VMs within the host, with the information about the host congestion status. This procedure enables DMCBM to detect and avoid congestion at destination host, using virtual machines live migration techniques. A scheme of the DMCBM software components distribution is shown in Fig. 3.2. It depicts several virtual clusters in different hosts ($HOST_y$), where y is the host id. The VMs hosted in the hosts are defined as: $VM_{p,q}$ where p is the VM id, and q is the virtual cluster id. The VMs run different application processes (P_n). The Cloud Manager is also depicted, it enables work with virtual appliances and workloads focusing on the end-user's perspective, rather than the IT or systems administrator's perspective. It offers capabilities that simplify the process of executing many common public or private cloud operations such as: provisioning and de-provisioning virtual machines.

The cloud manager helps the middleware to identify the status of each bare-metal host, when a VM live migration is needed.

The next sections describe all MCBM modules and controllers in detail.

3.3 System Characterization

As mentioned above, this module executes three functions. The functions are the ***Application Communication Pattern Recognition***, the ***Network Topology Characterization*** and the ***Alternative Paths Table Creation***. They are executed prior application execution, such as profiling operations. DMCBM uses two matrices to create the alternative path table; the Latency Distance and the Sensibility Matrices. 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.

3.3.1 Application Communication Pattern Recognition

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 [47], 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, DMCBM 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.

3.3.2 Network Topology Characterization

DMCBM characterize the underlying topology using communication patterns, as well as information of the virtual machines location and the performance obtained between virtual machines pairs. Thereby, an analysis of MPI communications between all pairs of virtual machines in cloud is performed. For this analysis the message size of the

application communication pattern with more repetitiveness is used. The network is characterized building two matrixes:

- ***Latency Distances Matrix***: With this matrix a basic detail of distances between VMs is obtained. The main idea is to build the matrix using *Ping Pong* communications, taking in to account the application communication pattern and its message sizes. The *Ping Pong* is performed between processes placed in different virtual machines of the virtual cluster. The *Ping Pong* application runs alone in the VMs of the virtual cluster, without disturbance of any other application. 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. Hence, each intersection of rows and columns in the matrix represents a path between one pair of source and destination VMs, obtaining a basic detail of distances between virtual machines in terms of latency time.

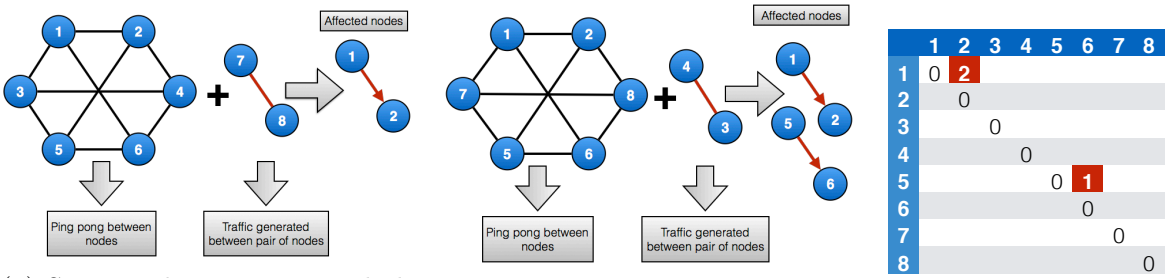
- ***Sensibility Matrix***: Determines how robust to congestion are the forwarding paths. Intersection of rows and columns represents a path between two VMs. The number of times in which a path is affected by congestion between any other pair of VMs is stored as a robustness weight. The main idea is to find relationships of affected VMs when a congestion is injected on other peer of VMs. Algorithm 1 is used to build the ***Sensibility Matrix***.

  The algorithm uses 3 helper functions to build the matrix. The *InjectTraffic* function injects a constant traffic given a pair of VMs. With the traffic injection, it is possible to evaluate how it affects the communications between other the

Algorithm 1 Sensibility Matrix (SM) Construction

Require: VMs = Set of VMs of a virtual cluster
function InjectTraffic(VM_i, VM_j)
    Inject a constant traffic between VM_i, VM_j
end function
function MeasureLatency(VMs)
    Measure latency between VM of VMs using *Ping Pong* application
end function
function CalculateSensibilityMatrix(VMs_a)
    Increment weight in the Sensibility Matrix for the intersection of VMs_a
end function

Require: Initialized SM for all communication intersections of VMs
for each peer VM_i, VM_j of VMs **do**
    InjectTraffic(VM_i, VM_j)
    MeasureLatency($VMs - (VM_i, VM_j)$)
    VMs_a = Get peers of affected VMs
    SM = CalculateSensibilityMatrix(VMs_a)
end for



(a) Steps with congestion in link 7 and 8 to determine related VMs

(b) Steps with congestion in link 4 and 3 to determine related VMs

(c) Sensibility Matrix

Fig. 3.3 Sensibility Matrix Construction

VMs of the virtual cluster, extracting the pair of VMs used in the *InjectTraffic* function. The traffic affects the communication of others VMs when the latency measured, by using the *MeasureLatency* function, beats the base latency measured in the ***Latency Distances Matrix***. When the measurement beats the base latency, the *CalculateSensibilityMatrix* function, update the sensibility matrix in the interception which represents the path between VMs, by adding 1 to the current stored weight in the matrix.

Figure 3.3 shows an example of how the sensibility matrix is populated.. The congestion between VMs 7 and 8 affects the path between VMs 1 and 2, so a weight of 1 is added in the interception of rows which represents that path. Then, congestion between VMs 4 and 3 affects the path between VMs 1 and 2, and the path between VMs 5 and 6. A weight of 1 is added in the interception of rows of VMs 1 and 2, and VMs 5 and 6. The path between VMs 5 and 6 is less affected by congestions, meaning a lower weight in the ***Sensibility Matrix***; this represents more robustness. The process is repeated for each combination of a congested VM pairs to obtain the robustness of every path.

3.3.3 Alternative Paths Table Creation

The alternative path table is created using the ***Latency Distances Matrix*** and the ***Sensibility Matrix***, in order to select at least two alternative paths for a given source-destination pair, employing an intermediate node if required. Then, instead of storing the two matrixes of $N * N$ at each node, only one vector (called alternative paths table from now) of N elements is stored at each node, in order to avoid storage overhead due to the size of matrixes. Each position of the alternative path table

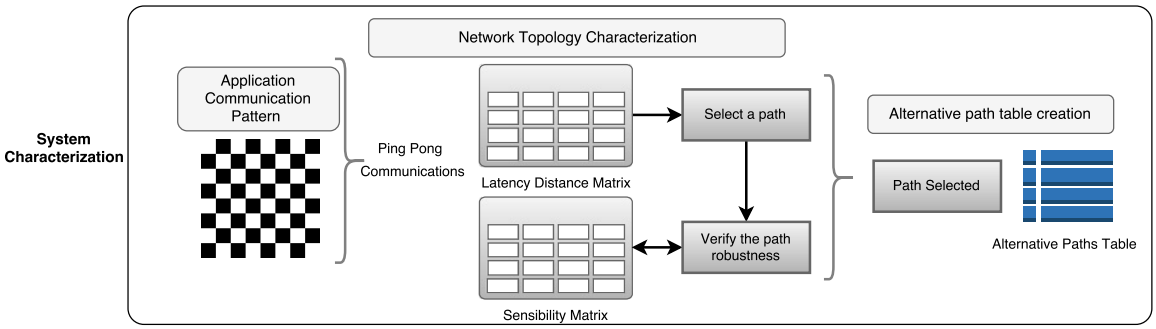


Fig. 3.4 Alternative Paths Table Creation

represents a destination to reach from the VM that contains the table, and the value stored in each position is the base latency of the path.

The alternative paths in the table are selected considering the highest robustness and lowest latency distance from source to destination nodes, including the intermediate node. The path with lowest latency distance is obtained from the ***Latency Distances Matrix***, then its robustness is verified using the ***Sensibility Matrix***. In order to obtain the robustness of a path, we sum up all the weights from source to the intermediate node, and then to the final destination. The whole process is shown in Fig. 3.4.

3.4 Application Runtime Module

This module monitors the communications of the application processes to detect and avoid congestions, besides it identifies issues regarding application mapping and parallel implementation. Upon detection of congestion, two complementary controllers operate to remove it: the Alternative path selection controller and the VM live migration controller. The *Alternative Path Selection Controller* is in charge of avoiding link congestions between the application processes that are in different VMs; and the *Virtual Machine Live Migration Controller* manages congestion at destination due to overflow of incoming messages in the host. However, to handled issues regarding the application mapping and parallel implementation, another controller is needed, the *User Mitigation Controller* helps users to handle this kind of problems, leveraging the methodology of MPI Communication Management (MCM) [9].

3.4.1 Monitoring, Detection and Notification tasks

The Monitoring and Detection tasks are launched with the application's processes, in order to monitor the virtual network status and detect congestion situations. There is a MCM daemon at each virtual machine in the execution environment for intercepting application messages before they reach the MPI library, and for selecting a congestion-free path to forward them. MCM also provides Notification task which is activated if a congestion is detected. This task maintains the alternative path table update with the correspond path latency time, and also initiates the selection of alternatives path if a link congestion occurs or a virtual machine live migration if congestion at destination hosts occurs.

Monitoring: The 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. The Monitoring task includes latency values accumulation and contending flows identification of in-transit messages. These actions are performed by each MCM daemon, the monitoring of latency is performed starting from the message source until it reach to its destination (Figure 3.5).

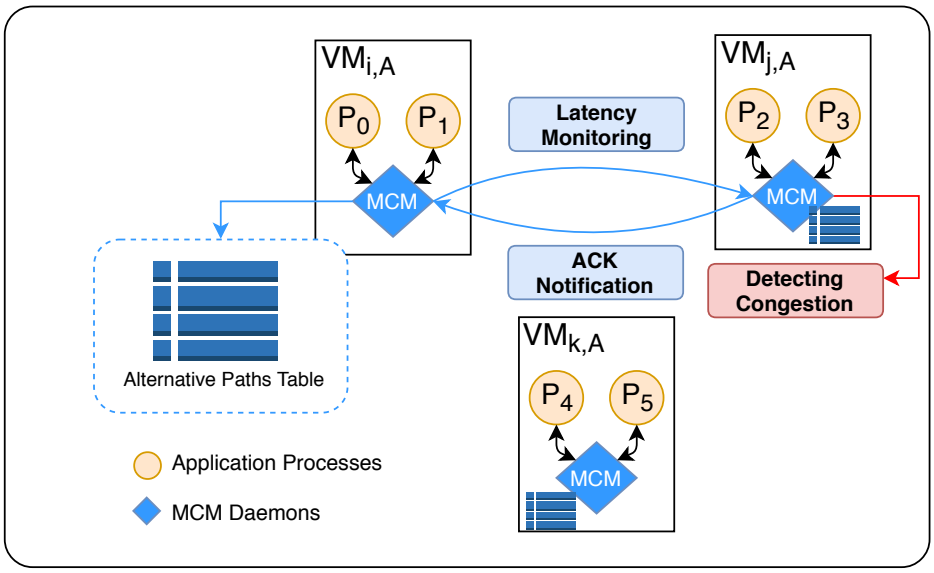


Fig. 3.5 MCM tasks without congestion

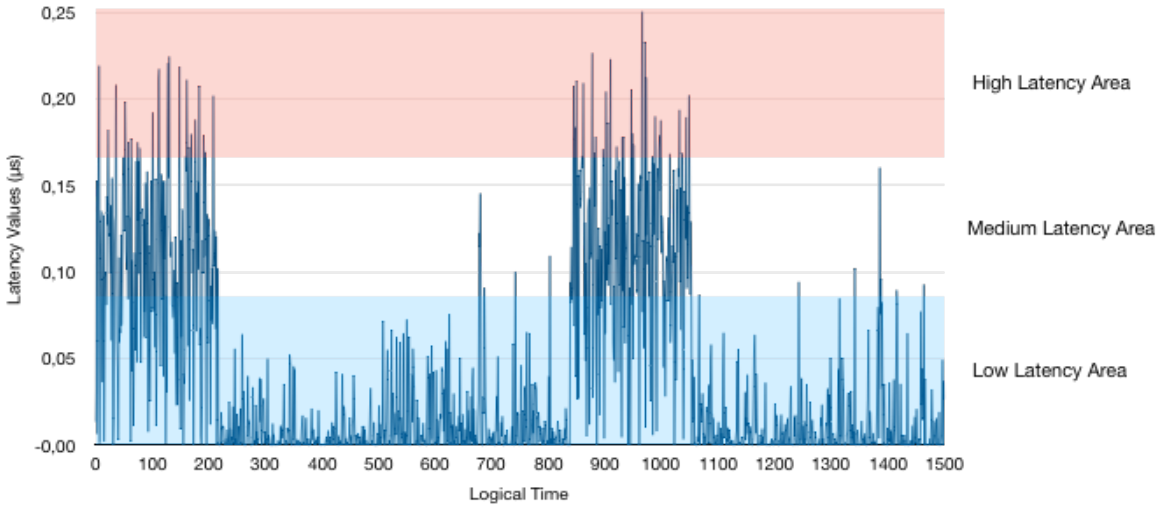


Fig. 3.6 Thresholds verification

Congestion Detection: This process identifies the moment when an action must be done to avoid congestion. It is performed when messages arrives to its destination (Figure 3.5). Three areas are selected, one with low latency, another with medium latency (where congestion is raising but network can handle it), and finally one with high latency (extreme congestion). In low and medium latency areas, we do not search for alternatives paths, but when a transition to medium and high latency occurs, our approach looks for an alternative path. Figure 3.6 shows an example of the defined areas and each transition.

Repetitive communication behavior allow us to compute better routes. When a repetitive pattern appears, we verify if the link latency is in the accepted range defined by a threshold. This threshold is defined as a factor of the base latency. When latency is acceptable, we send messages using the original path. If latency in the current path is not acceptable, an alternative path is chosen.

ACK Notification: It is initiated at destination VMs. An Acknowledge (ACK) message with the path latency is created and sent back to the source when the current message latency out range the acceptance threshold. The path latency is used to update the alternative path table. Hence, maintaining latency path updated, MCM is able to take into account cloud variability that affect to each path (Figure 3.5). This procedure also initiates the alternatives path selection in to the source if a link congestion is detected early or a virtual machine live migration if a congestion at destination occurs.

