Universitat
Autònoma
de Barcelona

PERFORMANCE PREDICTION AND TUNING IN A MULTI-CLUSTER ENVIRONMENT

Computer Architecture
and Operating Systems
Department

Thesis submitted by **Eduardo Argollo de Oliveira Dias Júnior** in fulfillment of the requirements for the degree of Doctor per la Universitat Autònoma de Barcelona.

Barcelona, June 6, 2006

PERFORMANCE PREDICTION AND TUNING IN A MULTI-CLUSTER ENVIRONMENT

Thesis submitted by **Eduardo Argollo de Oliveira Dias Júnior** in fulfillment of the requirements for the degree of Doctor per la Universitat Autònoma de Barcelona. This work has been developed in the **Computer Architecture and Operating Systems** department of the **Universitat Autònoma de Barcelona** and was advised by **Dr. Emilio Luque Fadón**.

Thesis Advisor

Emilio Luque Fadón

Barcelona, June 6, 2006

Aos meus pais

que me ensinaram a sonhar e acreditar no sonho.Que, para realizar este sonho, me ensinaram a navegar. Me deixaram partir com a garantia de sempre ter um porto seguro.

Für meine liebe Jana,

dafür das sie mich auf meinem Weg begeleitet hat, und Dank ihrer Zuneigung auch schwierige Momente zu glücklichen wurden.

Acknowledgements

When I first took the decision to leave my homeland, job, family and friends to a four-year PhD journey in Spain, I was not really aware of what a PhD was all about. My dream was to become able to fulfill the love for lecturing and interacting with students. I could never imagine how intense this period would be. Adaptation to another country, depressing Sundays, a total new way to approach problems, lots of hard work, wonderful and memorable trips, admirable people and different cultures: So much to learn in such different fields.

This PhD thesis represents the beginning of a new stage in my life and could not be completed without many people direct and indirect support. Not just during this four-year period, but along my entire life.

First of all, I need to thank my parents, Eduardo and Terezinha, for all their personal sacrifice to give me the educational background that brought me here. They were directly responsible for making me feel an endless need for knowledge and this special curiosity to understand how things work. They always taught me with examples the best possible inheritance I could ask in moral and ethical principles.

I am also thankful to my brother Leonardo Santana who shared with me unforgettable and appreciable moments along life. His natural motivation, determination and outgoing personality were always an inspiration. I would like to thank my 'bro' Douglas and his wife Theano for all the words of experience, love and support, especially at those worst 'giving-up moments'. I am thankful also to my nephews and nieces, Mary Helen, Dino and the recently born Guilherme who give me unconditional and sincere love. They remind me of the amazement with the world that just the kids are able to have.

Thanks to Jana, my beloved girlfriend, for all her support, affection and care. For being able to handle the distance and my temper so well, and for all the wonderful and unforgettable moments enjoyed around the world. Meeting you was the best gift I could have gotten from all of this adventure.

I would like to thank my working group, Emilio, Lola, Angelo and Guna, for all the unforgettable meetings and discussions that so much helped to provide content to this work. My thesis director Emilio was always a source of inspiration, giving me immensurable advices and all support. Deliberations with him are always delightful and memorable. Emilio was responsible for challenging me, setting some apparently "impossible" targets, putting pressure and deadlines, sometimes almost driving me crazy. All of this in name of science and of my personal growing. My other special director Lola is an admirable and sensible person who was always there to give me inestimable guidance and to make me believe in myself in the most difficult moments. They both are more than directors; they are truly friends. In the lack of better words to give the dimension of how thankful I am, I will put it simple: ¡Muchísimas gracias!

Of course I can not forget the others genuinely friends that I met in here and that shared with me not just the working hours, but also most of the leisure moments. Thanks Mauricio and Maru, the couple that I consider my uncle and aunt, for all their listening, opinions and family-care. For 'Dieguito', Angelo, Genaro, Guna and Paula, my best friends; invariable presence in all weekends of work as well as all weekends of fun. Thanks Angelo not just for the nice questioning at my work but also for being a wonderful home mate, Genaro for being an endless source of knowledge, ideas and patience; a great "guru" and a true friend. Guna for all the nice talks, fun and some great games match. Thanks to Carlos for his friendship and for some of the best parties ever. I am thankful to 'Jonny' Xu, for all discussions about life, future, ideas and also C++ and TCP/IP.

Thank you to the technical staff, Jordi and Dani, for all their support, patience and fun. I can't also forget to thank to every member of the Computer Architecture and Operating System Department. They make the best 'CAOS' environment to share and generate knowledge.

I am also very thankful to Josemar, Andre, 'Peu' and all members of Brazilian CEBACAD and to Adriana and the team in Argentina, for all help, advices and for making all the presented experiments possible. Special thanks to Josemar for his recommendation. I am also thankful to Petra for the English revisions along the PhD and for all the support.

Finally, I would like to thank my best Brazilian friends Walter, Murillo, Claudia, João, Ruy, Senna, Kennio, Paulo Jorge, and Alfrânio. It is good to feel that you are always by my side, no matter the distance or circumstance.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

The advances of general purpose programmable computers have brought revolutionary changes into the world as it was known by providing the possibility of obtaining fast and efficient solutions to all sorts of problems. Over the past decades, computers became part of everyday's life and their processing power exponentially increased, duplicating approximately every year and a half. This fact was first observed by Gordon Moore (Intel's co-founder) in 1965 and is widely known as Moore's law.

Despite this evolution of processing power, there is a wide range of problems that cannot be solved in a reasonable amount of time with today's computers. These problems are called Grand Challenge problems [77]. One alternative for reaching an acceptable time to solve these problems is to split the problem and employ several processors simultaneously in parallel, reaching the solution in a fraction of the single-computer's original time. This model is called parallel computing. Parallel computing can be defined as simultaneous use of more than one computer, or processor, working together to solve a common problem [25].

There are several ways to obtain the goals of parallel computing differing on processors communication strategy (shared or distributed memory), parallel programming model (threads, message passing), algorithm programming paradigm (master-worker, pipeline, divide and conquer, …), etc.

Nowadays the best cost-performance solution for parallel computing is the use of standard workstations interconnected by a network. This approach is known as cluster computing [25][26] and became even more popular with the use of commodity-of-the-shelf (COTS) components as presented at the Beowulf project [73] [L2].

Because of its relative low-costs and simplicity, clusters of workstations (COW) have become a pervasive supercomputing alternative, now seen in industry and academia worldwide.

Clusters might be *homogeneous*, i.e. formed by several identical computers, or *heterogeneous*, having computers with different hardware configurations. Heterogeneous clusters of workstations are gradually becoming the natural COW configuration because of the commodity nature of workstations and the multiplicity of vendors and platforms. Even homogeneously built clusters tend to become heterogeneous along time through computers replacement or new acquisition.

The heterogeneity brings some drawbacks to applications speedup like problem decomposition, scheduling, balancing the load, mapping and processor selection [31]. Several works propose parallelization techniques to overcome some of these problems in heterogeneous clusters of workstations [17][20][32][64].

Computational problems faced by those heterogeneous clusters of workstations (HCOW) are increasing in size and complexity. However, clusters limited performance prevents the problems' study grow.

Internet's evolution made possible to surpass the single cluster performance limitations by the joint of scattered HCOWs in a cooperative meta-computer, a multi-cluster. Multi-cluster computing systems are formed from a set of independent clusters interconnected by a wide-area network (WAN) (Figure 1-1).

Nowadays there is an emphasis on enabling applications to access resources in different and possibly widely dispersed locations, through computational and data GRIDs [38][47]. A collaborative multi-cluster application execution can occur as a result of the direct specific joint of clusters or as a result of a job request allocation from a GRID broker [40].

However, an effective execution speedup through multi-cluster usage is challenged by the addition of new levels of heterogeneity [58]. Despite of the heterogeneity within each cluster, in a multi-cluster it is also relevant the heterogeneity among clusters. A multi-cluster can be formed by clusters with different global performances and local area networks. There might also be differences in the latency and throughput in each pair of clusters communication.

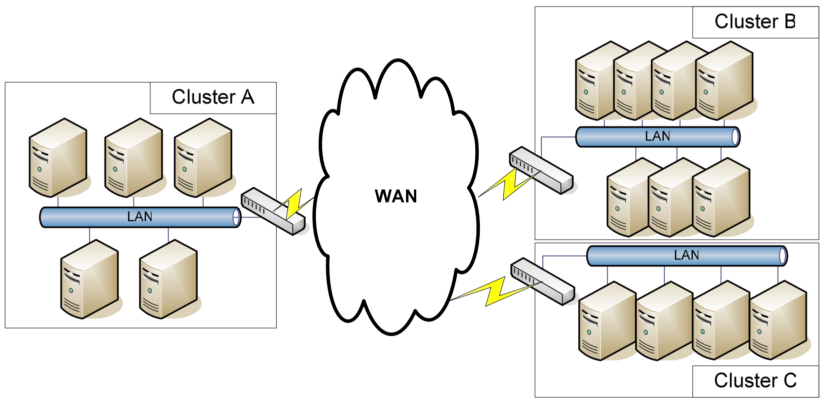


Figure 1-1: A multi-cluster as a group of clusters interconnected through a wide area network.

Another remarkable aspect that improves the difficulties in multi-cluster performance obtainment is the difference between clusters' local area networks (intra) and the wide-area network (inter) used to interconnect clusters [50][66]. Locally, in a cluster, computers are connected through a high-throughput, low latency and dedicated network. On the other hand, generally the network used to interconnect clusters is shared with other resources and offers low-throughput and high-latency when compared to local clusters networks.

When Internet is chosen as geographically distributed clusters interconnection network, the collaboration difficulties are multiplied. The cooperation problems increment are a result of greater latency and throughput differences, Internet's variability, vulnerability to external attacks and the possibility of intermittent communication failures.

The Message Passing Interface (MPI) is currently the *de facto* standard for parallel applications communication in clusters [36][L4]. Numerous implementations of MPI have been carried out in industry and academia, and MPI has successfully been used in many domains (e.g., scientific computing, visualization, and bioinformatics).

Some of the MPI solutions for clusters (MPICH[48][L5], LANMPI [L3], …) do not scale properly to multi-clusters, leading to research on MPI implementations for distributed domains executions (PACX [44], MPICH-G2 [54][L6], MagPIe [55], MPICH-VMI

[63][L7], and StaMPI [51]). These solutions address the communication heterogeneity problem [54] but they lack Internet fault tolerance facilities.

The complexity of Internet based multi-cluster systems pushed solutions to use coarse-grained (job-level) parallelism [39], targeting the speedup through job scheduling techniques [12][24].

In a near future scenario, when the usage of remote processing power could be charged, it is important not just to reduce the application execution time but also to guarantee a certain level of efficiency in the resources usage. Assuring a threshold level of efficiency enhances parallel computers' throughput, improving the investment return.

The efficient execution of applications in heterogeneous clusters is not a trivial matter [21][56] and this complexity is enhanced in a Internet-based multi-cluster environment [52]. It has been shown that parallel applications written for a single cluster do not run efficiently without modifications on multi-cluster systems [13].