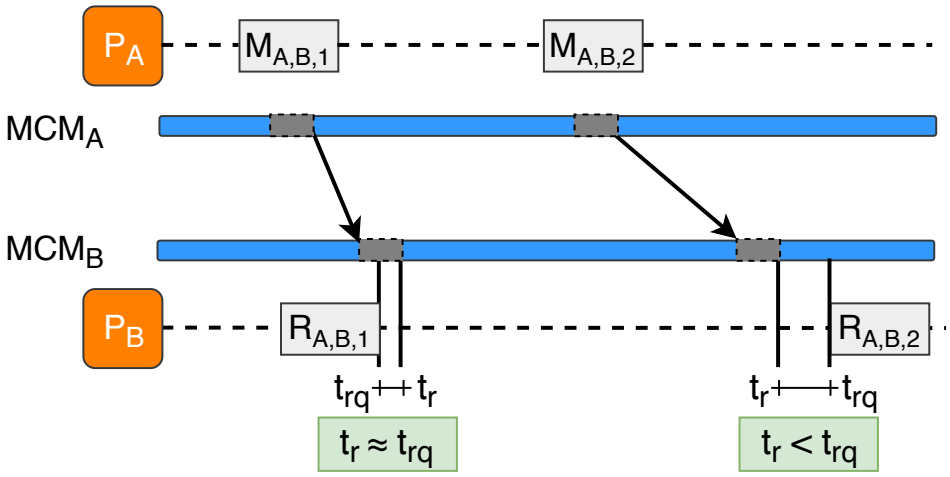


Fig. 3.7 Avoiding ACK notification

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. 3.7, 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 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. 3.8.

The technique can also calculate each message transmission time t_t, using the time measured when a message is sent t_s and the time when an ACK notification arrives t_a

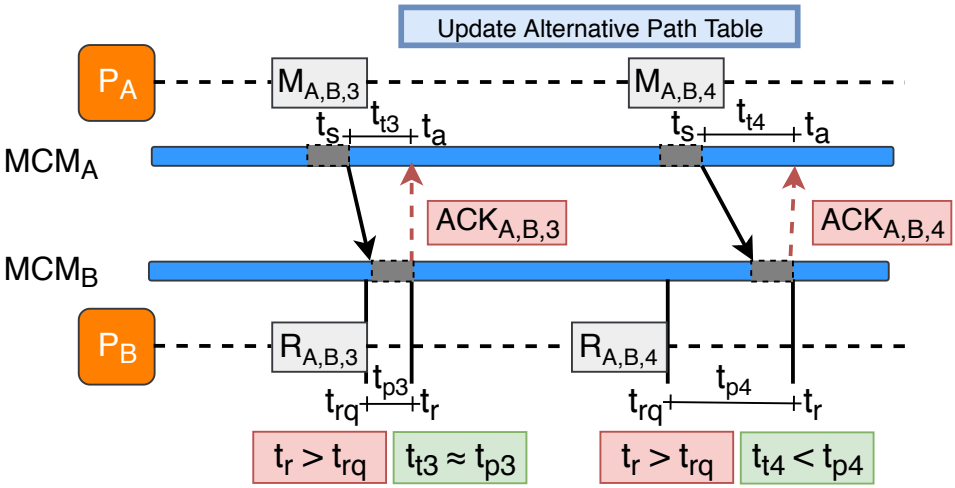


Fig. 3.8 ACK notification

in the source MCM daemon. The time t_t, includes the sending and waiting time for the ACK notification from the destination MCM daemon. The period of time that it takes for the destination MCM daemon 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.

3.4.2 Alternative Path Selection Controller using MCM

The *Alternative Path Selection Controller* leverages recurrent patterns in HPC workloads to select alternative paths using MCM daemons. The proposal identifies communication patterns to improve message forwarding and early avoid link congestion using intermediate VMs. The MCM daemon is executed at each VM to select alternative paths based on the latency values in the alternative path table, Fig. 3.9 illustrates the process. MCM does not add overhead to the MPI send and receive primitives since it operates in a non-blocking way, concurrently to the message forwarding.

Figure 3.10 provides an example of path selection, messages $M_{A,B,1}$ and $M_{A,B,2}$ are sent from process P_A to P_B. P_A and P_B are located in the virtual machines VM_A and VM_B, respectively. Both messages are intercepted by the MCM_A daemon, which selects a congestion-free path to perform the send operation. The monitoring process (see section 3.4.1) measures and updates the latency of the path.

In this example, $M_{A,B,2}$ message go through congestion in the path from VM_A to VM_B, so the configuration of a new path takes place. Hence, later messages between the same source and destination pair (e.g. $M_{A,B,3}$) are sent through this new path which contains an intermediate virtual node VM_C. In MCM daemons, the message

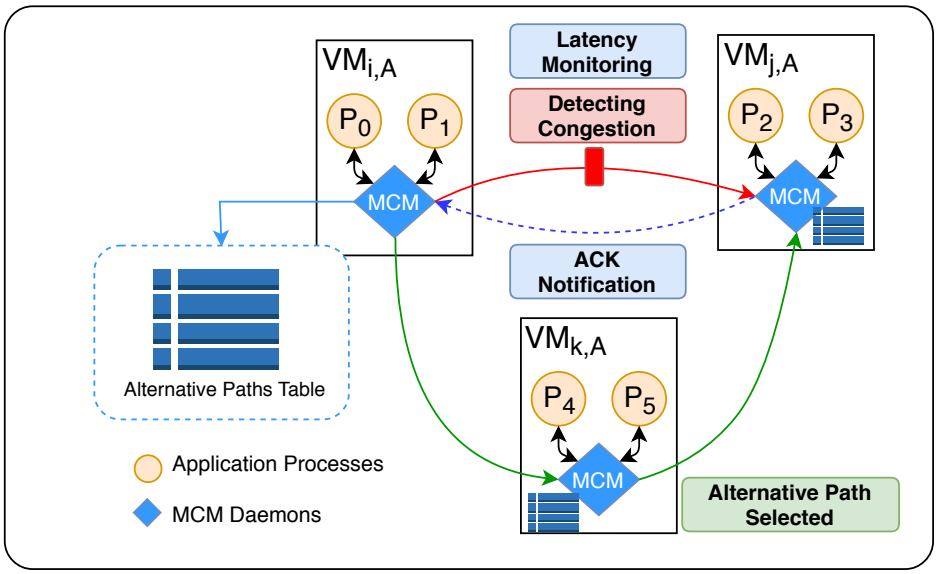


Fig. 3.9 MCM tasks with congestion

reception time is denoted as t_r, and the time when the process request a message is t_{rq}. We describe 3 key scenarios: 1) When t_r is smaller or equals to t_{rq}, we avoid the ACK notification message from MCM_B to MCM_A, because there is no delay on message reception for the destination process; 2) if the reception time (t_r) is bigger than the process request time (t_{rq}), MCM daemon creates and sends a ACK notification message from MCM_B to MCM_A, which notifies a delay in the message reception, and it has to be evaluated in case a congestion is detected; 3) for scenarios where $t_r > t_{rq}$, and a congestion is detected, MCM selects an alternative path, that will be used to send future messages using an intermediate VM (VM_C) with a MCM daemon (MCM_C).

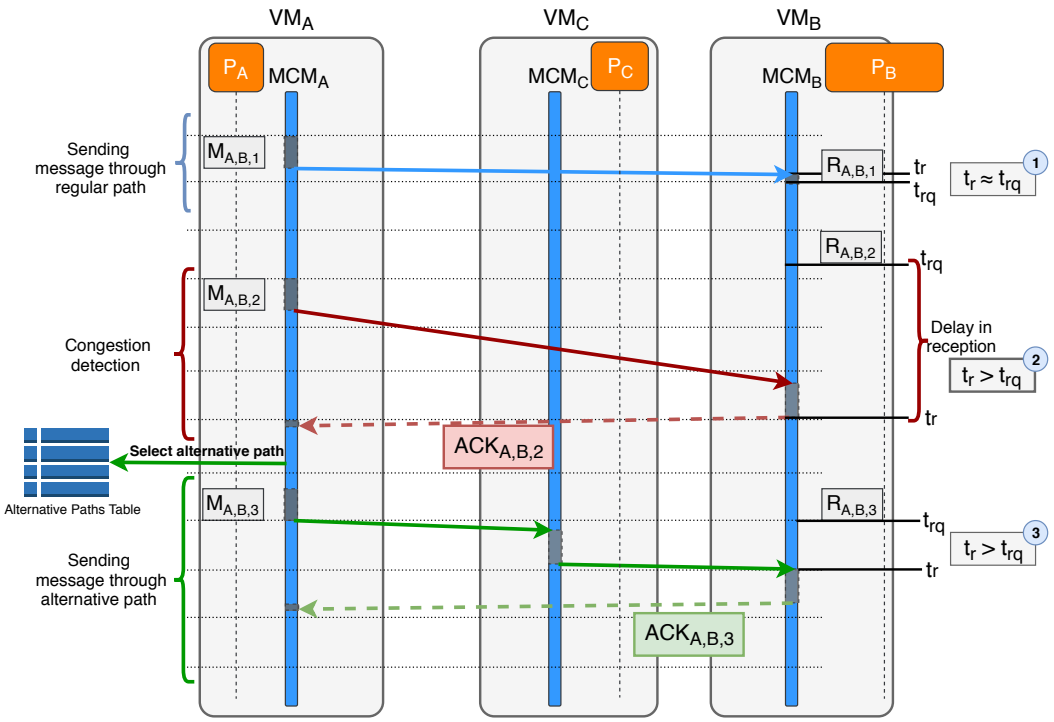


Fig. 3.10 Selection of alternative path example

MCM Functional Model

MCM treats messages differently depending on whether these are either in a source, in-transit or destination VM. At source VMs, forwarding paths are selected, whereas when the message is in-transit only forwarding is performed. At destination VMs, monitoring and detection tasks are performed as well as delivering the message to the proper application process.

In order to be more explicit, each daemon can behave in 3 different modes: source, in-transit or destination mode, depending on the message that is currently being processed. When a process in the application sends a message, its associated daemon proceeds to send the message (source mode). In this mode the daemon selects the path and then performs the forward through the In-Transit daemon. When the destination process requests the message to its associated daemon, the daemon (in destination mode) delivers the corresponding message. The MCM Daemon operation modes are depicted in Fig. 3.11.

Messages routing decision takes place when a message is sent or received in a MPI application process. MCM captures the message, encapsulate it to add information

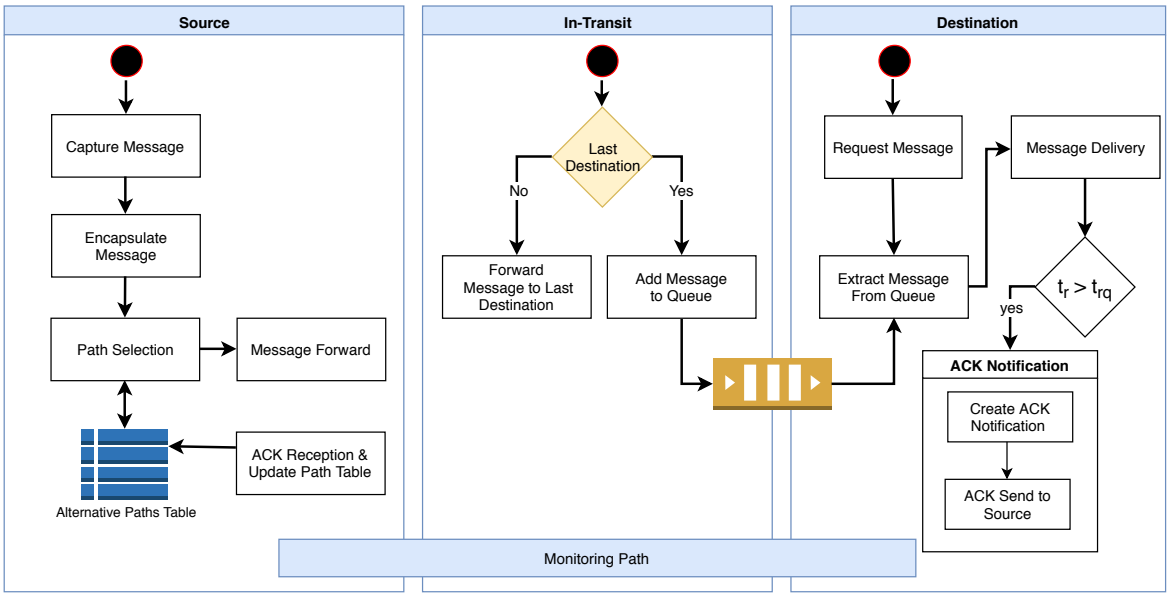


Fig. 3.11 MCM Daemon operation modes

about the message, like its source and destination in order to alter the normal course of its communication without losing relevant message information.

When a *send* primitive is caught, a path selection is done, using the alternative paths table. After the best route is found, messages are forwarded through this path. The message latency is measured including all the hops in path carrying out a ***Monitorization***. If an alternative path is used to send messages, an In-Transit mode of the MCM daemon is activated. For each arriving message, MCM checks whether the destination process of in-transit messages is in the current VM. If so, the daemon delivers the message to the receiver queue; otherwise, it forwards the message to the destination VM. Last, when the MCM daemon in destination mode caught a MPI *receive* primitives in the destination VM, it delivers the corresponding message to the application process from the receiver queue.

In the destination mode the reception time in the MCM daemon (t_r) is verified. If t_r is greater than the process request time (t_{rq}), an ACK notification message is sent back to the source process, as explained before. Another operation performed during the execution is the update of the ***Alternative Path table*** in the source. The update is performed using the ACK information received from the destination, maintaining the latency time of each path with its current state.

To avoid causing interference with the order in which the messages are delivered, we have designed a sorting algorithm (Figure 3.12) that provides the messages of the

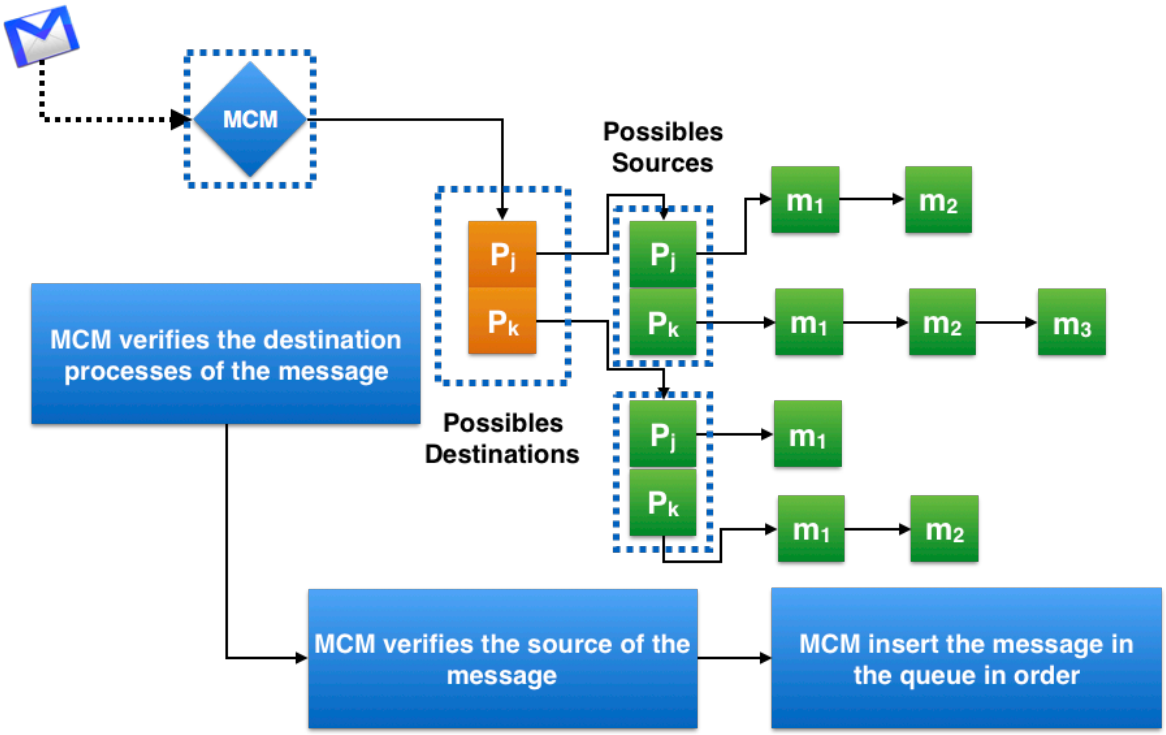


Fig. 3.12 Message order process

application in their original order from the source to their corresponding destination. So that the application ends correctly, avoiding the loss of messages and fulfilling the pre-established order.