Through the past years several works were published focusing on the analysis of heterogeneous environments and their performance parameters. These works might focus on single heterogeneous clusters [14][15][22], scattered heterogeneous computers [23][70] or multi-clusters [1][52][62].

The work presented in this thesis investigates how to reduce the execution time of parallel applications based on the master-worker paradigm, originally conceived for a single cluster, through the efficient use of multi-cluster resources. Transparency is aimed at the application migration strategy to the multi-cluster system and the specific resources involved in the application execution (local and external clusters) should just be selected when their efficiency will be above a certain threshold. A system architecture is proposed to provide transparency, scalability and to surpass the inter-cluster communications problems. An analytical model, including the multi-cluster system and the application features, was developed to support the efficiency evaluation, providing the basis for a proposed performance prediction methodology and a guide for the system tuning actions. The methodology aims to guide to the analysis of the application and selection of the multi-cluster resources in order to guarantee the execution time reduction within a certain level of efficiency. This methodology also gives the support to the application tuning in order to increase the efficient speedup through the multi-cluster. The architecture, analytical model and methodology are the key components of a multi-cluster middleware for parallel applications. This middleware would provide efficient execution, prediction and dynamic adaptability.

1.1. Objectives

The use of multi-clusters can be a cheap, flexible and adaptable alternative to reduce applications' execution time. Nevertheless a multi-cluster is a complex environment whose heterogeneity challenges the collaborative execution of applications and their efficient speedup. The complexity grows when multi-clusters are formed by scattered clusters interconnected through Internet.

The work presented in this thesis targets to reduce the execution time of applications written for a single cluster, using a multi-cluster environment. It is also aimed that this execution speedup is achieved within a certain level of efficiency.

In order to achieve this goal, this work aims to provide a middleware, that allow a collection of clusters to form a multi-cluster; an architecture, to efficiently organize clusters resources and a methodology to guide in the process of achieving, using multi-clusters, an application speedup with a certain level of efficiency.

At the lower level a communication middleware is aimed to interconnect clusters through Internet providing reliability, overcoming possible temporary communication failures and efficiency, exploiting in a maximum Internet's peaks of throughput.

Once multi-cluster resources are interconnected, a system architecture is targeted to organize these resources in a way that applications' adaptation to multi-clusters is transparent, avoiding significant modifications in single-cluster original programs.

This architecture is aimed to organize multi-cluster resources in a way that applications execution is scalable, robust, efficient and adaptable. Scalability is a necessity because multi-cluster executions might occur with different amount of clusters and resources. The execution must also be robust to environment changes like Internet throughput and latency variances, delays or failures.

Efficiency is one of the main targets of this work. The resources organization aims to support strategies to increase the efficiency, like simultaneous computation and communication, and to overcome efficiency obstacles, such as load imbalance.

Once clusters are interconnected and organized, another objective of this work is to provide a methodology to guide in the process of efficiently executing an application in a multi-cluster with the goal of reducing the application execution time.

This methodology is oriented to determine, for a determined application if it should be profitable to be executed in the multi-cluster system, i.e., if it is possible to reduce the execution time with the coordinated use of the scattered resources. When a multi-cluster execution is profitable, the methodology targets to evaluate the execution time and efficiency of the application in the multi-cluster environment and to select which clusters and specific resources inside the clusters, should be involved in the execution to guarantee a defined efficiency.

Finally, this thesis also aims to provide some hints to help in the application tuning process in order to increase the application speedup and execution efficiency in the multi-cluster system.

1.2. Related Works

The objective of the work described in this thesis is to reduce the execution time of applications through the efficient use of multi-clusters. Generally a multi-cluster is a heterogeneous system on computation (each cluster itself might have different processing power and computers in each cluster might have different processors) and communication (intra-cluster networks have different level of performance than inter-clusters interconnection network).

The differences in latency and throughput from clusters local area network and the wide-area network used to interconnect clusters caused message passing libraries for parallel applications to have problems of scalability for multi-clusters. The scalability problems can be specially observed when collective communications are used.

Karonis in [53] demonstrates the advantages of topology aware message passing communication between scattered resources in a GRID environment. Topology aware MPI implementations like MagPIe [55] and MPICH-G2 [54] try to minimize the data traffic for collective operations in the slow link.

However, these approaches just avoid loss of performance of message passing communication through the wide-area network, not considering the effect of high-latency communication in the parallel algorithm performance.

Plaat at all in [66] show the performance degradation for six applications because of the different latency and bandwidth between intra and inter-communication networks. The paper describes ways to change the applications to explicitly use the multi-level structure.

In order to enable the achievement of speedup and efficiency in multi-clusters it is necessary for the application to be aware of the topology and performance at its workload distribution level.

Bal in [13] demonstrates that simple optimizations to take the multilevel network into account might enable high performance in a multi-cluster structure for parallel applications originally written for a cluster. Some authors [59][11] defend that the communication middleware should provide applications with mechanisms to query topology information.

Scalability is also an important issue for multi-cluster execution of applications. Aida in [1] and Nieuwpoort in [62] solved the scalability problem, achieving improvements in the application execution through hierarchical approaches applied to master-worker and divide-and-conquer applications.

Moscicki in [61] introduces a load-balance algorithm for the multi-cluster execution of divide-and-conquer applications. This algorithm adapts itself to inter-cluster network configurations and job granularities, not requiring manually tuned parameters.

Our proposal is to use a topology aware architecture to efficiently interconnect clusters based on a master-worker hierarchical approach to provide scalability. Our proposal also targets that these changes are not application specific like in Aida [1]. For our system transparency, not requiring considerable modifications to the application, is a target issue.

Another important goal of the thesis is to guarantee a minimum efficiency in the multi-cluster resources utilization. A performance model is the kind of tool we use to predict execution time and efficiency.

There are several on going works targeting efficiency of applications in heterogeneous clusters of computers. The master-worker paradigm is analyzed for HNOW considering just heterogeneous computation [21] or both heterogeneous computation and communication [4][5]. These works consider the distribution of one communication between master and worker and do not consider the possibility of having clusters interconnected through Internet. Our work takes into consideration master-worker application with multiple tasks that facilitates the balance of the load especially with heterogeneous processors.

Some works on GRID computing address the problem of having scattered resources interconnected through Internet for task-parallelism [23] or master-worker [70] applications.

Shao presents in [70] a performance model and a methodology to determine a performance-efficient mapping of master and worker processes in a GRID, considering the communication heterogeneity, but this model does not contemplate the possibility of interconnecting clusters instead of single computers.

Our multi-cluster approach congregates aspects of both heterogeneous cluster and GRID behaviors, given that master-worker execution is done in heterogeneous clusters and workload distribution between clusters is done through a low-throughput network such as Internet.

We propose then an analytical model also based in the computation-communication analysis but considering the cluster heterogeneity and the fact that clusters are connected through Internet. It is also relevant for our model possible performance improvements like overlapped computation and communication.

1.3. Contributions

In order to overcome multi-cluster problems, such as computation and communication heterogeneities, and reach a reduction of master-worker applications execution time while the resources (clusters/computers) are efficiently used, this thesis proposes a hierarchical master-worker system architecture, an analytical performance model and a performance prediction and system tuning methodology.

The proposed system architecture organizes the multi-cluster system in a way that it is possible to reach a high-performance efficient execution of applications. The system architecture is organized around a hierarchical master-worker with managers for handling Internet communication. The proposed system architecture provides the necessary single system image to the multi-cluster, transparently interconnecting clusters and overcoming Internet communication problems. The system architecture is also responsible to enable scalability and efficient and balanced usage of existent resources, allowing different data distribution strategies at intra and inter-clusters communication levels.

The performance model is based on the computation-communication analysis. This model has as inputs some computation and communication characteristics of the application and of the multi-cluster system. The analytical model evaluates the possibility of using scattered resources to speedup the application execution and predicts the efficiency in each cluster and in the multi-cluster, estimating also the application execution time.

The performance prediction and system tuning methodology describes the steps to migrate an application from a cluster to a multi-cluster environment, using the system architecture. The methodology uses the proposed analytical model to evaluate the efficiency and execution time and to dimension the execution resources in order to guarantee a fixed efficiency threshold. Additionally, if it is still necessary to improve the application speedup, the methodology also gives support to the process of tuning the application to a multi-cluster environment, helping in evaluating the necessary changes for efficiently reducing the execution time.

The proposed system architecture, analytical performance model and performance prediction and system tuning methodology have been applied to three applications to verify the quality of the predicted values for execution time and efficiency. The selected applications were the matrix multiplication, Stochastic Resonant Memory Storage Device simulation (SRMSD) and traveling salesman problem. The testbed multi-cluster included three scattered clusters located in Argentina, Brazil and Spain.

The obtained results proved the accuracy of the analytical model over 90% precision. Through the usage of the performance prediction and tuning methodology to the Stochastic Resonant Memory Storage Device application it was possible to achieve a speedup of 13 from a maximum speedup achievable of 13.7. The efficiency in the resources utilization was kept over 95%.

1.4. Thesis organization

This thesis is organized as follows. Chapter 2 presents the proposed system architecture used to interconnect the scattered heterogeneous clusters and create the multi-cluster system. Chapter 3 describes the proposed analytical model, explaining in details its concepts and elements. Chapter 4 introduces methodology for the performance prediction and system tuning, describing its phases and steps. The selected applications and the carried out experiments used to validate the proposed methodology and analytical model are described in Chapter 5. Finally Chapter 6 explains the thesis conclusion, summarizing its contributions and outlining possible future works.

Chapter 2

System Architecture

2.1. Introduction

Due to its complexity and heterogeneity, building efficient and interoperable multi-cluster systems is not that simple as plugging in and setting up clusters' Internet connection. This is especially true when considering finer grain applications on which continuous communication between clusters is a requirement. Simply interconnecting clusters proved to be a strategy that, for many applications, do not speedup appropriately [13], and might even cause decrease in the overall performance [41].

A multi-cluster has several heterogeneity aspects. The most relevant ones for our study concern computation and communication. At a multi-cluster of HNOWs, each cluster as a single element might have different processing power as well as each computer inside a cluster.

A similar situation happens with communication, once each cluster might have different local area network bandwidths and the communication between each pair of clusters might have different levels of latency, bandwidth, and even reliability. The network performance differences are especially relevant when clusters are interconnected through Internet. Figure 2-1 illustrates a multi-cluster and its heterogeneities.

In a multi-cluster there might be differences of 3 or 4 orders of magnitude in latency and bandwidth between clusters LAN and the wide-area network used to interconnect clusters. Using Internet as inter-cluster WAN magnifies these latency and bandwidth differences.