3.4.3 Virtual Machine Live Migration Controller

The *Virtual Machine Live Migration Controller* is in charge of VMs migration from a host to another, when a congestion at destination host is detected. This operation is done by the Live Migration Controller (LMC) daemon. LMC takes over MCM to mitigate congestion at destination hosts, which is a type of congestion that cannot be addressed by selecting alternative paths. Congestion at destination means that a VM is overloaded and cannot receive messages. Usually, in traditional networks this type of congestion is solved with overload control techniques that discard messages. In HPC systems, messages are not discarded, but overload is controlled using traffic throttling techniques at source nodes. Cloud, however, offers the possibility to migrate instances to alleviate the congestion. Live migration is transparent to users, meaning that the application processes continue executing during the migration.

LMC is composed of three process to avoid congestion at destination host: Virtual Clusters Manager, Monitor VMs and VMs Live Migration. The Virtual Cluster Manager identifies and registers the location of the VMs of each virtual cluster. The Monitor VMs process, maintains the hosts status information to detect congestion at destination. This information is obtained from each MCM daemon allocated in VMs of each host, and it is constantly updated with the communication latency of incoming messages. When a VMs live migration is required, the VMs Live Migration process is triggered. There is one LMC for each host of the cloud environment, in the Fig 3.13 the processes of the LMC are depicted.

During the Monitor VMs process, each MCM Daemon has to inform the status of incoming messages latency on its respective VM. Once, all VMs information from a host is obtained, the status of the host is updated in LMC. When a high latency time is detected among all the virtual machines of a specific host, a congestion at destination situation is detected in LMC, and a VM live migration process is triggered.

The VMs Live Migration process verifies the status of each host to select the best candidate for migrating VMs. Then, it requests the cloud manager (e.g. Openstack)

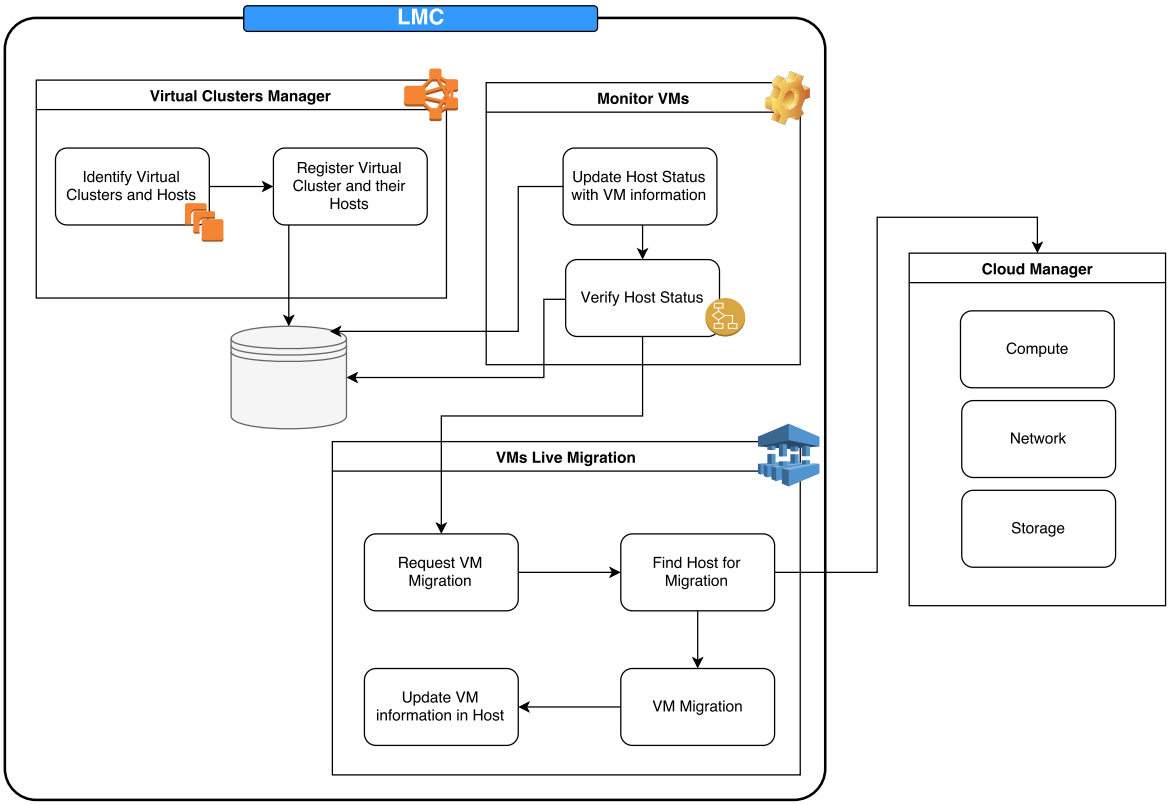


Fig. 3.13 Live Migration Controller

to migrate the VM to the selected suitable host. The cloud manager verifies that the selected host has enough resources to support the migration in terms of CPUs, memory and disk space. Usually, the cloud manager provides and API (e.g. Openstack nova) that responds the LMC request for migration either accepting it, or rejecting it if there are not enough resources available. In the latter case, LMC finds another target host.

Once the VM live migration process is successfully performed to the destination host, the VM location information has to be updated in LMC. The MCM Daemon of the migrated VM also has to update its location information in the LMC database, in order to continue verifying the status of the VM and with it maintaining the host status in LMC.

3.4.4 User Mitigation Controller

DMCBM 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 DMCBM, 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.

In this proccedure the technique uses each message transmission time t_t, using t_s and t_a. Also uses the period of time that it takes for the destination MCM daemon to deliver the message to the application destination process measured using $t_r and t_{rq}$, the processing time t_p. Both measures are shown in the Fig. 3.14.

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, hence a congestion is detected for the DMCBM middleware.

The User mitigation process is only activated when $t_r > t_{rq}$. Figure 3.14 shows an example of 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, DMCBM writes a log to inform the user's application the finding issues.

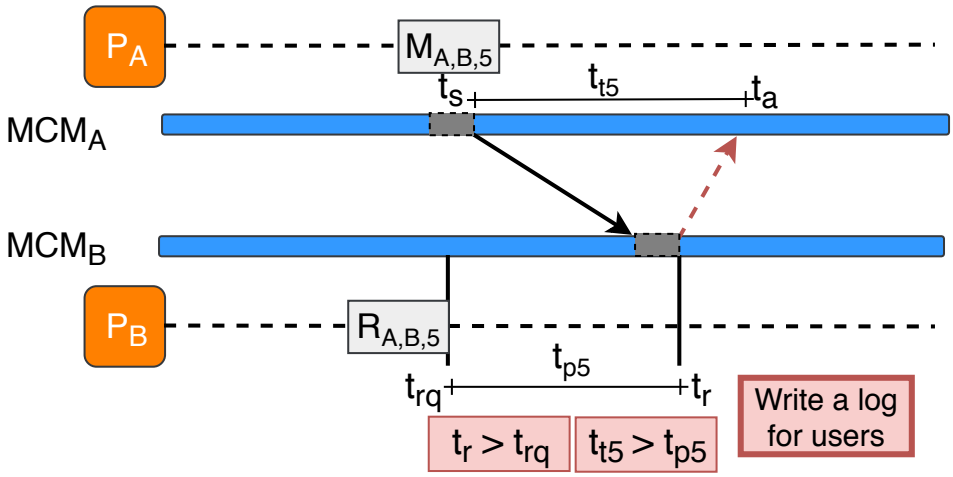


Fig. 3.14 DMCBM Mitigation processes

3.5 DMCBM Software Architecture

DMCBM architecture is composed by two main software elements: the MCM daemons and Live Migration Controllers (LMC). These elements are executed together with user processes. MCM daemons work between user processes and the MPI library. This daemon is in charge of the path selection for messages from user applications. It also implements functionalities to capture messages using a dynamic library, which intercepts messages before they reach to the MPI library. Meanwhile, LMC detects the congestion at destination host and triggers the virtual machine live migration when is needed. DMCBM software architecture for one host and its virtual machines is shown in Fig. 3.15.

For each VM in which the application processes are executed, there is a MCM daemon. This daemon is launched by the MPI library as a process like the application processes. A MCM daemon running on a VM, manages all MPI communications from processes that runs on the VM. The MPI communication management is transparent for the user application and the MPI library.

The architecture uses two different communicators to send and receive messages. Every communication made in the user application is done using an Application Communicator, which is created in a transparent way to the application. All MCM daemon's communications are performed using an specific communicator, Daemon Communicator. In this way communications conflicts between MCM daemon and application's processes are avoided. The user only needs to launch its application

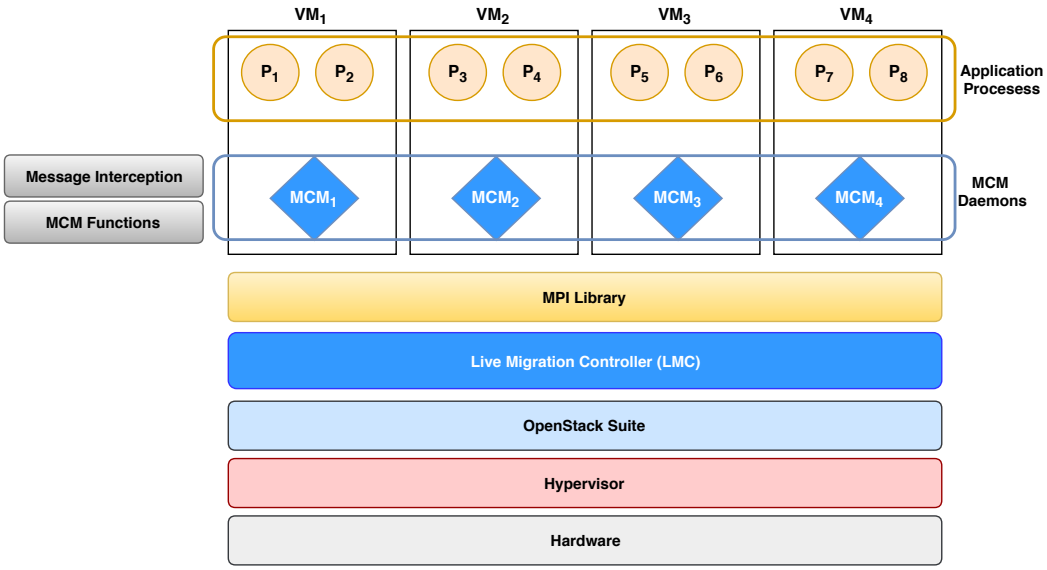


Fig. 3.15 DMCBM Software Architecture

together with MCM. A script is provided for that. User application re-compilation is not needed.

One Live Migration Controller (LMC) is launched on each real host of the cloud environment. The LMC manages the real host status by receiving the latency updates from MCM daemons, which run on each virtual node. It also manages the interactions with the cloud manager (e.g.: OpenStack Suite) to perform virtual machines live migration.

3.6 Summary

DMCBM method characterizes the underlying network topology, and looks for alternative paths to avoid congestion situations by distributing the messages communications between links with less usage. It detects links and host destination congestions, and provides mechanisms to avoid them, using alternative paths and virtual machine live migration. The method allows to obtain better execution time in case of congestion for the applications in cloud, optimizing the resources usage.

The method is based on the network topology information recognition. It uses this information, which is stored as two matrices, to obtain the alternative path table. The information of the network topology is obtained during the system characterization, which is done prior the application execution. This characterization comes with a cost, which is $n(n-1)$, where n is the number of virtual nodes that the virtual cluster use.

This process could be avoided, by creating and updating this information during the application execution, however, then application communications and characterization information would be mutually affected.

Applications may show lasting and adverse communication patterns, which are monitored and tackled by the DMCBM, due to its effects in the communications of the application, e.g.: increasing the latency in communications. Meanwhile, applications can present communication patterns that are not lasting. This kind of pattern are no treated by the technique because they not affect the application communication and execution in a several way.

Public cloud providers that offers Infrastructure as a Service (IaaS), enables DMCBM to take advantage of the VM live migration facility. This facility allows the technique to avoid congestion at destination. Although, other cloud provider offers, such as Platform as a Service (PaaS) may not provide the live migration facility. For private clouds, live migration facility is always possible.

Chapter 4

Evaluation

"Science, my lad, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth."
– Jules Verne, *A Journey to the Center of the Earth*

In this chapter, we present the evaluation of the Dynamic MPI Communication Balance and Management middleware. The evaluation aims to verify the functionality of the proposed solution, through the validation of each module that is provided for the middleware. The validations are performed using several scenarios executed in public and private cloud environments.

The evaluation objective is to validate the benefits of applying DMCBM to HPC applications executed in cloud environments during congestion situations. The middleware has two main modules: System Characterization and Application Runtime, both were described in Chapter 3. During the evaluation, the modules validation are covered. First, the technique that build the network topology abstraction is validated, demonstrating the existence of possible alternative paths for communications of scientific parallel applications processes executed in cloud. Then a validation of the MPI Communication Management method is also presented, during the application runtime.

The experimental scenarios design is done for the DMCBM evaluation. The applications used and their corresponding workload are also defined. The metrics to assess the benefits of our proposal are further explained.

The evaluation metrics cover functional, performance and scalability analysis of our proposal. The metrics have been chosen aiming to measure several aspects related to HPC applications execution in cloud. The main metrics are: the latency time between processes communications, the application total communication time, and; the application execution time.

4.1 Experimental Design

In this section, the experimental design to validate the Dynamic MPI Communication Balance and Management middleware is described. Several scenarios are defined to perform an extensive evaluation of the presented technique. The analysis of each functionality for the System Characterization and Runtime module is done in section 4.7.

The System Characterization module validation experimental results are shown in section 4.7.1. Then, the experiments of Runtime module validation are presented in the following sections: 4.7.2, 4.7.3, 4.8 and 4.9. The components that are part of the runtime module, are incrementally evaluated. Hence, a controlled environment is obtained for each stage of the module and with this, we are able to identify the effects of each added component for the application execution. Each evaluation of DMCBM is done by stages, activating components that are referenced in Table 4.1.