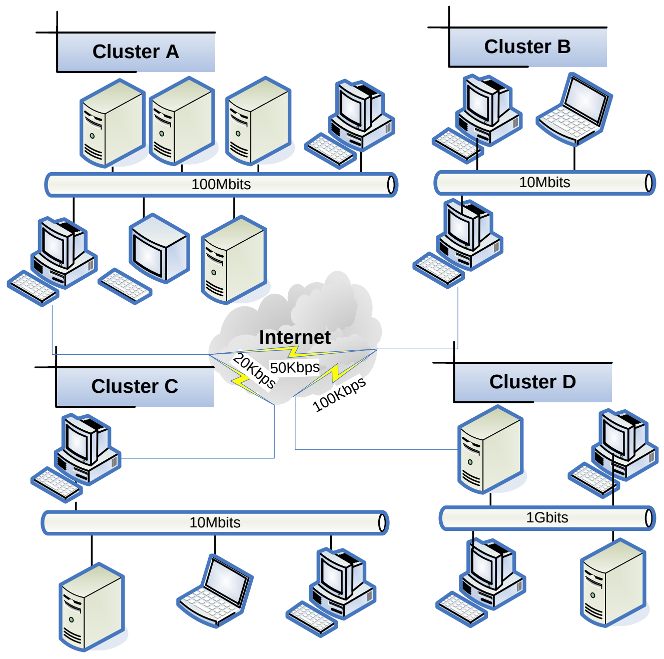


Figure 2-1: Heterogeneity in a multi-cluster formed by HNOW interconnected through Internet.

This heterogeneity between LAN and WAN made message passing solutions for a cluster (MPICH[48][L5], LANMPI [L3], …) not to scale properly to multi-clusters, suffering deterioration in communication performance, especially when collective communication is used [53] . The actual proposed solution for this problem is to turn MPI implementations aware of the topology as available in multi-cluster MPI implementations (PACX [44], MPICH-G2 [54][L6], MagPIe [55], MPICH-VMI [63][L7], and StaMPI [51]).

These multi-cluster implementations use the Internet Transmission Control Protocol (TCP) [67] for wide-area communication. Although, TCP is not well suited for Internet wide-area data transfers [33][37], reaching a fraction of the available bandwidth between the endpoints. Dickens [33] proposes a UDP based protocol to attain better communication performance. Other authors [3][71] approach this problem by dividing the data into partitions, opening several TCP sockets, and striping the data over several sockets.

An additional complication of Internet in wide-area communication, especially when long duration executions take place, is the possibility of temporary communication failures and consequent TCP disconnection. This problem evidences the necessity of a fault tolerance mechanism on inter-cluster communication. To the best of our knowledge, these communication fault tolerance mechanisms are currently inexistent in MPI implementations.

Another challenge on Internet usage is the high level of variation in latency and bandwidth along time. At Figure 2-2 it can be seen the Internet achieved throughput value along time between two computers, one located in Brazil and the other in Spain. The experiment had 46 hours duration and shows that the reached throughput varies from 8 to 30 KBytes/sec.

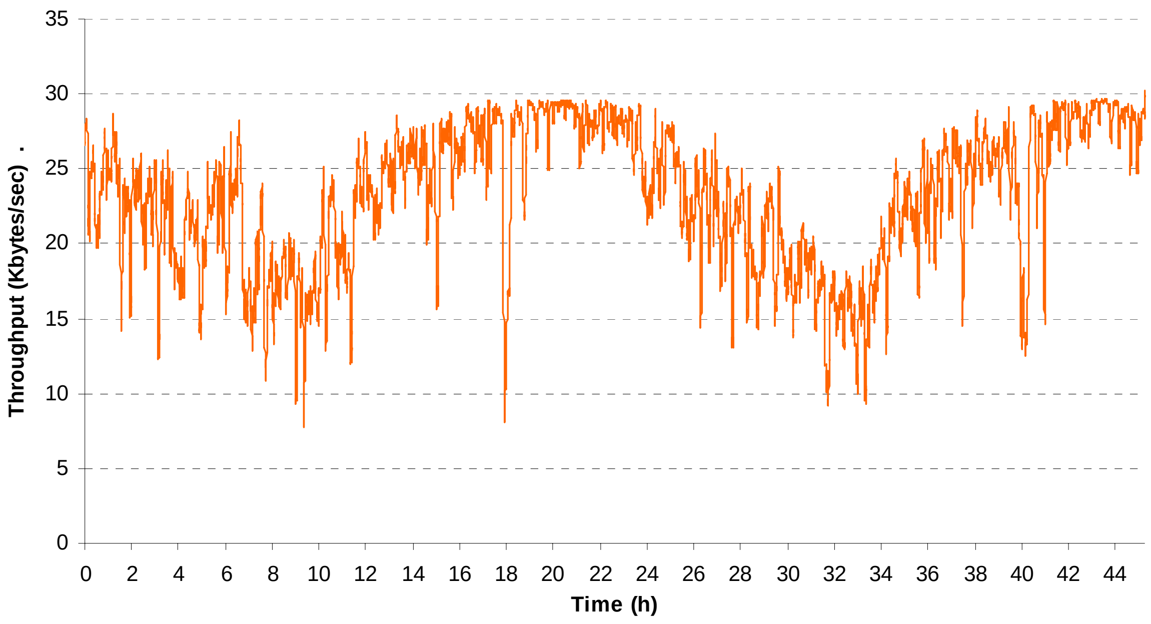


Figure 2-2: Internet throughput variance between one computer in Brazil and one computer in Spain.

A *system architecture* is proposed in order to support efficiency in multi-clusters applications execution [42][43][7]. This system architecture objectives to be: (i) scalable, allowing the collaboration between multiple clusters; (ii) robust, overcoming possible temporary communication failures in Internet and giving support to fault-tolerance schemes [72]; (iii) efficient, using strategies for exploiting Internet bandwidth in a maximum and to allow the application to use different strategies for both communication levels; (iv) adaptable, supporting the possibility of adapting the workload distribution to possible changes in the multi-cluster environment during the execution.

The proposed *system architecture* is organized in a way that the multi-cluster can be seen as a hierarchical master-worker, containing a component to manage wide-area communication: communication manager (CM). Hierarchical master-worker approaches were used by Aida [1] and Banino [16] to attain better levels of performance in multi-cluster and GRID respectively.

This chapter introduces the master-worker and hierarchical master-worker approaches, the system architecture functionality and components in more details.

2.2. Master-Worker Paradigm

On the way to parallelize an application, data and computational activities are partitioned into small tasks that can be executed simultaneously at the distinct processing elements. There are some paradigms for task and data decomposition. One of the most popular parallel programming paradigms is the master-worker.

The master-worker programming model is a fundamental and commonly used approach that consists of two entities: a master and multiple workers. The master is responsible for decomposing a problem into tasks and distributing these tasks among a farm of workers, as well as for gathering the partial results in order to produce the final computation result. The workers execute in a cycle: receive a message with the task (input), process the task, and send the result back to the master (output) (Figure 2-3).

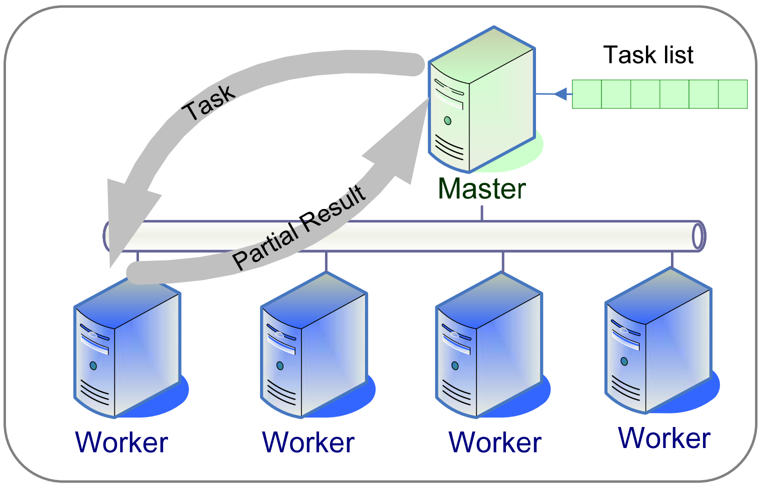


Figure 2-3: Master-worker paradigm.

Usually master-worker communication patterns are simple and well-defined, requiring communication only between the master and individual workers. Typically the master sends one or more tasks to each worker. Each time a worker finishes, it sends the result back to the master and the master sends to this worker a new task. This process is repeated until all tasks have been completed [25]. This characteristic turns the master-worker model to be easily adaptable to heterogeneous systems.

Goux [45] claims that the master-worker paradigm is suited for dynamic and heterogeneous environments that require programmability, adaptability, reliability and efficiency. These features grounded the creation of some master-worker based frameworks [46][29][61].

The master-worker (MW) model is used in many scientific, engineering and commercial applications, such as: software building and testing, sensitivity analysis, parameter space exploration, image and movie rendering, high-energy physics event reconstruction, the processing of optical DNA sequencing, neural networks training and stochastic optimization among others [18][27][76][L8].

The most important reasons to adopt the master-worker model in this work are:

- Adaptability to a changing number of machines

  The master-worker paradigm turns applications to become easily adaptable to executions with different numbers of computers. With a dynamic task management and the capacity to react to machine losses throughout execution time, the master-worker implementation is also adaptable to a dynamic changing number of machines during execution without major problems.

- Existence of a central element

  The existence of a central control element that distributes the work is quite important for such a heterogeneous and unpredictable system like heterogeneous multi-clusters. The master element can dynamically adjust the amount of work to each worker and to other connected clusters, depending on the variance of the execution parameters like network throughput or external clusters availability for example.

- High degree of predictability

  Empirical evidence has shown that, for a range of applications, the execution of each task in successive iterations tends to behave similar, so that the measurements taken for a particular iteration are input predictors of near-future behavior [68]. This is important for designing models on which predictions could support dynamic decisions.

- Efficiency

With the standard distribution approach, workers will be idle during the communication period. In order to avoid this waste of computational power a pipeline strategy might be employed at the master-worker implementation. Workers processes might have two buffers and two threads: one for communication and another for computation. In the meantime a result is being sent and a new task is being received. The worker will be computing the other task buffer during the communication process.

- Scalability

Applying to a collection of clusters, the master-worker architecture also provides a number of hierarchical models in order to gather data and optimize distribution over non-first-level clusters or sub-clusters. The master-worker model itself is hierarchical, and it can also be expanded into more complex hierarchical relations. In Figure 2-4 the system architecture is extended into different hierarchical levels. The possibilities of expanding this model are unlimited [1][16].

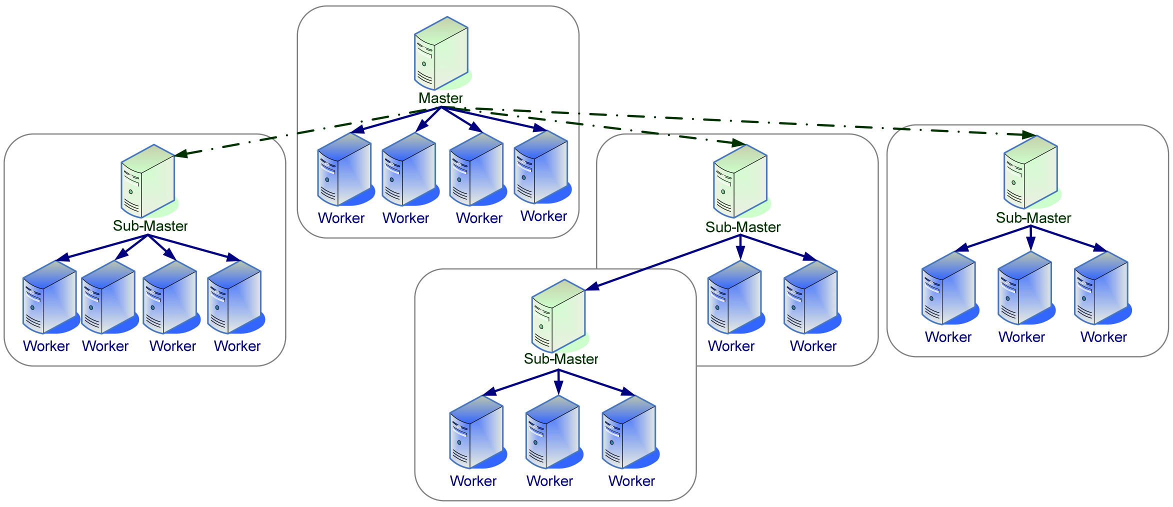


Figure 2-4: Hierarchical master-worker possibilities.

The connection between a master and another level master (sub-master) usually can be different (i.e. in latency and bandwidth) from the one that interconnects a master to a worker. The master and workers can be in the same LAN, the sub-master and its sub-workers in another LAN, and the connection between them can be a public WAN.

Therefore, the master-worker paradigm itself facilitates centralization of system management by using a central point that function as a global master, managing all communications with others clusters, monitoring network behavior, controlling task execution and also reallocating data, when necessary. One mentionable advantage of this architecture is that the master does not have to be always tied to one specific cluster.

2.3. Proposed System Architecture

The proposed system architecture is organized in a way that the multi-cluster can be seen as a hierarchical master-worker, containing an element to manage WAN communication: communication manager (CM).

In this architecture each cluster is a master-worker itself. The cluster that contains the main master on which all data resides and from which the application execution starts is considered the local cluster or main cluster. All other external clusters are also called sub-clusters, its masters, sub-masters and its workers, sub-workers. Communication managers are responsible for each pair of clusters communication through Internet (Figure 2-5).

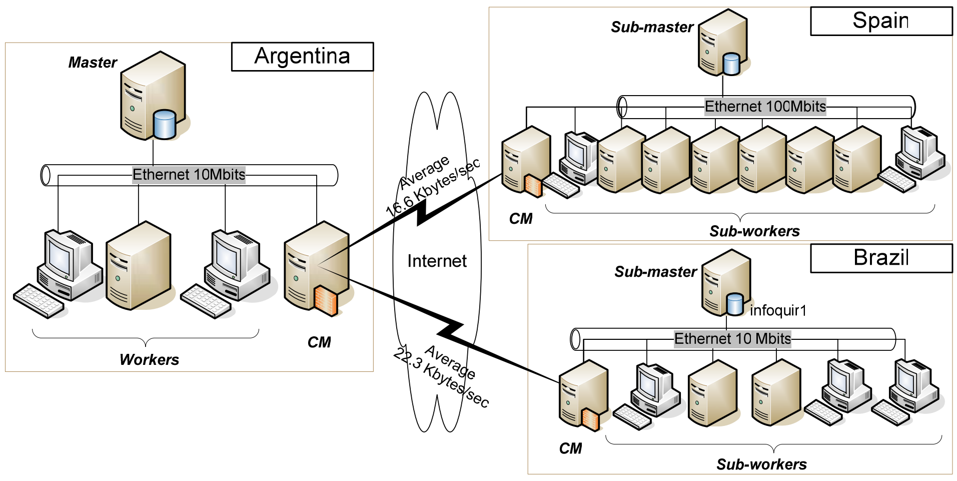


Figure 2-5: Multi-cluster system architecture as a hierarchical master-worker.

In the way it is organized, the System Architecture presents four distinct processes roles: master, worker, sub-master, and communication manager.

In order to improve the performance that a remote-cluster can attain to the system, the master might send to sub-masters tasks with coarser grains when compared to tasks sent to its local workers. There is no direct communication between master and external cluster's workers. To reach a balance of the load despite workers heterogeneity tasks are distributed in on-demand scheme.

The master is the element that has all the data and from which all distributions begins. Masters are responsible for division of the problem into several tasks of a certain granularity and for distribution of these tasks to its workers and to sub-masters at external clusters.

A pipeline strategy is implemented between clusters through the use of sub-masters and communication managers. Sub-masters implement a buffer/cache strategy so that data already received is recorded for further use, depending on the application characteristics. This feature allows an increase on the communication-computation ratio, avoiding communication of already sent data through the slow WAN link.

Sub-masters have the same basic functionalities of a master, except by the fact that its workload is received by the main master and represents part of the global problem. Sub-masters can receive bigger granularity tasks and then distribute this workload in finer grain tasks to its sub-workers. The same process is done with workers results, that can be joined into bigger grains results before being sent back to the main master.

Workers are responsible for labor function. Workers receive a task data, execute the task and answer to the master connected to its LAN the result data. To reach performance improvements a pipeline strategy is implemented at workers level. Each worker has two buffers and executes computation and communication functions simultaneously through the use of two threads.

Communication managers play a very important architectural role having different functionalities. For the main cluster master a CM represents a worker with greater computational power (corresponding to the connected sub-cluster) and high latency both on receiving tasks and returning partial results.

For the application, communication managers' worker behavior adds transparency to the system, allowing unaltered applications to work in a multi-cluster. For the external cluster, communication managers feed the sub-master with the workload to be split through its workers. For the system architecture, communication managers are necessary elements with functionalities that provide efficiency and robustness.

It is important to remark that the four processing roles existent in the system architecture are just related to execution processes and not to clusters computers. A workstation can be concurrently executing a master, communication manager and several workers processes.

Although, these processes have different resources needs. Masters are generally not CPU bound but require high principal and secondary memory capacity and IO throughput. Workers are CPU oriented. Communication managers need memory and preferentially two different network cards so that Internet traffic do not interfere in the LAN and local and remote communication can be done simultaneously.

In a heterogeneous environment, processes can be assigned each one to a specific workstation which can afford its characteristics. It might not be a good strategy to have two workers processes competing for one CPU, or even to have a worker interfering in the master capacity of distribution. High performance computers should be selected as workers while intermediate computers can have their memory and storage devices improved to play master roles. A brief description of the system architecture processes types is found in Table 2-1.

Table 2-1: System architecture processes roles.

Role	Needs	Responsibility
Master	Memory and IO.	Data distribution policies for workers and sub-masters. Storage of problem and result data.
Worker	CPU	Pipeline with tasks execution in a multithreaded scheme with double buffering.
Sub-master	Memory and IO	Same as master. Buffering of its dataset.
Communication Manager	LAN and WAN access.	LAN and WAN isolation. Add reliability to WAN communication. Exploit WAN low throughput in a multi-connections multithreaded organization.

2.4. Communication Managers

Communication managers proved to be necessary to guarantee to the multi-cluster system *transparency*, *efficiency*, *robustness* and *security*.

Transparency is achieved because for the master CMs act as any worker, receiving tasks and returning tasks results. The difference is that this worker has the processing power of an external cluster with a high latency on tasks and results communication. For a sub-master CMs act as the storage system where this master obtain the tasks to be distributed among its workers.

Communication managers are bridge elements that receive data from local MPI calls, store these data in a buffer and continuously send them to the remote cluster. This communication becomes again local MPI calls at the other cluster side. The CM existence takes out from the master the responsibility of dealing with Internet latency and bandwidth unpredictability and with possible Internet communication failures.

In order to provide *efficiency* to the system architecture, communication managers act as a bridge isolating traffic between local and wide-area networks. Being present at the LAN, CMs prevent the master to delay the tasks distribution to its local workers because of the long delay on sending tasks to external clusters through the low throughput WAN. Experiments without the use of CMs show the loss of local cluster performance, once the master stands long busy communicating with remote computers causing local workers idle time [41].

For *efficiency* reasons, CM also applies strategies of buffering, threading for local and wide-area simultaneous communication, and stripping for Internet communication. These strategies were implemented through the design and development of a communication library for the wide-area network.

The communication library is called Long-Distance Service (LDS) and is explained in more details at the Appendix A of this thesis. CMs use standard MPI for intra-cluster and LDS for inter-cluster communication.

At LDS initialization, buffers are allocated for sending and receiving communication. Communication managers continuously requests new tasks to the master until LDS sending buffer is full. With this strategy, CM tries to guarantee a constant communication between clusters so that the available throughput can be exploited in a maximum.

In order to maximize the communication performance LDS apply a network stripping strategy consisting of dividing the data into partitions, opening several sockets and striping the data over the sockets [3][71].

Figure 2-6 shows the obtained throughput along time for communications between Brazil and Spain using LDS library configured for different amount of sockets. It can be seen that the communication improves its throughput almost linearly until 6 simultaneous communications through sockets are used. Then the increase in the amount of sockets does not change the obtained throughput meaning that the available throughput was reached.

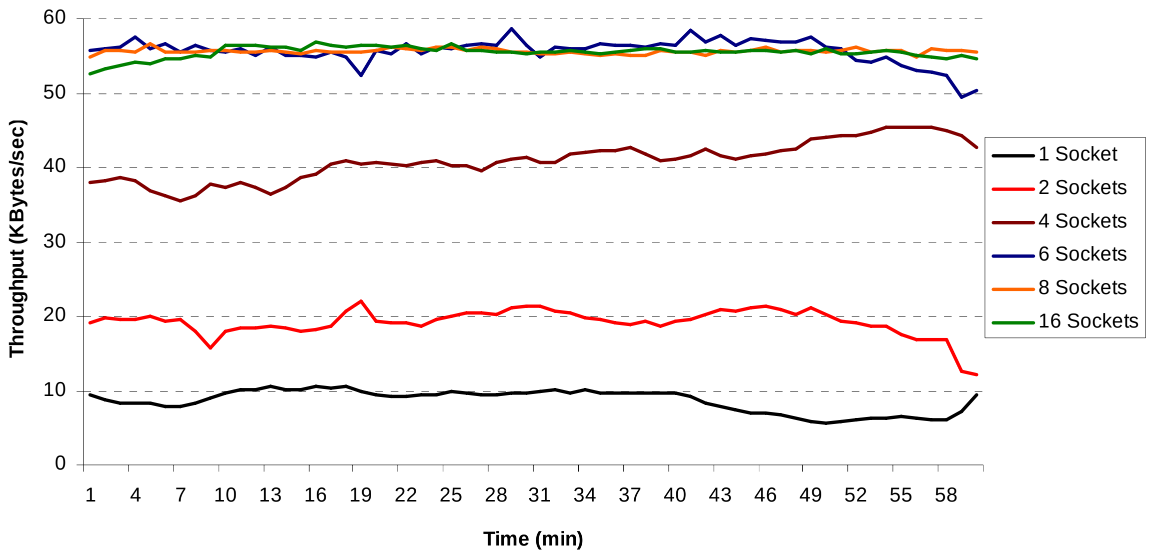


Figure 2-6: Experimental throughput between Brazil and Spain using LDS with different amount of simultaneous communication.