DMCBM with the aggregated components, are evaluated in terms of performance, scalability and communication latencies on a public and private cloud in sections: 4.8 and 4.9, respectively. The DMCBM middleware implementation for public cloud is called DA-MCM. Meanwhile, the middleware implementation for the private one is named MCBM. Furthermore, a comparison of our solutions with currently available state of the art similar solutions is performed using a quantitative and qualitative approach in section 4.10.

Table 4.1 Experimental scenarios

Components	DMCBM Stages		
	MCM	DA-MCM	MCBM
Monitoring	yes	yes	yes
Detection	yes	yes	yes
Alternative Path Selection Controller	yes	yes	yes
VM Live Migration Controller	no	no	yes
User Mitigation Controller	no	yes	yes
Section	Section 4.7.3	Section 4.8	Section 4.9

For the runtime module evaluation, the applications are tested in scenarios with and without congestion. This is done, to evaluate the effects of applying our solution to the application execution, showing the benefits of the proposed technique. Namely, the scenarios are described as follow:

- Application without congestion (NC): The application is executed in the cloud environment without any congestion created and without the solution approach applied. This is the base case.
- Application with no congestion and with the proposed method included (NC-DMCBM stage): The application is executed with the proposed method included, and no congestion is created in the VMs of the cloud. This scenario mesure the method overhead when no congestion is present.
- Application including congestion and without the proposed method included (IC): In this scenario the application is executed without applying the method proposed in this thesis. In this case a congestion is generated by a synthetic parallel application with communication intensive behavior between VMs of the cloud. This shows application degradation due to an adverse traffic.
- Application including congestion and applying the proposed methos (IC-DMCBM stage): The application is executed in virtual nodes of the cloud, applying the technique presented. At the same time a congestion is generated between VMs of the cloud, by a synthetic parallel application with a random communication intensive behavior. This shows the improvements made by the solution proposed in relation to the situation with congestion.

4.2 Evaluation Metrics

The metrics evaluated on each experiment of this thesis are described in this section. This catalogue of metrics can be used as a dictionary entry of each performed test. We select the metrics from a catalogue of metrics for evaluating commercial cloud services [28]. The selected metrics are the followings:

- Performance Evaluation Metrics: In this kind of evaluation we observe the total execution time of the application execution with and without the proposed solution applied.

- Communication Evaluation Metrics: Communication refers to the message latency time of each message transfer during the MPI application execution.
- Correlation between Total Runtime and Communication Time: this evaluation observe a set of applications about their runtime and the amount of time they spend communicating in the Cloud. This metric is to observe a set of applications about their runtime and the amount of time they spend communicating in the Cloud.
- Scalability Evaluation Metric: the proposed solution strong-scalability is evaluated in some experiments during the analysis phase. It requires the variation of workload and cloud resources.

4.3 Evaluation Methods

In this thesis an statistical evaluation is done on each analysis performed, by executing multiple times the different scenarios. The evaluations are done with different kind of applications and workloads. We run each evaluation between 2 to 10 times, in order to have statistical results.

The experiments are executed several times due to the cloud variability, in order to find an statistical behavior of the maximum, minimum and average of the each execution time. Following this methodology to perform the experiments, it is possible to avoid inconsistent results, acquiring a degree of confidence.

In this chapter, the evaluations were developed according to the statistic aspects described in this section. All the evaluations results are further explained in the following sections.

4.4 Applications and workloads

The evaluation of solution proposed is performed using benchmark and real HPC applications with a repetitive communication pattern. Each scientific parallel application used was implemented using the MPI library. The applications used are the followings:

NAS Parallel Benchmark (NPB): It is probably the benchmarking suite most used in the evaluation of languages, libraries and middleware for HPC. One of the most communication intensive codes of the suite have been evaluated, CG (Conjugate

Gradient). This kernel tests irregular long-distance communication and employs sparse matrix-vector multiplication. The kernel is evaluated with different workloads:

- The NAS-CG Class B: The application was tested with 75000 rows, 1000 iterations, 60 Eigenvalue shift and 13 non-zeros.
- The NAS-CG Class C: It was tested with 150000 rows, 1000 iterations, 110 Eigenvalue shift and 15 non-zeros.
- The NAS-CG Class D: The implementation of the kernel in Fortran have been compiled with 1500000 rows, 100 iterations, 500 Eigenvalue shift and 21 non-zeros using the GNU compiler.

The NAS-CG is selected due to its repetitive pattern of communications between processes. The different classes tested are useful because they have many communications between nodes.

N-Body Simulation: This is a real-world HPC application that performs simulation of a dynamical system of particles, usually under the influence of physical forces. The application was implemented in C as a circular pipeline with a communication pattern. It was compiled with the GNU compiler and tested with 100000 particles and 50 iterations.

4.5 Experimental Environment

To analyze the functionality of the approach, a series of experiments evaluating latency of MPI communications and the total execution time of applications running on cloud are conducted. The experiments where performed in a public and private cloud. Amazon EC2 instances for the public cloud and the instances mounted with OpenStack in the private one.

The DMCBM middleware main objective is to improve communication latency time for scientific parallel applications that are moving to cloud environments, and indeed this improving their execution time, making the solution cost effective. It tries to cover its functionality using general purpose instances and also HPC instances. The HPC instances are for user applications that requires high network performance, fast storage, large amounts of memory, very high compute capabilities, or all of these included.

4.5.1 Public cloud Configuration

The main components and configurations of the system used for the public cloud to perform the experiments are described in the Table 4.2. The experiments where performed in instances provided by Amazon EC2 [2]. The solution presented was tested in different kind of instances, instances for general and HPC base purpose, in order to demonstrate that it is useful in both types. Two of the selected instances are the t2.micro and t2.medium. They represent the set of low cost instances for general purpose. The c3.2xlarge represents instances for HPC purpose.

Table 4.2 Experimental setup for public cloud

Component	Cluster Amazon EC2		
Instance	t2.micro	t2.medium	c3.2xlarge
Memory	1 GiB	4 GiB	15 GiB
Storage(GB)	EBS	EBS	EBS
Network	Low to Moderate	Low to Moderate	Low to Moderate
Physical Processor	Intel Xeon 2.5GHz	Intel Xeon 2.5GHz	Intel Xeon 2.5GHz
Clock Speed (GHz)	2.5 GHz	2.5 GHz	2.5 GHz
vCPU	1 vCPU	4 vCPU	8 vCPU

Figure 4.1 shows an example of how MPI applications interacts with MCM daemons of DMCBM in a cluster environment in Amazon EC2.

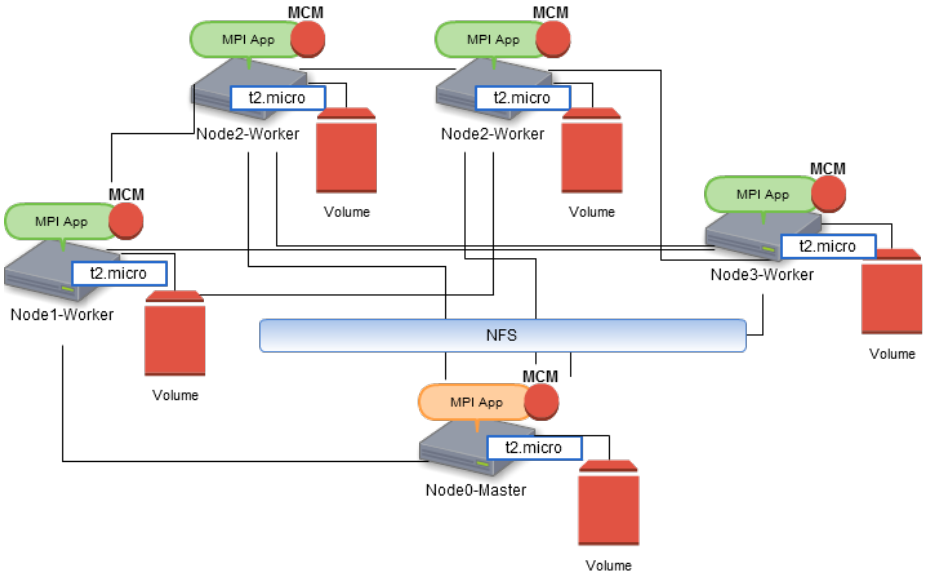


Fig. 4.1 Cluster Diagram in Amazon EC2

4.5.2 Private cloud Configuration

The private cloud used to perform the experiments is mounted using OpenStack, on a real cluster composed by 7 hosts with the characteristics shown in Table 4.3. The experimental setup is composed by one host that acts as a controller and compute node, and the others 6 remaining hosts are for computing. The controller node is managed by a cloud controller. The Kernel-based Virtual Machine (KVM) based cloud cluster is installed with Quemu-kvm-2.6.0 as the hypervisor.

Table 4.3 Real Hosts configuration.

Component	Details
CPU	Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
Cores	48 (w/ HyperThreading)
HD	1080 GB
RAM	125 GB
Network Interface	Up to 1000 Mbps (full duplex)

The experiments are done on 16 and 32 m1.medium instances launched using OpenStack software suite. Each m1.medium instance has 4 GiB of memory, 40 GB of disk space, a Intel Xeon physical processor of 2.5GHz, 2 vCPU and a network of 10 Gbps.

4.6 General Software Configuration

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 [41]. This is an open source cluster-computing toolkit for Amazon EC2.

In public cloud each instance is a Hardware-assisted virtual machine (HVM). This virtualization type provides the ability to run an operating system directly on top of a virtual machine without any modification, as it was running on the bare-metal hardware. All instances are acquired from US East (N. Virginia) data center of Amazon and their operative system is Ubuntu 10.11.

For private cloud the virtual cluster is built with OpenStack Newton. 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.

The measurement of point to point MPI communications latencies and bandwidth are obtained using OpenMPI, which is a High Performance Message Passing Library [12]. We measure latency and bandwidth as two key network performances.

Actually the runtime environment of the proposed solution does not need other language dependencies in order to provide a high throughput computing. DMCBM is implemented in C/C++ and the communications are done based on calls of the standard MPI routines. MPI applications are executed using the Open MPI 1.10 library. All the procedures implemented in DMCBM are hidden for the end-user and are transparent for their MPI applications.

4.7 Analysis of functionalities

In this section we provide experimental results to perform a concepts validation for System Characterization, including Network Topology Characterization and Alternative Path Configuration; and also validation for the Application Runtime module, in which the main elements are the MCM daemons. Both modules are part of DMCBM method. We verify the existence of alternatives path for communications between processes

of the NAS-CG parallel benchmark, using simulation. Finally we evaluate the real execution time of NAS-CG and NBody applications, without and with congestion scenarios, applying or not our MCM method.

4.7.1 System Characterization Validation

A primary procedure for DMCBM is to characterize network topology with the ***Matrix of Latency Distances*** and the ***Sensibility Matrix***. The ***Matrix of Latency Distances*** is built obtaining the base communication latency time between every pair of VMs in the cloud, without any other application running. The ***Sensibility Matrix*** is built to obtain the robustness of all the paths.

The DMCBM middleware needs to find alternative paths, in order to use them if a congestion is detected. In this experiment, we run a *Ping Pong* parallel application to measure latency between nodes, and create the ***Matrix of Latency Distances*** filling it with the base latency without any running application. This process is performed before running the users application in the cloud due to the variability of this environment.

The procedure to create the ***Matrix of Latency Distances*** is described as follows: For N virtual machines, we need N iterations of *Ping Pong* latency evaluation in order to get all pair-to-pair performance. In each iteration, $\frac{N}{2}$ pairs are measured with MPI Send in both directions. When the number of instances is 20, the total evaluation overhead is usually smaller than 30 seconds.

In order to catch system variability, we run several experiments on different days and message sizes. Initially, we evaluate the latency with a message of 1 MegaByte (MB). The result of the latency between nodes is showed in Fig. 4.2, in this graphic we can see that sending a message of 1 MB from node 3 to node 8 has a latency of more than 2500 microseconds, but this is not the only way to perform this communication because in this type of cluster we find that all nodes are interconnected. MCM proposes the selection of path with low latency, i.e.: instead of sending directly to the node 8, we can select the node 5 to send the message through it to node 8, and the sum of latency from node 3 to node 10 and then to node 8 will be approximate 50% lowest than through the first path. Due to space limitations, not all the communication between nodes are exposed.

Second we run the same experiment every 10 minutes for 1 hour and the results of one of it are in Fig 4.3. In this figure we can detect communications that can have lowest latency like the previous experiment, but also the variability of the network in cloud. We can observe with this two figures that according to an specific period of

Fig. 4.2 Latency of Communication Message Size = 1MB in day 1

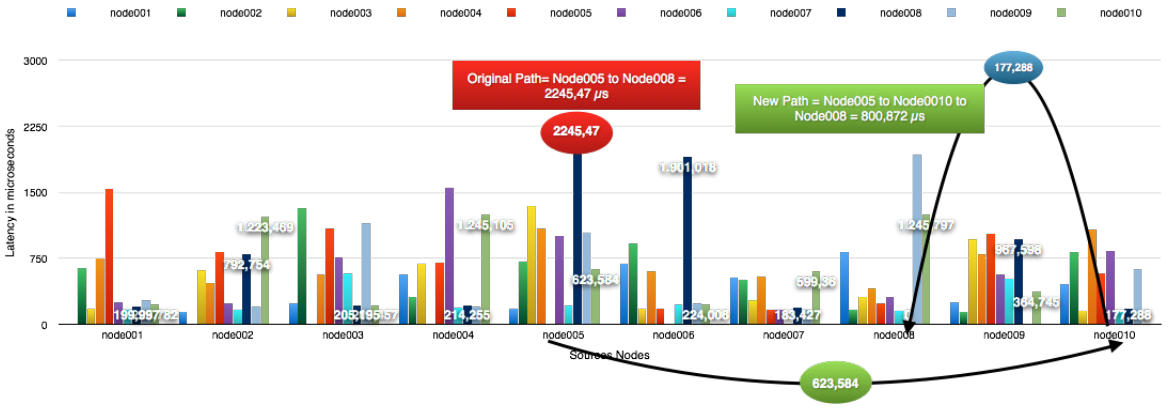


Fig. 4.3 Latency of Communication Message Size = 1MB day 2

time, the latency time of messages can be balanced or unbalanced, depending on the link usage, but it is still possible to find alternative paths for messages.

Also we measure the latency with messages of 64KB to prove the behavior with smaller messages. The results are showed in the Fig. 4.4. In this scenario, latency is maintained between 125 and 225 microseconds. For this reason, our technique have to avoid the selection of new paths, because it would add more latency time.

Network characterization and alternative path creation

An evaluation of the latency using *Ping Pong* with 1 MegaByte (MB) messages and 20 instances t2.micro of Amazon EC2 is also done. This experiment is to validate the whole process of the alternative path creation.