Robustness is achieved at Internet communication because LDS also implements a fault tolerance strategy handling possible TCP disconnections and guaranteeing the arrival of the data.

Although CM was not designed with *security* concerns, it is possible to attain *security* because only one gateway is required to the cluster interconnection to the Internet and the communication ports of LDS can be easily configured. In the gateway computer firewalls can be used guarding the cluster from attacks.

It is important to remark that the CM is a logical concept that physically is represented by a process which can be running on any computer, even concurring with a master or worker. Despite that, because of the networks isolation, it is highly recommended to run CM process in a workstation that has two network cards: one in the LAN and the other in the WAN.

Internally, the communication manager's architecture is organized around two threads to make the bridge of MPI sending and receiving operation to LDS. LDS by its turn is a transparent multithreaded buffered system that allows reliable inter-cluster communication. The CM architecture can be seen in Figure 2-7. The mutex semaphores are necessary to avoid two simultaneous MPI calls that might result in MPICH to halt.

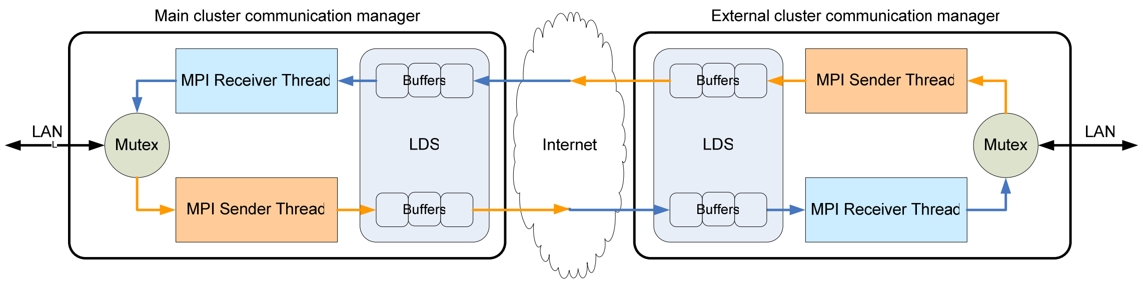


Figure 2-7: Communication manager architecture.

2.5. Adapting to the Proposed System Architecture

In order to have a better overview of the system and to guide the adaptation of applications to the proposed system architecture, a layered model is introduced. This model is formed by five layers from the application to the physical layer (Figure 2-8).

At the application layer there is the application written under a programming paradigm. The system architecture aims transparency and the application should be unaltered on its main basic functionalities concerned with the solution of the problem

At the programming paradigm layer the system architecture uses the hierarchical master-worker paradigm. It is interesting to remark that it is possible to map different paradigms to the master-worker and use the proposed system architecture as run-time architecture for applications, extending the range of applications that can be passed transparently to a multi-cluster.

In order to execute in a multi-cluster, the only necessary change at this layer in the application is to enable the master, when acting as a sub-master, to have the communication-manager computer as its input/output subsystem.

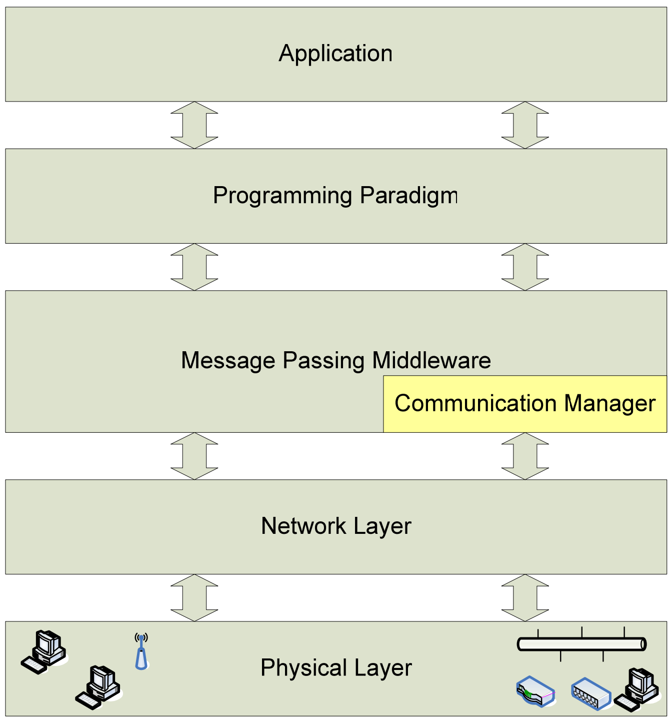


Figure 2-8: Layered model.

The message passing middleware is the layer responsible to the communication between multi-cluster resources. At this layer it is necessary to add the communication manager. The communication manager is a worker and inserting it consists on selecting a specific worker to act as communication manager being a bridge between MPI and LDS.

In the network layer there are the standard communication protocols while the physical layer corresponds to the existent physical components like computers, network cards, switches, routers, etc.

With the objective of improving efficiency some other modifications can be performed at the programming paradigm layer. In order to implement the intra-cluster pipeline, reducing idle time of workers, it is necessary that workers have two buffers and use asynchronous communication or separate threads for overlapping computation and communication.

Another possible change that can result in performance improvements is to send tasks with a different granularity to the worker representing the communication manager, aiming a better computation-communication relation for the high-latency external clusters.

2.6. Conclusions

In this chapter the proposed system architecture was described. The system architecture main objective is to support multi-cluster execution of applications providing scalability, robustness, efficiency and adaptability. It is also a goal of the proposed system architecture to enable applications to be adapted transparently to a multi-cluster environment.

The proposed system architecture is organized in a way that the multi-cluster can be seen as a hierarchical master-worker. A master is responsible for tasks distribution in each cluster. The cluster that contains the problem data is responsible for distributing tasks to the external clusters masters, as well as to its own workers.

Communication managers were added to the system architecture to provide transparency for the adaptation of applications, robustness and efficiency to the inter-cluster communication.

Transparency is reached because communication manager encapsulates in a local cluster normal worker the complexity and performance of external clusters. To the main master a communication manager is a worker with greater performance and high latency on responding to tasks. For external clusters masters, communication managers are the input/output subsystem for the problem tasks and results.

Efficiency and robustness at Internet communication between clusters is achieved through a developed communication library called LDS. The LDS communication service uses buffering strategy for asynchronous sending and receiving messages, targeting to keep constant communication, exploiting Internet in a maximum. A network stripping strategy aims to obtain the maximum throughput out of Internet's transmission protocol. Fault tolerance is implemented to transparently solve problems of possible communication disconnections.

Chapter 3

System Performance Model

3.1. Introduction

At Chapter 2 the proposed system architecture was presented. The system architecture organizes a multi-cluster in a way that scattered clusters can be used for an application execution. The target of this thesis is to reduce the execution time of applications through the use of distributed clusters and to achieve this speedup guaranteeing a certain level of efficiency in resources utilization.

It is also important to have a reliable estimation of the amount of time local and external resources will be in use because of the fact that resources might be in different domains, belonging do different organizations.

A performance model is then necessary to estimate execution time and efficiency of master-worker applications. Different authors are working on performance modeling for master-worker applications with heterogeneous resources. These models are centered in communication and computation analysis considering single heterogeneous clusters [4][21] or scattered resources obtained from a GRID allocation [15][70].

Our multi-cluster approach congregates aspects of both mentioned behaviors, given that master-worker execution is done in heterogeneous clusters and workload distribution between clusters is done through a low-throughput network such as Internet.

In order to improve efficiency, avoiding workers idle computation time, the architecture defined a pipeline strategy where communication and computation occurs simultaneously in both intra and inter cluster levels. The execution performance with this strategy is either limited by workers computation (computation-bounded) or by network communication (communication-bounded) capacities.

The master-worker pipelined execution has two stages (computation and communication) and can be divided in three different states: *startup*, *steady* and *end*. The *startup* is the state comprehended between the execution start and the moment when all workers compute tasks. At this moment the system reaches its *steady* state. At the *steady* state the best execution performance is achieved. The *steady* state finishes when a worker becomes idle because the master has no more tasks to distribute. From this moment until the master receives the last task takes place the *end* state.

Figure 3-1 illustrates the performance behavior along time for a pipeline execution. It is possible to distinguish in this graphic the *startup*, *steady* and *end* states. Even when the *steady* state is computation bounded, the overall efficiency is reduced because of *startup* and *end* states.

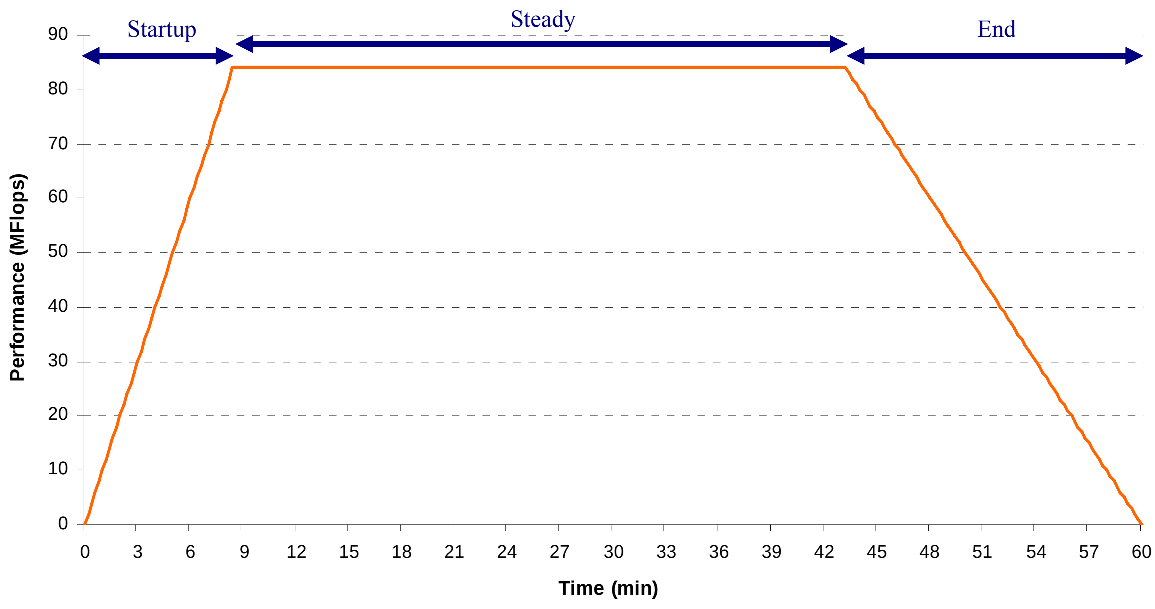


Figure 3-1: Sample graphic of performance for a pipelined execution, showing the different states.

An *analytical performance model* is then proposed [9][8] to evaluate these three states and estimate efficiency and execution time of an application at a multi-cluster system organized as defined in the proposed system architecture. The main objective of the model is to evaluate the possible performance that each cluster can attain in the multi-cluster execution, estimating the multi-cluster application execution time and the achievable efficiency.

The *performance model* is grounded in a computation-communication analysis. This model has as inputs the application computation and communication requirements, the multi-cluster system computation and communication capacity and the target efficiency.

It is also an aim of the model to help the selection of resources for which the execution carries out a desired threshold level of efficiency. This resources selection determines which clusters and computers can be used guaranteeing efficiency over the threshold.

In this chapter the proposed analytical model parameters and design for each state are presented in more details.

3.2. Multi-cluster Input Parameters

The analytical model proposed in this thesis is based in computation-communication analysis. The model communication and computation parameters can be divided in two basic types: application and system.

Application parameters are those that depend exclusively on an application, i.e., parameters that are not modified when the target execution resources is changed. Examples of application parameters are the amount of bytes of a task and the amount of tasks of a certain problem.

System parameters are the ones that depend on the target multi-cluster system where an application execution is planned to happen. The number of nodes, the performance of each node and network throughputs are examples of system parameters.

Application parameters need to be analyzed just once for a certain application and just need to be reevaluated if the application implementation has changed. System parameters need to be updated every time an execution will take place with different resources, i.e., within a different multi-cluster configuration.

3.2.1. Application Parameters

Application parameters refer to the application computation and communication requirements for the solution of a problem. Analyzing computation, **application parameters** are associated to the problem division into tasks and tasks complexity.

On the communication side, **application parameters** are related to the volume of communication required for communicating tasks and their results.

Before analyzing the computational application parameters it is important to determine the application *basic operation*. The *basic operation* of an application is the metric of computation performance that is used to compare resources performances and to predict execution time. The *basic operation* represents the most important and common operation executed in an application task, the operation that computationally limits the execution time. For example, a possible *basic operation* for a matrix multiplication algorithm could be a single float point operation. It is possible to determine the amount of float point operations of a matrix multiplication and also the float point operations are the most important operation in a matrix multiplication task.

For some applications it might not be straightforward to determine such a simple operation as the basic one and the *basic operation* might be a Fourier transform or a group of calculations for example. If no important repeatable operation is evident for an application, an application task itself can be considered as *basic operation*.

Computation

The model computational **application parameters** for an algorithm are the amount of *tasks of a certain workload*, the amount of *basic operations of a task* and the amount of *basic operations of a workload*.

The problem or a significant part of the problem is considered to be a workload. The amount of *tasks of a workload* (*Tasks*) depends on how an application divides a workload into single tasks that are sent to workers. For some applications there might be several ways to divide a workload into tasks, resulting in different amount of tasks depending on the granularity. In this case, the amount of *tasks of a workload* is a function of some algorithm parameters that define this granularity.

The amount of *basic operations of a task* (*Oper*) also depends on the way tasks are divided in an application and it represents how many single basic operations needs to be done for a task completion.

The amount of *basic operations of a workload* (*Workload*) for a certain granularity can be seen as the multiplication between the amount of *tasks of a workload* and the amount of *operations in a task* (1) and represents how many single basic operations needs to be done for the workload completion.

$$Workload(G) = Tasks(G) * Oper(G) \tag{1}$$

Table 3-1 summarizes the **application parameters** for computation, with their symbol and explanation.

Table 3-1: Model computation application parameters for a multi-cluster.

Computation Parameter	Symbol	Explanation
Tasks of a workload	Tasks(G)	Represents the quantity of tasks of a workload divided in a specific granularity.
Basic operations of a task	Oper(G)	Represents the number of basic operations of a task with a specific granularity.
Basic operations of a workload	Workload(G)	Represents the number of basic operations of a workload divided in a specific granularity. It is the number of tasks for this workload multiplied by the operations of a task.

Communication

For the communication, the **application parameters** are related with the data communication volume of tasks and results. The communication application parameters are the *size of a task* (S_{Task}), the *size of a task result* (S_{Result}) and the *total communication of a task* (S_{Comm}).

The *size of a task* is the amount of bytes that needs to be communicated from master to worker in order to send an application task. Similarly, the *size of a result* is the amount of bytes that needs to be communicated from worker to master in order to send a task result. Both the *size of a task* and the *size of a result* depend on the granularity on which the tasks were divided.

The *total communication of a task* (2) is the sum of the *size of a task* and the *size of a result*.

$$S_{Comm} = S_{Task} + S_{Result} \tag{2}$$

Table 3-2 summarizes the **application parameters** for communication, with their symbol and explanation.

Table 3-2: Model communication application parameters for a multi-cluster.

Communication Parameter	Symbol	Explanation
Size of a task	$S_{Task}(G)$	Represents the amount of bytes that needs to be communicated for sending a task from master to worker with a certain granularity.
Size of a result	$S_{Result}(G)$	Represents the amount of bytes that needs to be communicated for sending the result of a task from worker to master with a certain granularity.
Total communication of a task	$S_{Comm}(G)$	Represent the total amount of bytes needed to be communicated between master and worker for a task completion. It is the sum of the size of a task and the size of a result.

3.2.2. System Parameters

System parameters refer to the multi-cluster system computation and communication capacity. Analyzing computation, **system parameters** are associated to the capacity of workers on computing tasks. For communication, **system parameters** are related to clusters networks throughputs and inter-cluster throughputs.

Computation

For the computation, the **system parameters** are each computer *available performance* (*$Perf_{Avail}$*), each *cluster available performance* (*$CPerf_{Avail}$*) and the *multi-cluster available performance (*$MCPerf_{Avail}$*)*.

The *available performance* of a computer is defined as the amount of basic operations per second a computer can reach executing application tasks. This measurement might also depend on the task division granularity. This value is reached by executing in each computer stand alone application tasks of the desired granularity.

For a specific execution, the sum of workers available performance in a cluster represents the *cluster available performance* (*$CPerf_{Avail}$*) (3). Analogously, the addition of the *cluster available performances* for all clusters in the multi-cluster execution represents the *multi-cluster available performance* (*$MCPerf_{Avail}$*) (4). These indexes represent an approximation to the best possible performance of the system using the selected workers.

$$CPerf_{Avail} = \sum_{workers} Perf_{Avail} \tag{3}$$

$$MCPerf_{Avail} = \sum_{clusters} CPerf_{Avail} \tag{4}$$

A summary of the dynamic computational parameters can be seen at Table 3-3.

Table 3-3: Model dynamic computation parameters for a multi-cluster.

Computation Parameter	Symbol	Explanation
Available Performance	$Perf_{Avail}(G)$	Represents the amount of basic tasks per second a computer is able to execute using tasks of a certain granularity.
Cluster Available Performance	$CPerf_{Avail}(G)$	Represents the sum of the available performance for the computers which have worker role.
Multi-cluster Available Performance	$MCPerf_{Avail}(G)$	Represents the sum of the cluster available performance for the clusters selected to the application execution.

Communication

The **system parameters** for communication are each cluster *local area network throughput* (*$TPut_{LAN}$*) and the average *Internet throughput* (*$TPut_{Inet}$*) between the main cluster and each external cluster in the multi-cluster.

Local area networks are dedicated environments for which the nominal throughput is known. Despite that, the real useful throughput of a LAN can vary with the size of the data being transmitted and its relation with the size of the total header data for all the protocol layers used.

Because of this variation possibility, it is recommended to have the *local area network throughput* measured as the average throughput using a program that continuously communicates application task-sized and result-sized packets. This value might be dependent of the task granularity.

Internet throughput between clusters is not stable and can significantly vary along time. For this work, the *Internet throughput* is an average value reached through a long duration communication experiment between clusters. It might be possible to have different values for incoming and outgoing *Internet throughputs*. Because of this both communications throughputs should be experimentally determined.

A summary of the dynamic communication parameters can be seen at Table 3-4.

Table 3-4: Model dynamic communication parameters for a multi-cluster.

Communication Parameter	Symbol	Explanation
Local area network throughput	$TPut_{LAN}$	Represents the average real throughput of the local area network communication task and results sized packets of a certain granularity.
Internet throughput	$TPut_{Inet}$	Represents the average throughput of the Internet between the main cluster and a specific external cluster.

3.3. Performance model

The performance model is based in the computation-communication analysis for the architecture pipelined approach. As defined in the proposed system architecture, during an application execution computation and communication run simultaneously using double-buffered multi-threaded workers for intra-cluster level and using sub-masters and communication managers for the inter-cluster level.

The aim of the performance model is to evaluate the execution time and efficiency for an application execution in a specific multi-cluster. This evaluation is done analyzing the different states for an application execution.

A master-worker pipelined execution has two stages: computation and communication. The pipeline execution time can be divided in three different states: *startup*, *steady* and *end*.

The *startup* is the state comprehended between the execution start and the moment when the pipeline is full and the system reaches its best execution performance. The *startup* is the period of time where each worker first task is communicated.

The *steady* state is the period where the execution is stabilized in its maximum and steady performance, where the best computation-communication ratio is reached. The *steady* stage finishes when there are no more tasks to be distributed and workers start to become idle. From the moment the first worker ends its execution until the last task is received by the master represents the *end* state.

Figure 3-2 illustrates a double buffered pipelined execution of 20 tasks in a cluster with 4 heterogeneous workers. This figure graphically shows each execution state and their respective duration: *startup time* (T_{Up}), *steady time* (T_{Ste}) and *end time* (T_{End}).

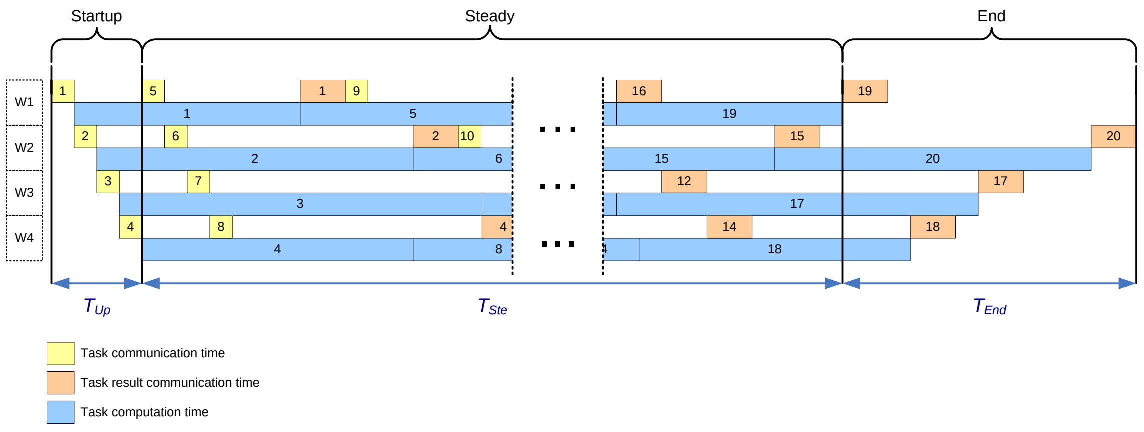


Figure 3-2: Illustration of the execution time states for a sample execution of 20 tasks in a cluster with 4 workers.