The results of latency between nodes are shown in Figure 4.5. It is possible to notice that sending a message of 1 MB from node 5 to node 8 has a latency of 2.245 microseconds. As MCM proposes the selection of two paths with lower latencies, and in this type of cluster all nodes are interconnected, we select the node 7 and the node

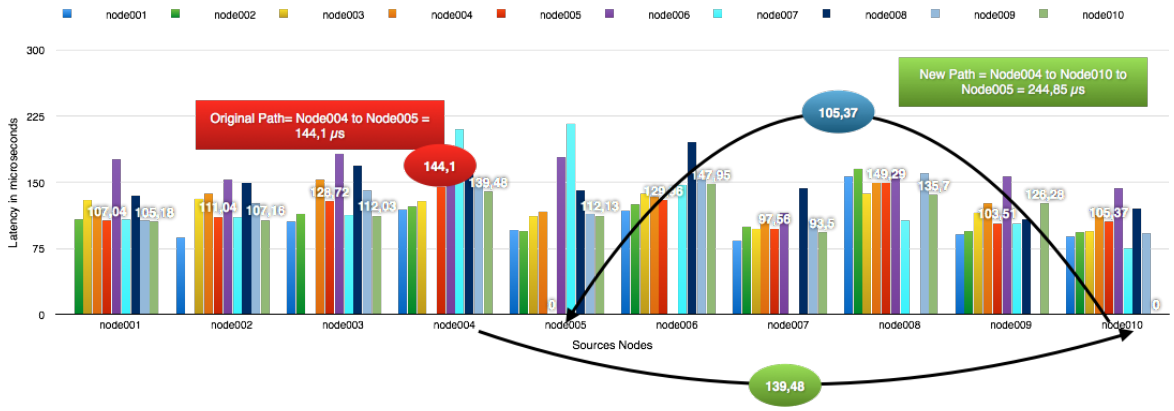


Fig. 4.4 Latency of Communication Message Size = 64KB

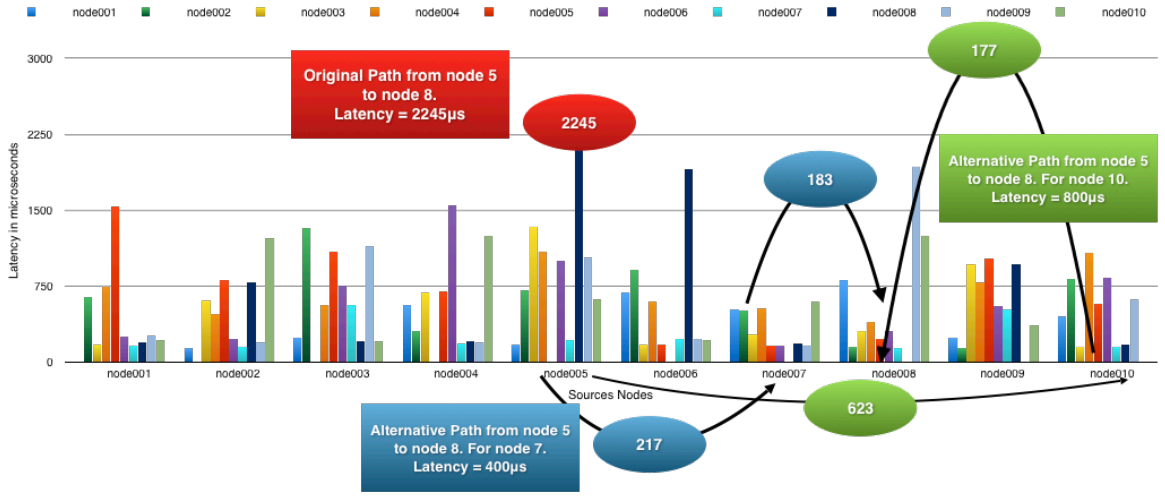


Fig. 4.5 Alternatives Path Selection

10 to send the message through them to node 8. The sum of latency from node 5 to node 7 and then to node 8 will be approximate 5,63 times faster than through the original path and the sum of latency from node 5 to node 10 and then to node 8 will be approximate 2,8 times faster than through the original path.

To verify the robustness of the alternative paths with lowest latency we create the ***Sensibility Matrix***. It allows us to find out an approximation about how the nodes are related to each other. This information is useful for the selection of alternative paths. In MCM, if we find a path with congestion by which we should send a message, we try to select a new path with the least possible relationship to the congested path, and to find it we use the ***Sensibility Matrix***.

Nodes	1	2	3	4	5	6	7	8	9	10
1	0	80	37	12	16	10	23	11	14	20
2	348	0	41	27	43	14	35	37	44	34
3	191	551	0	61	69	29	53	45	62	44
4	100	290	224	0	54	23	47	36	43	39
5	46	60	33	6	0	4	10	9	11	9
6	47	64	33	9	13	0	12	14	11	10
7	43	60	28	5	15	5	0	7	11	10
8	40	59	22	7	13	5	9	0	12	6
9	31	58	21	7	13	3	10	6	0	7
10	37	61	16	5	15	3	9	12	12	0

Fig. 4.6 Sensibility Matrix with path selections

To create the ***Sensibility Matrix*** with weigths of relationship between nodes, we execute the *Ping Pong* message application between processes using $N-2$ nodes, where N is the total number of nodes, also we add a constant traffic in the remaining two nodes. If this congestion affects messages communication between nodes of the *Ping Pong*, this effect denotes a relationship with the congested nodes, and it is stored at the intersection of affected nodes in the ***Sensibility Matrix***.

In Figure 4.6, it is possible to observe the ***Sensibility Matrix*** obtained. For example if a congestion occurs in path from node 5 to 8, we use the ***Matrix of Latency Distances*** to verify which nodes can be used as intermediate node for the alternative path. In the last example the selected nodes were node 7 and node 10. We verify with this ***Sensibility Matrix*** if those alternative paths are robust.

We corroborate the robustness of a path starting from node 5 to 7 and then node 8 (marked in blue on the matrix), and the path starting from node 5 to 10 and then node 8 (marked in green on the matrix). Its possible to observee the robustness of those paths, checking the sum of weights from source to destination, is less than the other possible paths.

The main idea is to avoid the congestion and its related links, so we try to improve the usage of non-congested links. Furthermore, in congestion scenarios we make use of the found alternative paths.

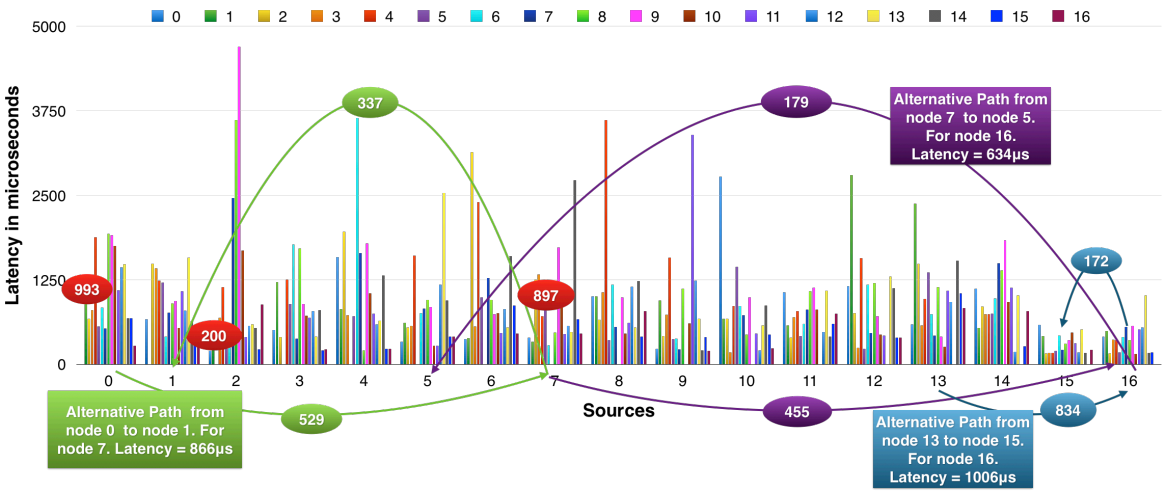


Fig. 4.7 Alternative Paths for NAS-CG Communications

4.7.2 MCM Simulation Evaluation Using NAS-CG Benchmark

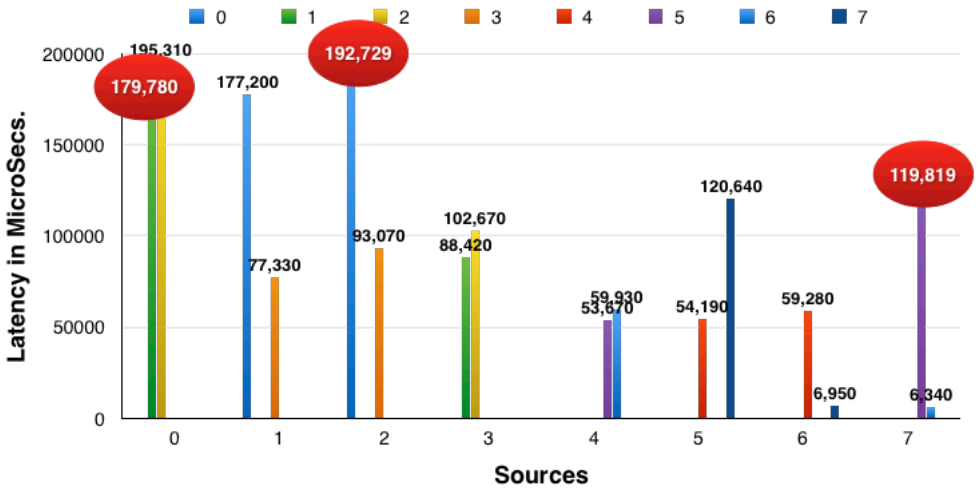
An experiment to simulate MCM daemons functionality is performed, executing NAS-CG parallel benchmark Class B with t2.micro instances (Amazon EC2) in 16 nodes, and the *Ping Pong* message calibration between every pair of virtual machine, assuming that there are not problems on the network, using sixteen process. The results are shown in Fig. 4.7 and Fig. 4.8.

- For Message Communication in the NAS-CG Class B, between node 0 to node 1, a total latency of 179,782 microseconds for the message size predefined for the application is measured (Fig. 4.8a).

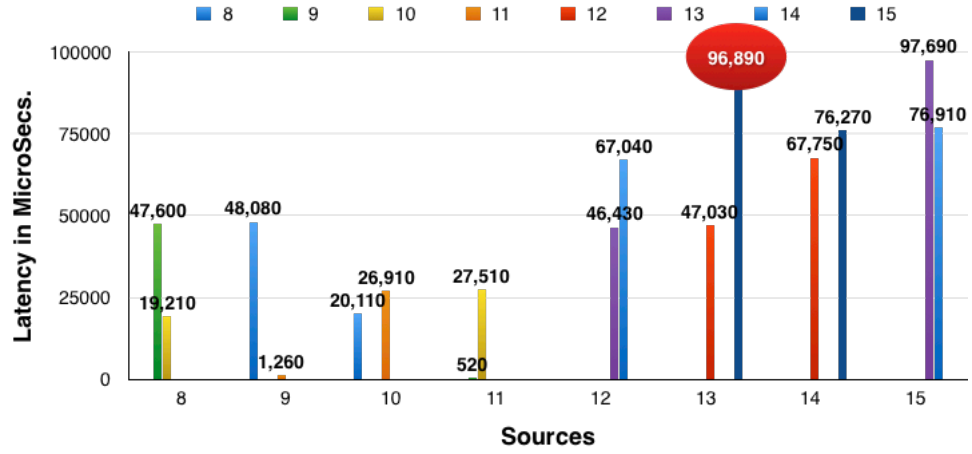
  Also is executed the *Ping Pong* calibration of communications between all the virtual machines, sending message of 1 MB. In this scheme from node 0 to node 1 a latency of 993 microseconds with the *Ping Pong* calibration is measured. If the path is changed, for example node 0 to node 7 to node 1 the latency will be 866 microseconds (green path in Fig. 4.7).

  This means that if the same technique is used with the message sent by the NAS between the same nodes the total latency of its message will be decreased around 12.7%.

- For Message Communication in the NAS-CG Class B, between node 7 to node 5 a latency of 119,819 is measured (Fig. 4.8a). Like the first one we also have the



(a) NAS-CG communication time between nodes 0 and 7



(b) NAS-CG communication time between nodes 8 and 15

Fig. 4.8 NAS-CG communication time

Ping Pong calibrations measures, sending message of 1 MB between all pairs of processes.

From node 7 to node 5 a latency of 897 microseconds is measured. If the route is changed, for example node 7 to node 16 to node 5 the latency will be 634 microseconds (purple path in Fig. 4.7). In this case MCM also will be working with the NAS message between those nodes, and the latency will be decreased around 29.26%.

- There are more communications that can reduce their latencies times, for instance, for message communication in the NAS-CG Class B, between node 13 to node

15 (Fig. 4.8b) the alternative path for it is marked in blue on Fig. 4.7. For illustration proposal not all paths and their respective alternative paths are marked.

- For Message Communication in the NAS-CG Class B, between node 2 to node 0 a total latency of 192,729 microseconds is measured (Fig. 4.8a). And then finally send message of 1 MB from node 2 to node 0 have a latency of 200 microseconds with the *Ping Pong* calibration. In this case the path of the message sent does not have to change, because there are not betters routes analyzing the behavior of the network with message of 1 MB.

With all the experiments it is verified that the proposed method can work, due to the existence of alternative paths. We find intermediate nodes that can be used to send the message and in which the sum of total latencies is less than the original path. Allowing to decrease the traffic congestion and the message latencies.

4.7.3 MCM Evaluation

Experiments are done in order to evaluate the MCM daemon software component like a first stage to test the method, with the monitoring, detection and the alternative path selection functionality. The experiments are performed with the NAS-CG application running the Class B and C in 16 nodes. The evalutions are done using t2.medium instances in Amazon EC2 cloud. Both applications with 1000 iteration to converge. We run this experiment with 16 processes. The experiment tries to expose the benefits of our method for the application execution time if a congestion occurs. Also tries to analyze the overhead added by the technique and evaluates the scalability of MCM daemons, testing it with different problem sizes of NAS-CG.

To test the functionality of MCM daemon and the overhead added by the middleware, a congestion is generated by another application between two nodes, causing a delay of 0.005 seconds in the link that connect those nodes, and NAS-CG running at the same time. The NAS-CG Class B without this congestion have an execution time of 247.64 seconds. The same application with MCM and without congestion have 260.61 seconds. This represents a 5% of overhead. On the other hand, with the congestion it have an execution time of 513.36 seconds. Instead with MCM and 2 alternative paths available to avoid congestion, the application have an execution time of 375.56 seconds. MCM reduce the execution time, in approximately 30% compared with the congested one. These results are shown in Figure 4.9.