In order to evaluate the efficiency it is important to analyze the three states. At the *startup* and *end* states there is a loss in efficiency caused by idle computation time of workers while idle computation can be found in the *steady* state when the application is communication bounded.

The proposed model estimates the duration of each of these three states. The estimated *execution time* (T_{Ex}) (5) is then the sum of *startup time* (T_{Up}), *steady time* (T_{Ste}) and *end time* (T_{End}).

$$T_{Ex} = T_{Up} + T_{Ste} + T_{End} \tag{5}$$

The best possible execution time (T_{Best}) would occur when all the tasks are executed in computation bounded *steady* state. In this case the *startup* and *end* time would be null and all tasks would be executed in a perfect pipeline, using the whole *multi-cluster available performance* (4). The *best execution time* would then be the ratio between the amount of *basic operations of the workload* and the *multi-cluster available performance* (6).

$$T_{Best} = \frac{Workload}{MCPerf_{Avail}} \quad \textbf{(6)}$$

The estimated *efficiency* can be defined as the ratio between the *best possible execution time* and the estimated *execution time* (7).

$$Efficiency = \frac{T_{best}}{T_{ex}} = \frac{T_{best}}{T_{Up} + T_{Ste} + T_{End}} \quad \textbf{(7)}$$

In the following sub-sections we study the possible best and worst time for each state of the pipeline execution (startup, steady, end), so that the overall best and worst execution time and efficiency can be estimated. This study is initially done for a local cluster and then extended to an external cluster.

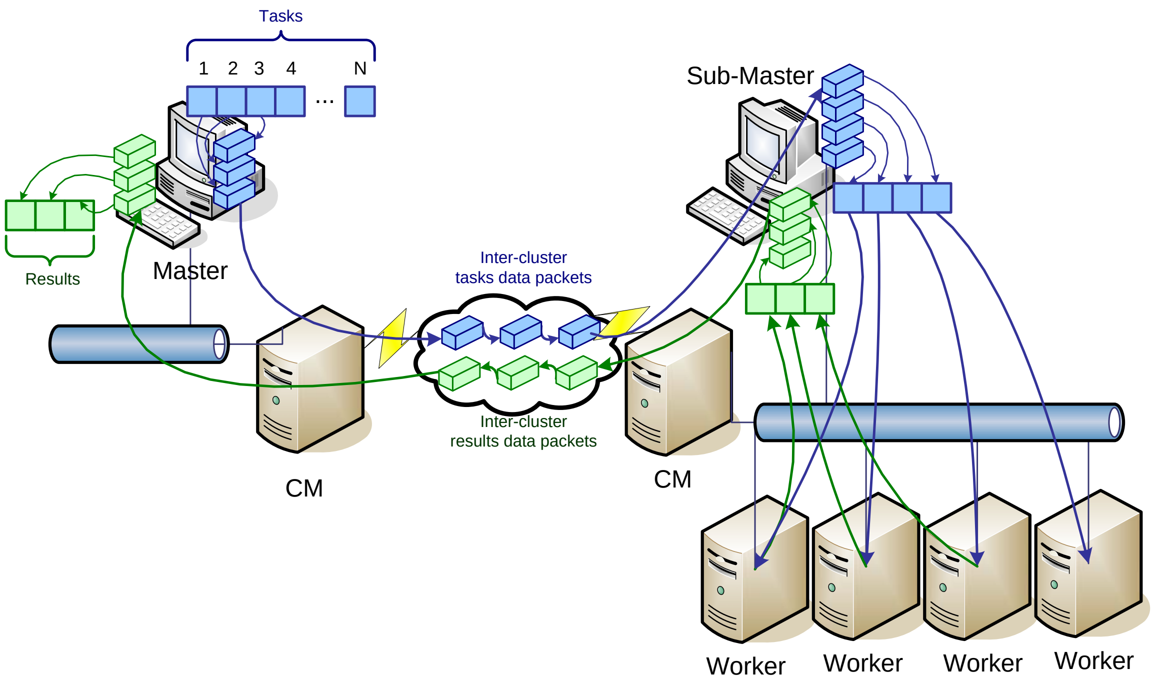


Figure 3-3: Illustration of one-to-one inter-cluster studied distribution strategy, where one inter-cluster task corresponds to one task inside the external cluster and so does one inter-cluster task result.

For external clusters the startup, steady and end times are dependent on the inter-cluster tasks distribution strategy. This strategy might be specifically oriented for the application. On the following sub-sections we study two possible distribution strategies for inter-cluster tasks distribution: one-to-one and one-to-many.

At one-to-one distribution strategy, one inter-cluster task data sent to an external cluster represents one task for external cluster workers. For this task one result will be sent back from external to main cluster (Figure 3-3).

At one-to-many distribution strategy, one task data packet sent to an external cluster represents an N fixed number of tasks locally in the external cluster. An N fixed number of results will be joined together in an inter-cluster task result (Figure 3-4).

The one-to-many approach is profitable when it is possible to increase the computation-communication ratio of an application by the means of increasing the task volume of data.

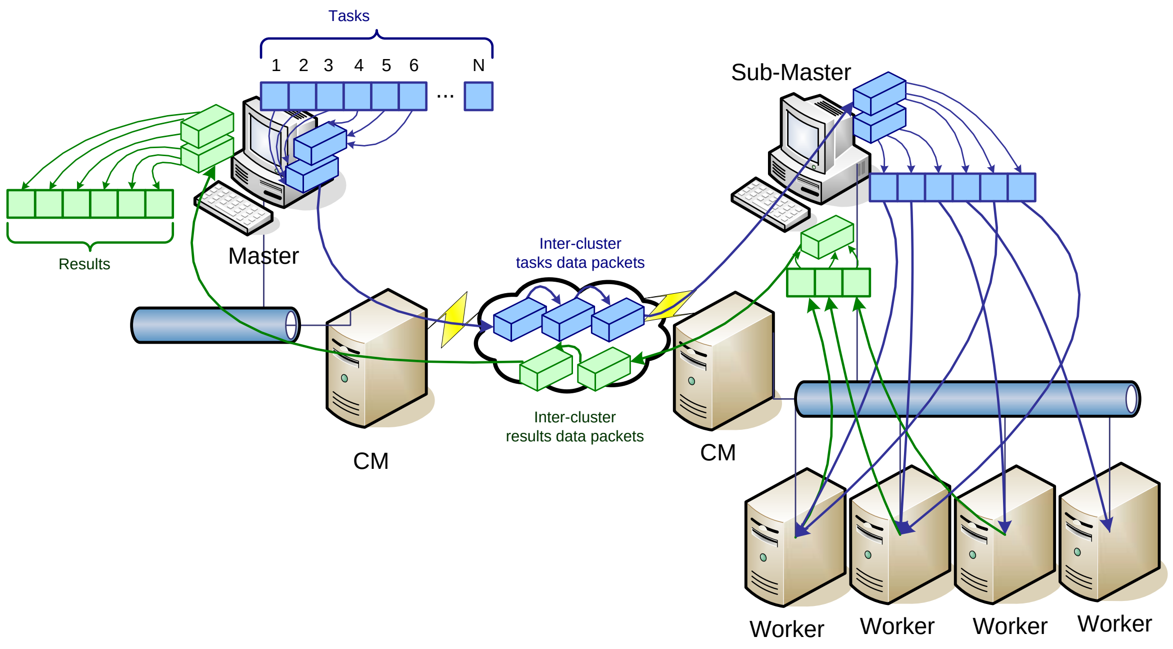


Figure 3-4: Illustration of one-to-many inter-cluster distribution strategy, where one inter-cluster task corresponds to N tasks inside the external cluster and one inter-cluster result corresponds to the joint of N external clusters results.

3.3.1. Startup Time

The *startup time* is the elapsed time between the execution start and the moment the pipeline is full and steady. At the beginning of the startup state, all workers are idle waiting for tasks. During this state the first task is distributed to workers so that gradually the system improves the workers usage until there are no idle workers. At this moment the startup state reaches its end.

At the next sub-sections the *startup* state is studied for a local cluster and for external clusters with both proposed distribution strategies.

Local cluster

The time spent by the master to communicate one task for the first worker in the local cluster (Tc_{TaskLC}) is the startup time for this worker. The second worker waits the time that it takes for the master to send two tasks (one for the first worker and one for it). For each worker the *startup time* grows successively as shown in Figure 3-5.

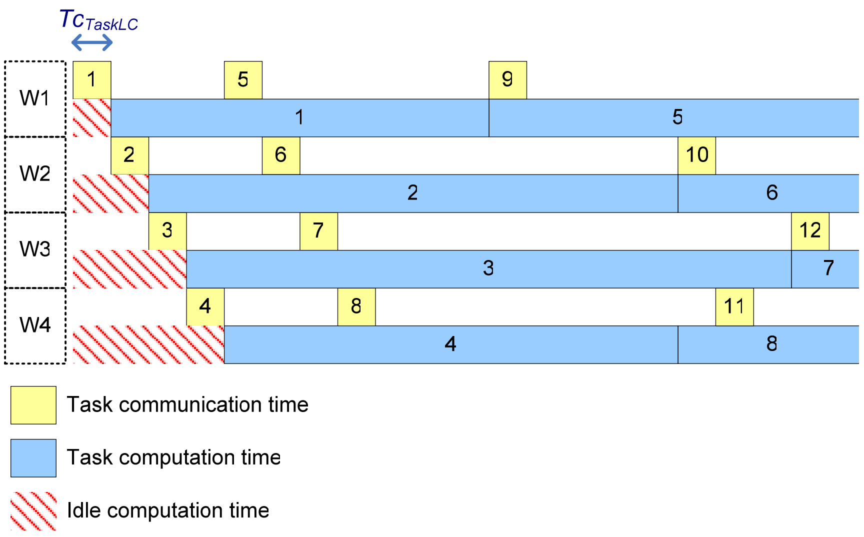


Figure 3-5: Startup time for each worker in a local cluster execution.

The total computation idle time (Tp_{Idle}) for a W workers cluster startup is then the sum of these idle times (8) which represent an arithmetic progression (9). The whole system is penalized in the average of the total idle time, which is reached by dividing the idle time by the amount of workers. The *startup time* for a local cluster (T_{UpLC}) is then given by equation (10).

$$Tp_{Idle} = Tc_{TaskLC} + 2 * Tc_{TaskLC} + \ldots + W * Tc_{TaskLC} \tag{8}$$

$$Tp_{Idle} = Tc_{TaskLC} * \left[\frac{(W+1) * W}{2} \right] \tag{9}$$

$$T_{UpLC} = Tc_{TaskLC} * \frac{(W+1)}{2} \tag{10}$$

The time spent to send a task can be defined as the size of a task (S_{Task}) divided by the local cluster LAN average throughput ($TPut_{LC}$) (11). The *startup time* for the local cluster can also be defined as seen in equation (12).

$$Tc_{TaskLC} = \frac{S_{Task}}{TPut_{LC}} \tag{11}$$

$$T_{UpLC} = \frac{S_{Task}}{TPut_{LC}} * \frac{(W+1)}{2} \tag{12}$$

External cluster – One-to-One Strategy

In order to analyze the one-to-one strategy we consider an external cluster connected to a local cluster through the Internet. One inter-cluster task represents one external cluster task. In this case the communication of tasks happens in a double pipelined scenario where the master to sub-master communication is overlapped with the communication between sub-master and its workers inside the external sub-cluster as illustrated in Figure 3-6.