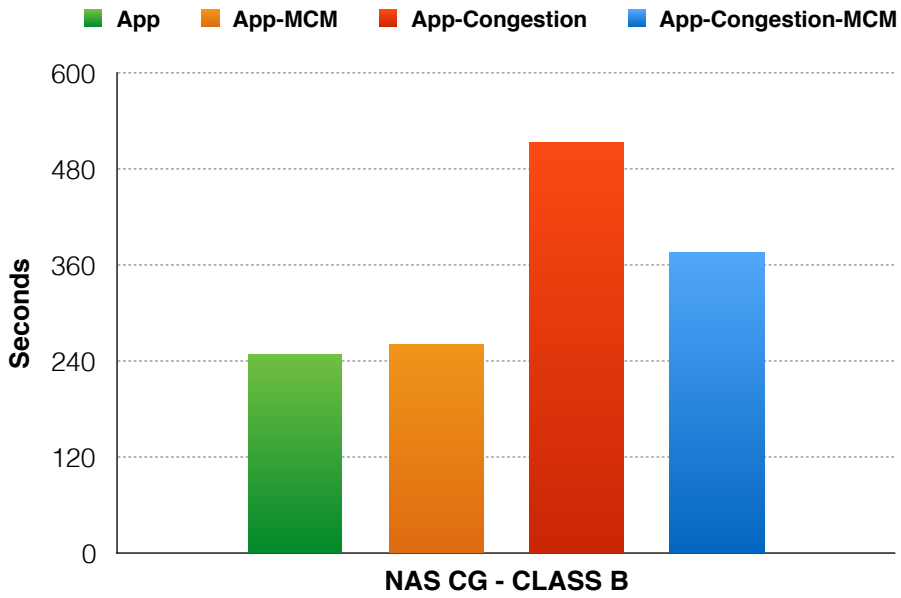


Fig. 4.9 MCM functionality and overhead added

The scalability of the method is evaluated testing MCM with 1 alternative path (the worst scenario), and different problem sizes of NAS-CG, also applied the congestion between two nodes by a delay of 0.005 seconds in the link that connect those nodes. The NAS-CG Class B without congestion have an execution time of 252.74 seconds, with congestion have an execution time of 782.33 seconds, with MCM avoiding congestion have an execution time of 655.15 seconds. These results are shown in Figure 4.10. In NAS-CG Class C, the results obtained have similar behaviors. NAS-CG Class C without the congestion have an execution time of 633.03 seconds, with the congestion have a execution time of 1320.82 seconds and with MCM avoiding congestion have a execution time of 1152.33 seconds.

The MCM method reduces the NAS-CG Class B execution time 16.2% compared to execution time of the same application with congestion, and with NAS-CG Class C reduce the execution time 12.7%.

The usage of the MCM daemon improves the application execution time if congestions occurs in links used for the application processes communications. The main idea is to avoid the congestion and its related links, trying to improve the usage of non-congested links.

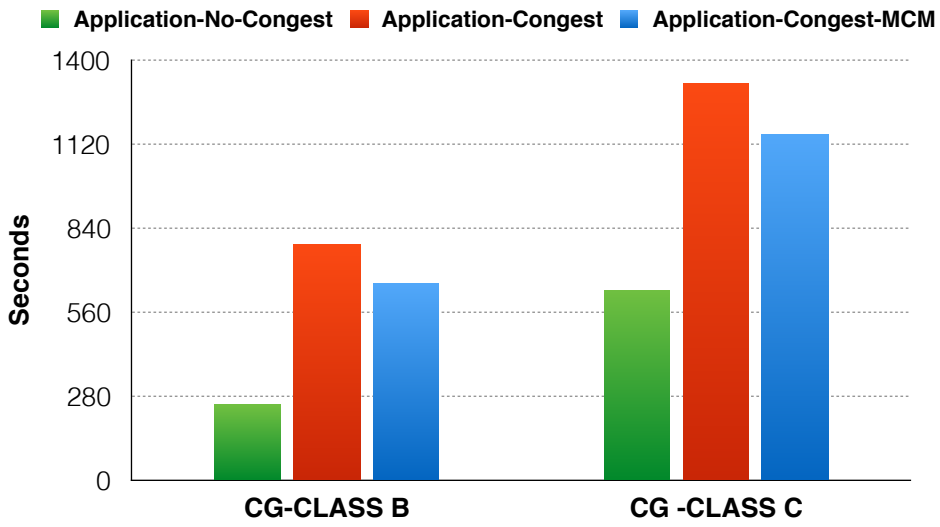


Fig. 4.10 MCM and NAS-CG Execution Times

4.8 DMCBM Evaluation in a public cloud

In this section an analysis of DMCBM improving the MCM daemon with an optimal ACK notification and the user mitigation process is done, and it is called DA-MCM. The functionality of this dynamic approach, that takes into account the application communication pattern is tested. A series of experiments evaluating latency of MPI communications and the total execution time of applications running on cloud are conducted.

4.8.1 Performance Analysis

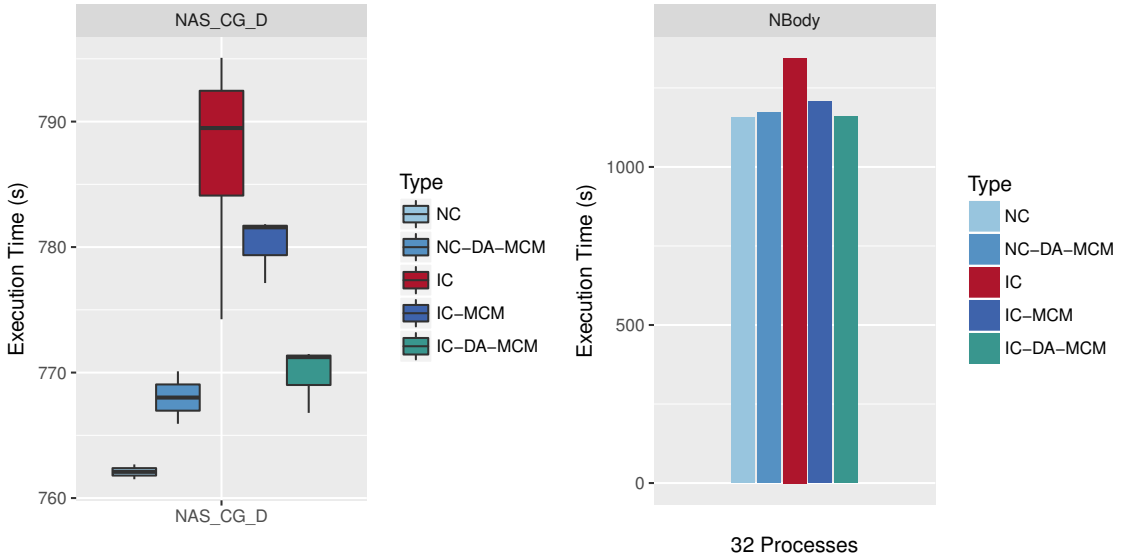
The NAS-CG is executed in a virtual cluster of 16 VMs on Amazon EC2 public cloud. The t2.medium and c3.2xlarge instances are used in Amazon EC2. The experiments expose the benefits of the DA-MCM method for the application execution time, by improving the processes messages latency time when congestions are detected. The results also show the added overhead and evaluate the scalability of DA-MCM, by testing it with different workloads 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 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. 4.11a, 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.

At the same time, we test and compare the presented DA-MCM with MCM, in a controlled execution environment, using a virtual cluster built with OpenStack. This experiment is done in order to notice the effects of the DA-MCM improvements over the MCM in the application execution. The NBody application is executed using m1.medium instances. In Fig. 4.11b, NBody simulation executed with DA-MCM reveals approximately 2% of overhead compared to execution without DA-MCM.



(a) Amazon EC2 (b) OpenStack

Fig. 4.11 DA-MCM analysis

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 MCM approach, as observed in the presented results.

4.8.2 Communications Analysis

In this section, the latency of the parallel application communication is measured and analyzed. This analysis is done to illustrate how the technique deals with congestion situations. The application used in this analysis is the NAS-CG Class D with 16 processes. Each process is executed in one VM of the virtual cluster.

In order to measure the communications of the application processes, they are captured using DA-MCM, which intercepts all messages sent between processes. After the interception, the message latency information is stored into a file. There is one file for each application process.

Figure 4.12 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 are turned into lower latency ones, denoting the

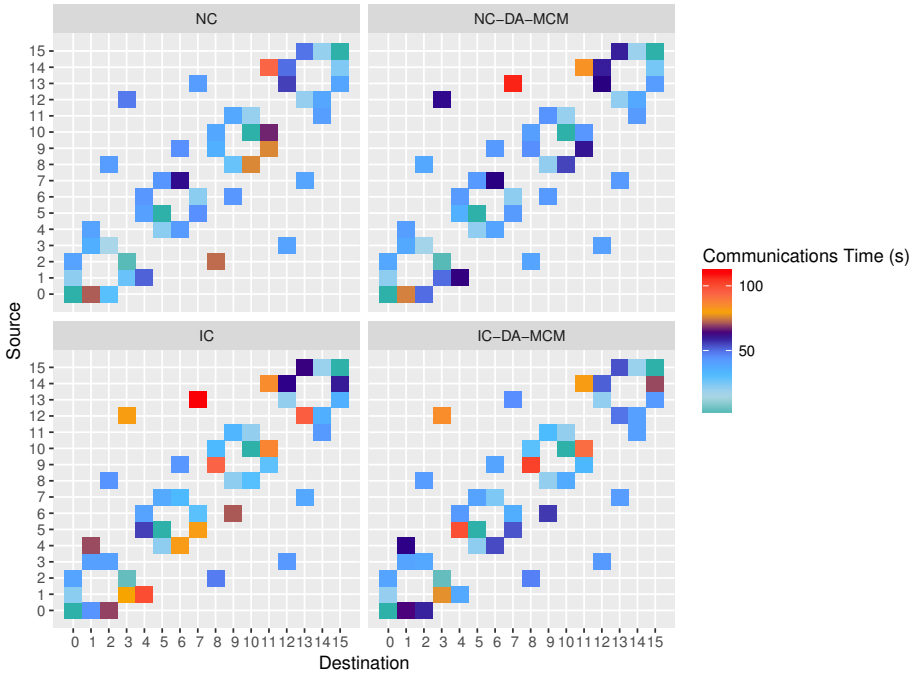


Fig. 4.12 Communications between processes

reduction of communication time between processes, when the proposed technique is applied.

4.8.3 Scalability Analysis

Mechanism scalability is evaluated, testing it with different problem sizes of the NAS-CG benchmark with different 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. 4.13). 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.

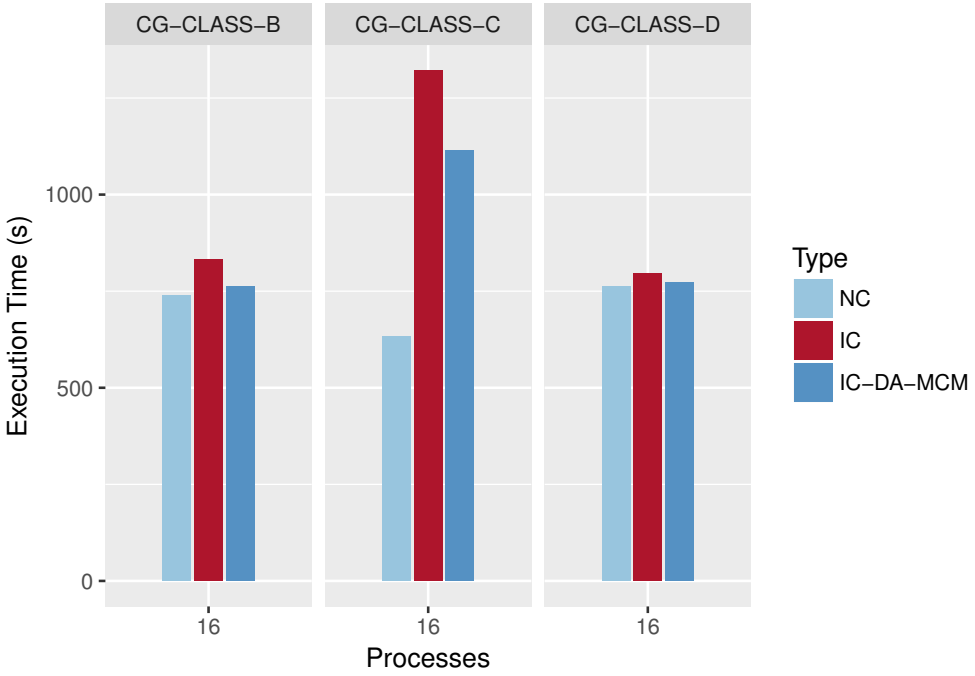


Fig. 4.13 DA-MCM scalability

DA-MCM leverages MCM improving even more the application execution time when congestion occurs in the virtual links of a cloud environment. It provides to users an efficient usage of virtual network resources. Lower message latency time allows the method to obtain improved application execution time in congestion scenarios.

4.9 DMCBM Evaluation in a private cloud

In this section, an evaluation of the Dynamic MPI Communication Balance and Management Evaluation (DMCBM) middleware applied to HPC applications executed in a private cloud is presented. For this execution environment, the implementation of our middleware is called MCBM. The evaluation is performed using MPI applications with repetitive communication patterns, by analyzing their performance, scalability and communications time in scenarios with and without congestion.

4.9.1 Performance Analysis

The performance of the different applications is analyzed, in order to verify the overhead introduced by the technique and to show how the application performance is improved when congestion scenarios appear and MCBM is applied, during the application execution. The experiments are repeated several time for the cloud variability, in order to achieve an statistical significance in the results. Maximum, minimum and average values of relevant metrics for each application are shown next.

In this analysis experiments are done using NAS-CG Class D and NBody applications, each of them with 32 processes distributed in 32 virtual nodes (or virtual machines). The virtual nodes are m1.medium instances type. For scenarios in which the technique is applied, there is one MCM daemon per virtual node. Therefore, there are 32 MCM daemons in total, 1 in each virtual node.

Figure 4.14 shows the execution time of the four scenarios under test, note that the overhead introduced by MCBM is less than 5% in congestion absence. The MCM daemons make traffic rebalancing decisions only during congestion, and have almost no impact with normal traffic. In the case of NAS-CG, the overhead is only slightly larger than for NBody even when both communication patterns are very different in topology and intensity. This indicates that MCMB overhead is decoupled from the communication traffic to some extent.

The experiment also shows the behavior of the applications in congestion situations, when MCBM is applied or not. The evaluation of the execution time for NAS CG

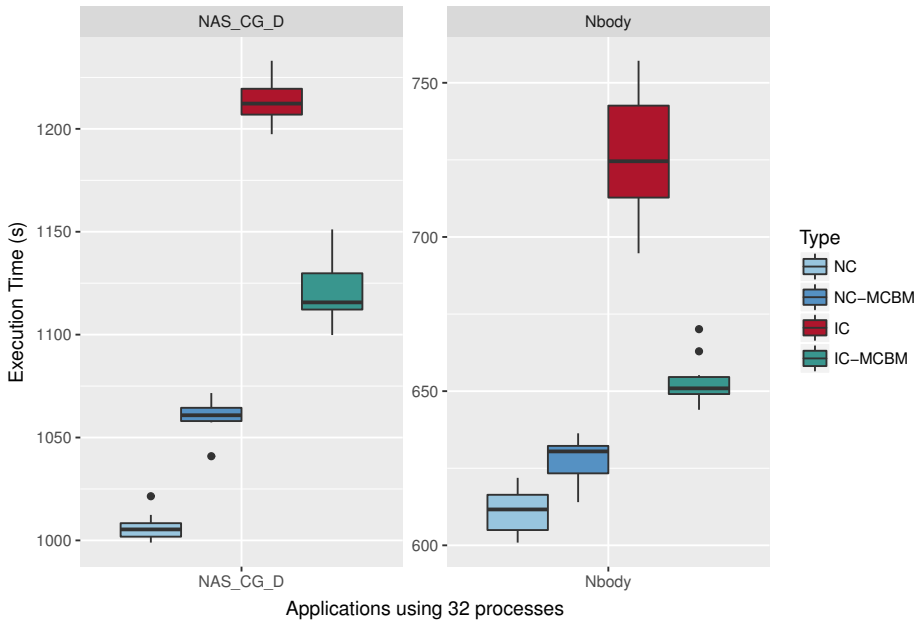


Fig. 4.14 Applications Execution Time

Class D and NBody applications in congestion situations shows an overhead in the range of 18% to 22% respect to the execution without any congestion. Applying MCBM technique in congestion situations, obtains a reduction of the execution time of approximately 10% compared to the execution without it, for the NAS CG Class D application.

A similar behavior is depicted for the NBody application with a different communication pattern, applying the technique, a reduction of at least 9% is obtained, showing the benefits of avoiding congestion applying MCBM. Furthermore, it is also possible to observe that for both applications MCBM provides more stable execution times, opposite to IC scenarios that show larger variations due to the lack of congestion management.

The live-migration operations are analyzed during the evaluations. We measured each migration operation during the application execution, when MCBM is activated. Figure 4.15 shows the experiment results for the NAS CG benchmark and the NBody application. In the results is observed that when the migration is performed during the execution of the NAS-CG application, the operation takes significantly longer time compared to the NBody application. Hence, it is possible to state, that the live migration operation time is related to the application communication pattern, which affects the execution time of the application. The amount of times that the migration is done, depends on several factors, such as the network status, more accurately, the

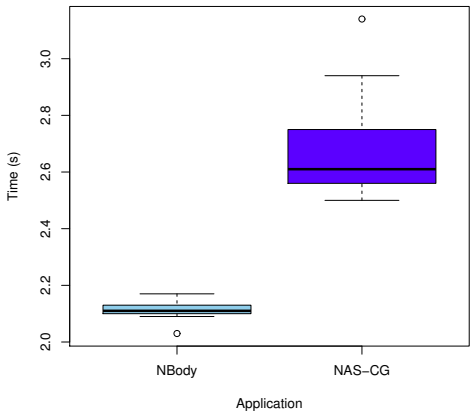


Fig. 4.15 Live-migration operation analysis for NAS-CG and NBody applications.

detection of destination congestion by the MCBM. It is important to remark, that although the migration operation has an associated cost, it is worth to pay compared to the unpredictable over-cost that a congestion scenario may incur over the application execution.

4.9.2 Scalability Analysis

The scalability analysis of MCBM is done, in order to verify how the technique works when user's applications is scaled in quantity of processes and application workload size, taking into account the performance in terms of the application execution time. The applications are executed with 16 and 32 processes in 16 VMs using m1.medium instance type.

Figure 4.16 shows how the technique successfully scales with the application. The overhead added when applying the technique despite the amount of processes used remains similar in the range of 5%. Similarly, MCBM maintains the reduction of 10% in execution time for both applications compared the congestion scenario without MCMB support.

MCBM improves the application execution time when destination and link congestion occurs, choosing an alternative path if the congestion is present in the link and performing a virtual machine live migration if a destination congestion occurs. During the experiments, a destination congestion occurs and a virtual machine live migration is done to avoid this congestion.

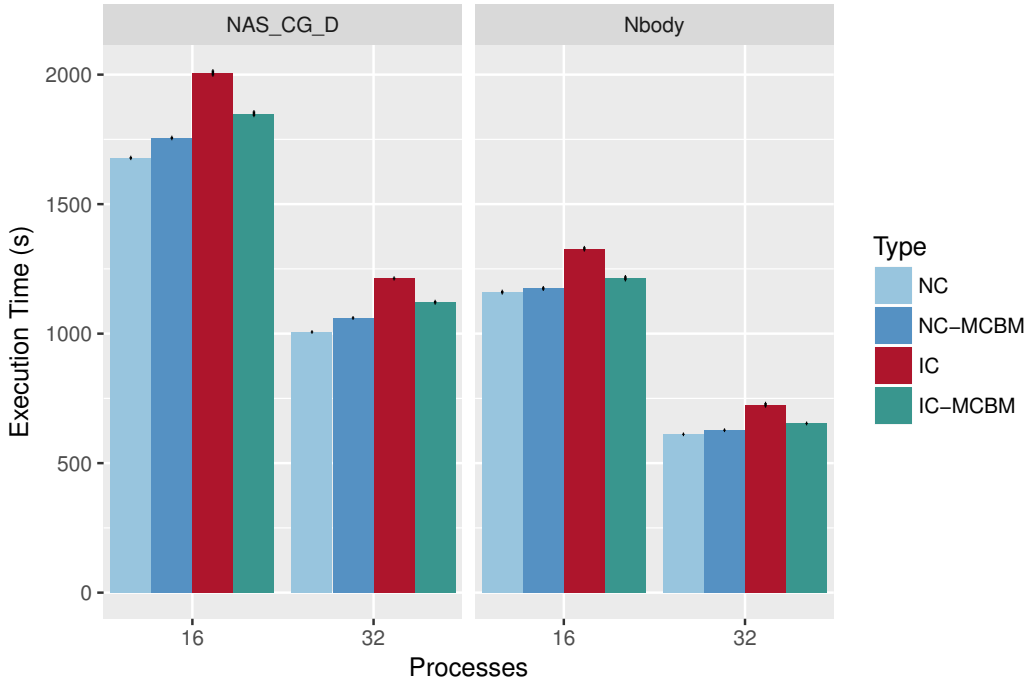


Fig. 4.16 Applications with different quantity of processes

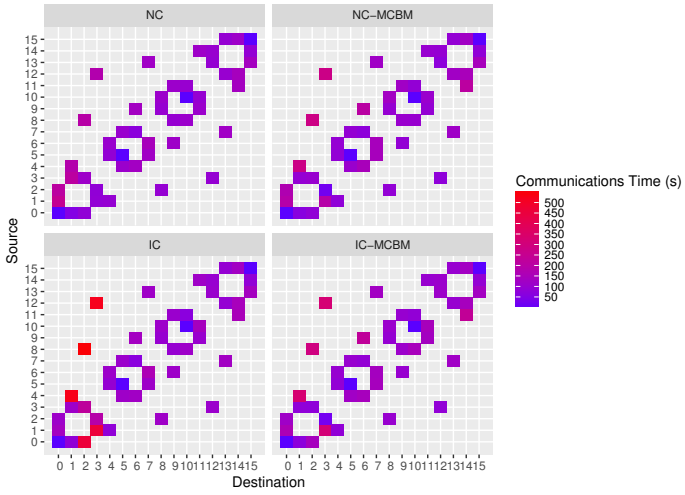
4.9.3 Communications Analysis

The communications time analysis is performed to illustrate how MCBM tackles congestion scenarios reducing the latency of the MPI application communications in cloud.

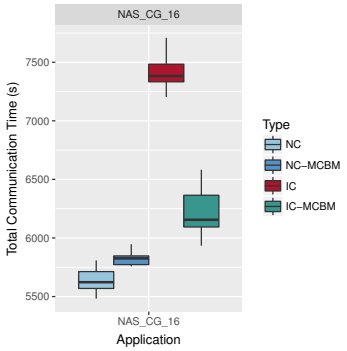
For the analysis of the communications, the experiments were performed using both applications: NAS CG Class D and NBody with 16 and 32 processes. During the experiments, the communication pattern for both applications remained the same when scaling them.

The communications were captured using the MCM daemon. We leverage the monitoring component of MCM to store latency values in a file for collecting the results shown in this section. There is one file per application process, and contains all communications performed by the process throughout the execution. For the scenarios with congestion situations, the congestion is created using a synthetic application, which communicates between its processes, in parallel to the user application execution.

Figure 4.17a shows the communications pattern of the NAS-CG benchmark. It also depicts the total communication time for each source-destination pair communicating processes. There are 16 processes running in 16 virtual machines. Note that in scenarios



(a) Communication time between processes



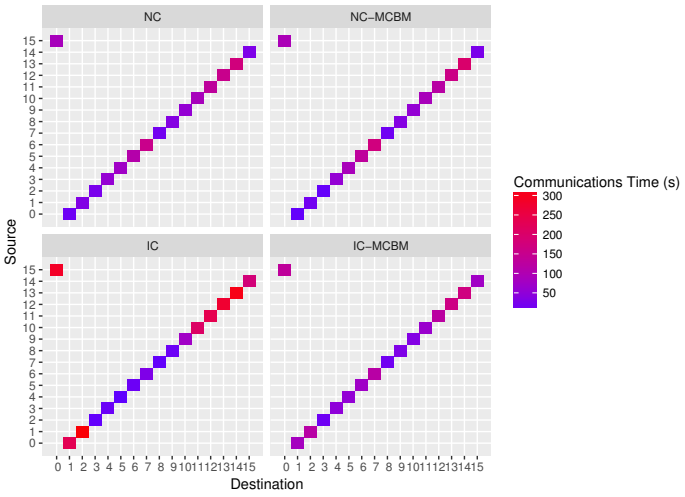
(b) Application total communication

Fig. 4.17 NAS CG CLASS D Benchmark with 16 processes

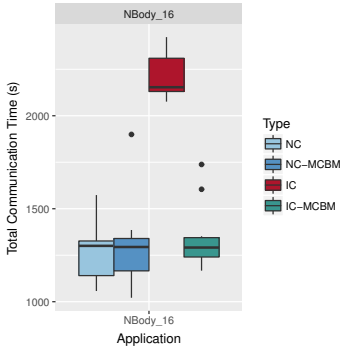
with congestion (IC), the latency for both applications is significantly increased, whereas with MCBM support (IC-MCBM) congestion can be reduced to values similar to the congestion free case (within $\sim 3\%$). The application total communication time of the Fig. 4.17b, is obtained by accumulating the latency results of each communication pair. Note that when congestion scenarios appear, the application communication time is drastically affected, implying a high-penalty for the users in terms of execution time for their application. MCBM comes with an overhead for the users application execution, although when congestion scenarios are detected, the technique avoid uncontrolled impact on the application communication time.

In Figure 4.18a, similar results are obtained for the NBody application. The communication pattern in this case is represented as a pipeline between process. Results indicate how the technique improves communications in the case of congestion scenario compared to the execution without applying MCBM (which add an overhead of approximately 3% in congestion free scenarios). The figures illustrate the pairs with congestions, and how the congestions are avoided with the MCBM, acquiring better communications distribution by using MCBM. It depicts that high latency hot-spots are turned into lower latency ones, denoting the reduction of communication time between processes, when MCBM is applied. For this application, with a different communication pattern, MCBM also reduce the application total communication time in congestion scenarios as shown in Fig. 4.18b. This result is obtained comparing the

scenario with congestion (IC) and the scenario with congestion but applying MCBM (IC-MCBM)



(a) Communication time between processes



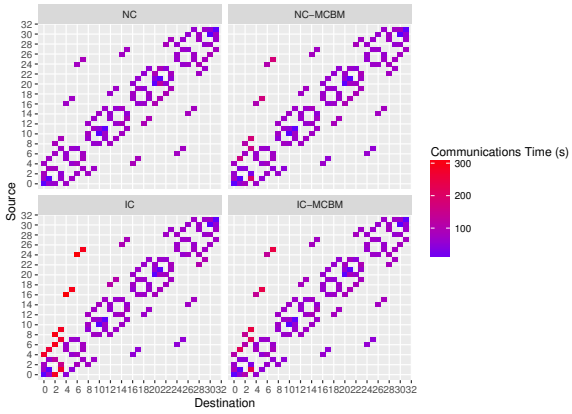
(b) Application total communication

Fig. 4.18 NBody, simulation of a dynamical system of particles with 16 processes

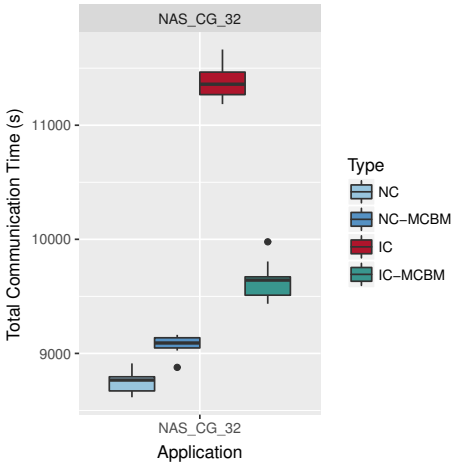
For both applications with 16 processes in 16 virtual nodes. In scenarios where a congestion situation is created (IC), the communication time for both applications is significantly affected. For the application execution in congestion scenarios with MCBM applied, the application communication time is reduced between different pair of processes (IC-MCBM), achieving a reduction of the application total communication time.

The MCBM technique reduces the application total communication time for congestion scenarios of the NAS CG Class D on about 16% compared to the execution without MCBM. Similar results are obtained for the NBody application with an improvement of the application total communication time, which is in range of 30%.

We have scaled out the applications to 32 virtual machines and repeated the experiments above. Figures 4.19a and 4.19b show the communication pattern between each pair of processes for the NAS-CG, as well as the application total communication time. Figure 4.19a shows the re-balance of the communications between processes performed by MCBM in congestion scenarios (IC-MCBM), compared with congested scenario (IC). With the re-balance performed by MCBM, a reduction of approximately 15%, in terms of application total communication time, is obtained compared to the execution without the technique.



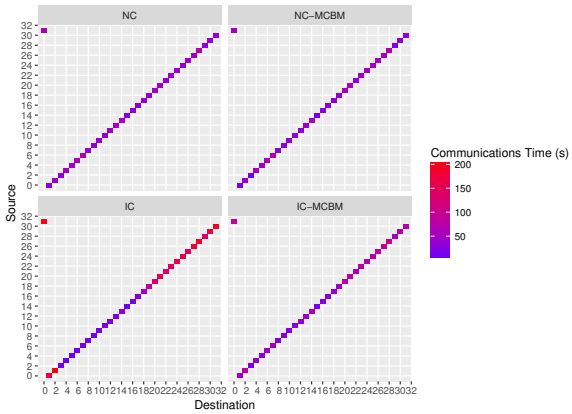
(a) Communication time between processes



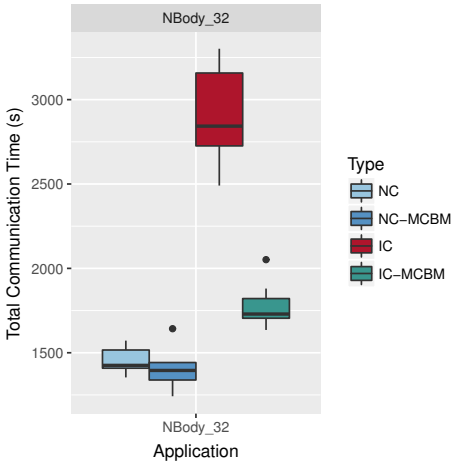
(b) Application total communication

Fig. 4.19 NAS CG CLASS D Benchmark with 32 processes

Last, Figures 4.20a and 4.20b show same results for the NBody application with 32 virtual machines. The figures also highlight the overhead added by MCBM and the total communication time reduction, which is the effect of the communication distribution performed by the MCBM technique. With the distribution done by MCBM, a reduction of around 40% is obtained compared to the execution without the technique. Summarizing, the figures illustrate pairs of communication processes with congestion, and how they are avoided taking into account the application communication pattern with the MCBM, acquiring better communications distribution.



(a) Communication time between processes



(b) Application total communication

Fig. 4.20 NBody, simulation of a dynamical system of particles with 32 processes

4.10 Comparative with others techniques

In this section, a comparative study with the state of the art techniques is done. Some of them use processes reallocation to improve the HPC communications in cloud environments, for e.g.: Gomez-Folgar et al. [14] obtain latency improvement of 23% for benchmarks applications, in comparison with our technique, in which using virtual machine live migration we obtain a latency improvements of about 40%. For real applications, they obtain an improvement of around 4%, and for special cases 26%, both improvements are in terms of execution time and depends on the application. With similar kind of applications, we managed to improve the execution time in at least 7%. For example for a NAS CG benchmark application applying MCBM we obtain a reduction of 16% in congestion scenarios using a the Class D. Hence, with a real application of a dynamical system of particles NBody we obtain a reduction of 10% in congestion scenarios with 10000 particles and 50 iterations.

Another kind of solution is presented by Gong et al. [16], they develop network performance aware algorithms for MPI collective operations, though not resolving the issues of point to point communications. This approach requires MPI library modification. Our technique deals with point to point communications, and dynamically avoids congestion during the application execution without modifications in the MPI library. Furthermore, we obtain similar benefits in terms of application execution time and communications.

We have to remark that in all the analysis, the experimental environments, and in some case the applications are no the same. The comparison only gives an idea of how the existing techniques are related to our proposed solution in a quantitative and qualitative analysis. It demonstrates that the improvements of our technique related to communications latency time and execution time reduction are in an aceptable range or in some cases even better than others techniques.

4.11 Summary

In this chapter the evaluation of the *Dynamic MPI Communication Balance and Management* (DMCBM) middleware is presented. First, an analysis of the system characterization is performed. With this analysis is possible to notice that several alternatives paths, can be found and used during application runtime applying our technique. Then, the MCM daemon is also analyzed, in order to test the selection of alternative paths and how this affect to the application execution.

The components that are part of DMCBM were tested using several applications and execution environments including public and private clouds. For benchmark and real applications executed the reduction of the execution time in congestion scenarios applying the technique is approximately 10%. Furthermore, improvements in the application execution time can be achieved thanks to the latency reduction in the application communications that are up to 40%.

As shown in this evaluation, DMCBM helps users to improve their application's execution time, when congestion situation is presented. The presented technique successfully avoid congestion, distributing the application processes communication by selecting alternative path, or live-migrating virtual machines when destination congestion is detected. With the presented results, it is also noticeable, that by lowering application processes message latencies, we can achieve an improvement on the application execution time. The fundamental concept behind the middleware is to avoid congested links, improving the usage of non-congested ones in the execution environment.

Chapter 5

Conclusions and Future Works

"The greatest victory is that which requires no battle."

– Sun Tzu, *The Art of War*

5.1 Final Conclusions

This thesis studies the migration of HPC applications to cloud environments and proposes a middleware to overcome the communications challenges that appears. It presents several aspects like the state of the art study and design of new techniques, along with the evaluation of the presented solution. However, the work also open new lines of research and future works.

5.1.1 Dynamic MPI Communications Balance and Management

In this work the main area of study is how to improve HPC applications communications in cloud, due to the lost of performance that applications experiments in this kind of environments. The proposal aims to HPC applications, which use the MPI standard. In order to improve the performance of MPI applications in the cloud is needed a better manage of MPI communications in a virtual network or the modification of MPI sentences.

This thesis proposal, Dynamic MPI Communications Balance and Management (DMCBM), 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.

DMCBM provides the management of MPI communications in virtual network environments. The technique operations are done in parallel with the application execution and without modifying neither the application nor the MPI library implementation.

DMCBM middleware characterizes the underlying network topology, and looks for alternative paths to avoid congestion situations by distributing the messages communications between links with less usage, considering the variability of Cloud networks and the dynamic traffic behavior. It detects links and host destination congestions, and provides mechanisms to avoid them, using alternative paths and virtual machine live migration.

The proposed solution allows to obtain better execution and communication time in case of congestion for MPI applications running on clouds. DMCBM can provide an efficient cost-benefit trade-off of resource usage to cloud users, taking in to account the application communication pattern. Experiments done demonstrated the viability of the proposed technique applied to real-world and benchmarks applications executed in public and private clouds. We also present a comparative analysis of DMCBM with other techniques of the literature, showing the benefits of our method in terms of communication time of at least 30% and execution time of at least 7% in congestion scenarios.

5.2 Future Works

This work tries to be the most complete research to fulfill the main goal of this thesis, which is to improve communications in cloud. However, some parallel future works appear during this years. Some of them are the following:

- Our middleware can be integrated in new architectures like Software Defined Networking using cloud. In order to detect alternative paths for MPI applications communications management in this kind of architectures.
- Look at improving the DMCBM mechanism using different technologies in private clouds, such as containers and with this, reducing the costs without losing application performance.
- Use the elasticity characteristics of cloud to dynamically provide new virtual machines in case of destination congestion.

References

[1] Amazon (2018). AWS, High Performance Computing. https://aws.amazon.com/hpc/. Accessed: 2018.09.11.

[2] AmazonEC2 (2018). Instances of amazon ec2. https://aws.amazon.com/es/ec2/instance-types/. Accessed: 2018-09-11.

[3] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I. (2010). A view of cloud computing. *Magazine Communications of the ACM*, 53(4).

[4] Balis, B., Figiela, K., Jopek, K., Malawski, M., and Pawlik, M. (2017). Porting HPC applications to the cloud: A multi-frontal solver case study. *Journal of Computational Science*, 18:106–116.

[5] Dashdavaa, K., Date, S., Yamanaka, H., and Kawai, E. (2014). 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.*, pages 885–894.

[6] Espínola, L., Franco, D., and Luque, E. (2015). MPI Communications Management in Cloud. In *JORNADAS DE PARALELISMO, JP 2015*, pages 331–336.

[7] Espínola, L., Franco, D., and Luque, E. (2015). MPI Communications Management in Cloud. In *Int'l Conf. Par. and Dist. Proc. Tech. and Appl. / PDPTA'15 /*, pages 673–679.

[8] Espínola, L., Franco, D., and Luque, E. (2016). Improving mpi communications in cloud. In *ACM-W Europe WomENcourage 2016 Celebration of Women in Computing.*

[9] Espínola, L., Franco, D., and Luque, E. (2017). ScienceDirect MCM: A new MPI Communication Management for Cloud Environments. *Procedia Computer Science*, 108:2303–2307.

[10] Espínola, L., Franco, D., and Luque, E. (2018). DA-MCM: A Dynamic Application-Aware Mechanism for MPI Communications in Cloud Environments. In *Int'l Conf. Par. and Dist. Proc. Tech. and Appl. / PDPTA'18 /*, pages 211–217.

[11] Expósito, R. R., Taboada, G. L., Ramos, S., Touriño, J., and Doallo, R. (2013). Performance analysis of hpc applications in the cloud. *Future Generation Computer Systems*, 29(1):218–229.

[12] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open mpi: Goals, concept, and design of a next generation mpi implementation. *In Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104.

[13] Goga, K., Pilosu, L., Parodi, A., Lagasio, M., and Terzo, O. (2019). Performance of WRF Cloud Resolving Simulations with Data Assimilation on Public Cloud and HPC Environments. pages 161–171. Springer, Cham.

[14] Gomez-Folgar, F., Indalecio, · . G., Seoane, · . N., Pena, · . T. F., Garcia-Loureiro, · . A. J., and Gomez-Folgar, B. F. (2018). MPI-Performance-Aware-Reallocation: method to optimize the mapping of processes applied to a cloud infrastructure. *Computing*, 100:211–226.

[15] Gomez-folgar, F., Indalecio, G., Seoane, N., Garcia-Loureiro, A. J., and F. Pena, T. (2016). Study of Point-to-Point Communication Latency for MPI Implementations in Cloud. *The 22nd International Conference on Parallel and Distributed Processing Techniques and Applications*, page 60132.

[16] Gong, Y., He, B., and Zhong, J. (2015). Network Performance Aware MPI Collective Communication Operations in the Cloud. *IEEE Transactions on Parallel and Distributed Systems*, 26(11):3079–3089.

[17] GoogleCloud (2018). Setting instance availability policies. https://cloud.google.com/compute/docs/instances/setting-instance-scheduling-options. Accessed: 2018-09-11.

[18] Gupta, A., Faraboschi, P., Gioachin, F., Kale, L. V., Kaufmann, R., Lee, B.-S., March, V., Milojicic, D., and Suen, C. H. (2016). Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud. *IEEE Transactions on Cloud Computing*, 4(3):307–321.

[19] Gupta, A. and Milojicic, D. (2011). Evaluation of HPC applications on cloud. *Proceedings - 2011 Fifth Open Cirrus Summit, OCS*, pages 22–26.

[20] Gupta, A., Sarood, O., Kale, L. V., and Milojicic, D. (2013). Improving HPC application performance in cloud through dynamic load balancing. *Proceedings - 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013*, pages 402–409.

[21] Hassan, H. A., Mohamed, S. A., and Sheta, W. M. (2016). Scalability and communication performance of HPC on Azure Cloud. *Egyptian Informatics Journal*, 17(2):175–182.

[22] Hassani, R., Aiatullah, M., and Luksch, P. (2014). Improving HPC Application Performance in Public Cloud. *IERI Procedia*, 10:169–176.

[23] He, Q., Zhou, S., Kobler, B., Duffy, D., and McGlynn, T. (2010a). Case study for running hpc applications in public clouds. *HPDC*.

[24] He, Q., Zhou, S., Kobler, B., Duffy, D., and Mcglynn, T. (2010b). Case Study for Running HPC Applications in Public Clouds. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC'10*, pages 395–401.

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

[26] Koibuchi, M., Matsutani, H., Amano, H., Hsu, D. F., and Casanova, H. (2012). A case for random shortcut topologies for HPC interconnects. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 177–188. IEEE.

[27] Ledyayev, R. and Richter, H. (2014). High Performance Computing in a Cloud Using OpenStack. *CLOUD COMPUTING 2014 : The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization*, (c):108–113.

[28] Li, Z., He Zhang, C., and Cai, R. (2012). On a Catalogue of Metrics for Evaluating Commercial Cloud Services. *2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 164–173.

[29] Mariani, G., Anghel, A., Jongerius, R., and Dittmann, G. (2017). Predicting Cloud Performance for HPC Applications: A User-Oriented Approach. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 524–533. IEEE.

[30] Mauch, V., Kunze, M., and Hillenbrand, M. (2013a). High performance cloud computing. *Future Generation Computer Systems*, 29(6):1408–1416.

[31] Mauch, V., Kunze, M., and Hillenbrand, M. (2013b). High performance cloud computing. *Future Generation Computer Systems*, 29(6):1408–1416.

[32] Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, 145:7.

[33] Netto, M. A. S., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., and Buyya, R. (2018). HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. *ACM Computing Surveys*, 51(1):1–29.

[34] Nuñez, C., Lugones, D., Franco, D., Luque, E., and Collier, M. (2013). Predictive and distributed routing balancing , an application-aware approach. *International Conference on Computational Science, ICCS*, 00.

[35] Oracle (2018). Bare Metal Cloud Services. https://cloud.oracle.com/bare-metal. Accessed: 2018.09.11.

[36] Panda, D. K. and Lu, X. (2018). HPC Meets Cloud: Building Efficient Clouds for HPC, Big Data, and Deep Learning Middleware and Applications *. 17.

[37] Polash, F., Abuhussein, A., and Shiva, S. (2014). A survey of cloud computing taxonomies: Rationale and overview. In *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*, pages 459–465. IEEE.

[38] R Systems (2017). Dedicated Clusters. http://rsystemsinc.com/. Accessed: 2018.09.11.

[39] Righi, R. d. R., Rodrigues, V. F., da Costa, C. A., Galante, G., de Bona, L. C. E., and Ferreto, T. (2016). AutoElastic: Automatic Resource Elasticity for High Performance Applications in the Cloud. *IEEE Transactions on Cloud Computing*, 4(1):6–19.

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

[41] StarCluster (2011). Starcluster documentation (v.0.95.6). http://star.mit.edu/cluster/index.html. Accessed: 2018-09-11.

[42] Stockton, D. B. and Santamaria, F. (2017). Automating NEURON Simulation Deployment in Cloud Resources. *Neuroinformatics*, 15(1):51–70.

[43] Subramoni, H., Kandalla, K., Vienne, J., Sur, S., Barth, B., Tomko, K., Mclay, R., Schulz, K., and Panda, D. (2011). Design and Evaluation of Network Topology-/Speed- Aware Broadcast Algorithms for InfiniBand Clusters. In *2011 IEEE International Conference on Cluster Computing*, pages 317–325. IEEE.

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

[45] Telfer, S. (2016). *The Crossroads of Cloud and HPC: OpenStack for Scientific Research Exploring OpenStack cloud computing for scientific workloads.*

[46] Walker, E. (2008). benchmarking amazon ec2 for high-performance scientific computing. *Program*, pages 18–23.

[47] Wong, A., Rexachs, D., and Luque, E. (2015). Parallel Application Signature for Performance Analysis and Prediction. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):2009–2019.

[48] Zahavi, E. (2012). Fat-tree routing and node ordering providing contention free traffic for mpi global collectives. *Journal of Parallel and Distributed Computing*, 72(11):1423–1432.

[49] Zahid, F., Gran, E. G., Bogdanski, B., Johnsen, B. D., and Skeie, T. (2015). A Weighted Fat-Tree Routing Algorithm for Efficient Load-Balancing in Infini Band Enterprise Clusters. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 35–42. IEEE.

[50] Zhang, J., Lu, X., Jose, J., Li, M., Shi, R., and Panda, D. K. D. K. (2014). High performance MPI library over SR-IOV enabled infiniband clusters. In *2014 21st International Conference on High Performance Computing (HiPC)*, pages 1–10. IEEE.

[51] Zhang, J., Lu, X., and Panda, D. K. (2017). High-Performance Virtual Machine Migration Framework for MPI Applications on SR-IOV Enabled InfiniBand Clusters. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 143–152. IEEE.