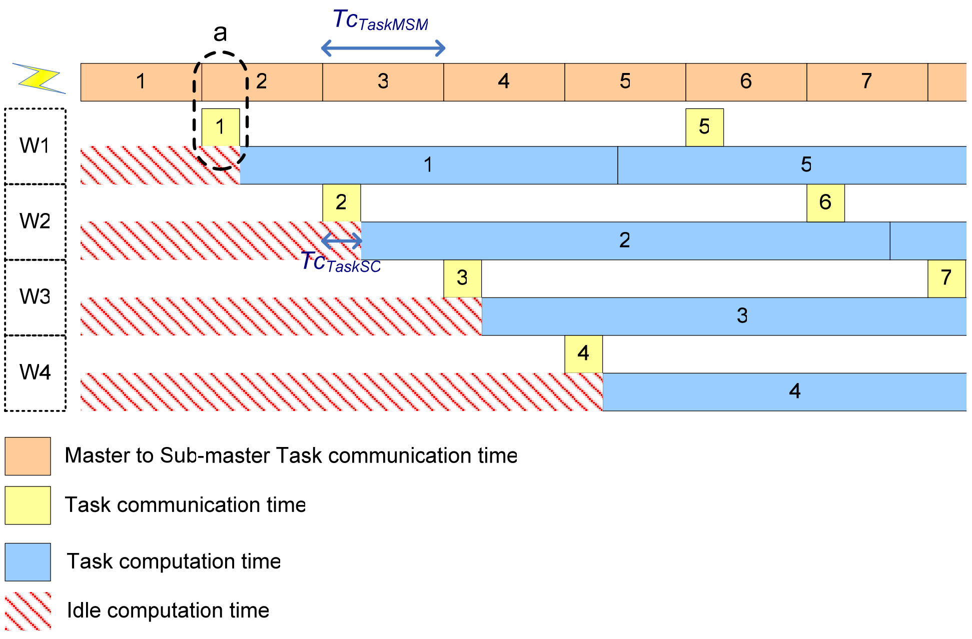


Figure 3-6: Startup time for each worker in an external cluster considering the one inter-cluster task as one intra-cluster task.

The time to send the first task from master to sub-master ($Tc_{TaskMSM}$) is the sum of the time to send this task from master to its communication manager in the local cluster (Tc_{TaskLC}), the time to send the task from one CM to the other through Internet ($Tc_{TaskInet}$) and the time to send a task from CM to sub-master in the sub-cluster LAN (Tc_{TaskSC}) (13).

$$Tc_{TaskMSM} = Tc_{TaskLC} + Tc_{TaskInet} + Tc_{TaskSC} \tag{13}$$

For the second task sent from master to sub-master, Internet communication runs simultaneously with local communications (Figure 3-6 a). The communication time would be the sum of the time for the first communication from the master to its communication manager, two times the time to send a task through Internet and the time to send the task from the communication manager to the master (14). The master to sub-master communication time grows in one Internet communication for each task.

$$Tc_{TaskMSM} = Tc_{TaskLC} + 2 * Tc_{TaskInet} + Tc_{TaskSC} \tag{14}$$

Considering that Internet communication throughput is at least one order of magnitude worst than local communication, we assume that the communication of a task between master and sub-master can be simplified as being the Internet communication time.

The time to send a task through the Internet link can be defined as the size of a task divided by the sub-cluster Internet incoming throughput ($TPut_{InetIN}$) (15).

$$Tc_{TaskMSM} = Tc_{TaskInet} = \frac{S_{Task}}{TPut_{InetIN}} \tag{15}$$

The first worker idle time is the communication time of a task from master to sub-master ($Tc_{TaskMSM}$) plus the communication time of a task in the sub-cluster LAN (Tc_{TaskSC}). For the second worker, the idle time is two times the master to sub-master task communication time plus the LAN communication time. This grows successively until the last worker.

The total computation idle time (Tp_{Idle}) for a sub-cluster with W_{SC} workers is the sum of each worker idle time (16). The total idle time will then be defined by equation (17). The sub-cluster *startup time* (T_{UpSC}) is then the average of the total idle time (18).

$$\begin{aligned} Tp_{Idle} &= (Tc_{TaskMSM} + Tc_{TaskSC}) + (2 * Tc_{TaskMSM} + Tc_{TaskSC}) + \ldots \\ &+ (W_{SC} * Tc_{TaskMSM} + Tc_{TaskSC}) \end{aligned} \tag{16}$$

$$Tp_{Idle} = Tc_{TaskMSM} * \left[\frac{(W_{SC} + 1) * W_{SC}}{2}\right] + W_{SC} * Tc_{TaskSC} \tag{17}$$

(18)

$$T_{UpSC} = Tc_{TaskMSM} * \frac{(W_{SC}+1)}{2} + Tc_{TaskSC}$$

External cluster – One-to-Many Strategy

For the one-to-many distribution strategy, the main cluster sends tasks data packets that result in more than one task on the external cluster. Figure 3-7 illustrates an example of the startup state on which each inter-cluster task has two external cluster tasks.

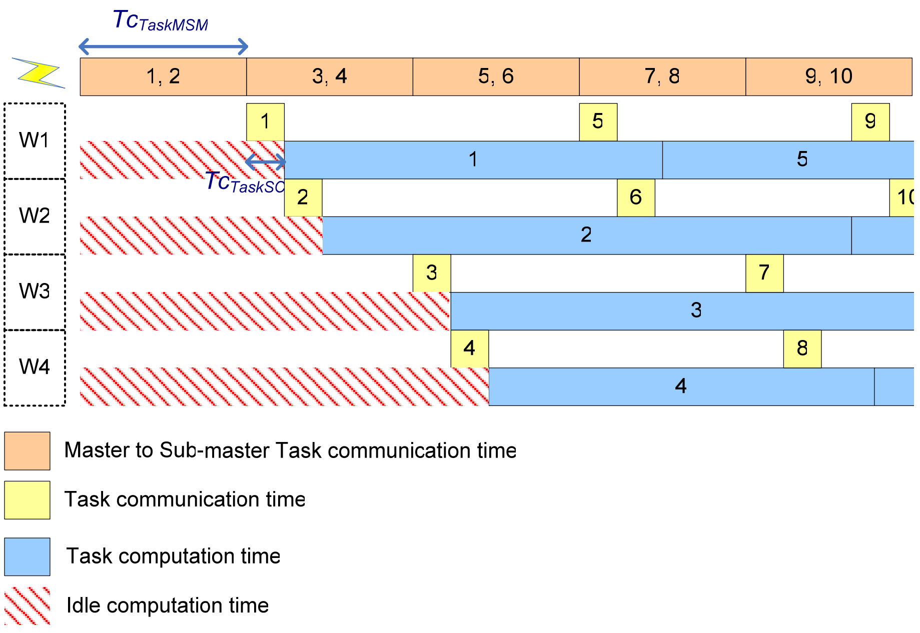


Figure 3-7: Startup state for each worker in an external cluster considering one inter-cluster task represents two intra-cluster tasks.

For this case the sum of the idle time is a bit more complex and has to consider the amount **N** of sub-cluster tasks inside an inter-cluster data packet. It is also relevant the fact that the size of an inter-cluster task ($S_{TaskInter}$) is different than of an intra-cluster task (S_{Task}). The master to sub-master communication time is then the ratio between the size of the inter-cluster task and Internet average throughput (19).

(19)

$$Tc_{TaskMSM} = \frac{S_{TaskInter}}{TPut_{InetIN}}$$

The first worker idle computation time is the sum of the master to sub-master inter-cluster task communication time and the sub-cluster task communication time. From the second worker until the **Nth** worker, the idle time is increased in one LAN task communication time at the sub-cluster.

From the worker **N+1** to the **2*N**, the idle time is similar, increased in a new inter-cluster task communication. The total idle time sum would then be (20) and the *startup time* would be the average of the total idle time for the workers (21).

$$\begin{aligned} Tp_{Idle} &= (Tc_{TaskMSM} + Tc_{TaskSC}) + (Tc_{TaskMSM} + 2 * Tc_{TaskSC}) + \dots \\ &+ (Tc_{TaskMSM} + N * Tc_{TaskSC}) + (2 * Tc_{TaskMSM} + Tc_{TaskSC}) + \dots \end{aligned} \qquad \textbf{(20)}$$

$$T_{UpSC} = \frac{Tp_{Idle}}{W_{SC}} \qquad \textbf{(21)}$$

3.3.2. Steady Time

The steady time is the period on which the pipeline is running steady and computation and communication are overlapped. In the steady state the execution reaches its best performance. The steady state finishes when there are no more tasks to feed workers and workers become idle.

Local cluster

During the steady execution, considering the master does not represent the bottleneck, the execution performance is either limited by the computers performance (computation-bounded) or by the network throughput (communication-bounded).

Figure 3-8 illustrates a computation bounded execution of 20 tasks in a heterogeneous cluster with 4 workers. It can be seen during the steady period some idle moments on the network communication but no idle processing time in workers.

The same 20 tasks sample execution, this time communication bounded because of bigger task sending and result receiving time, can be seen at Figure 3-9. In this example, during the steady execution there was no idle communication moment but there was idle processing time in workers.

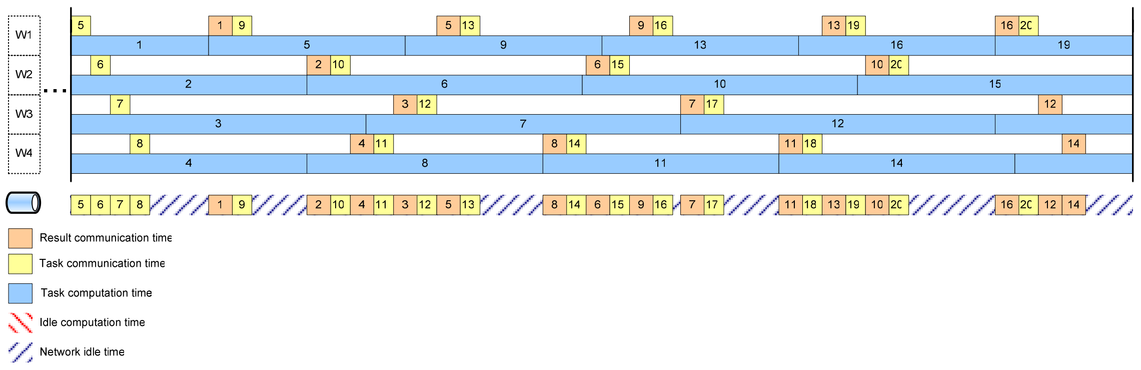


Figure 3-8: An example of a computation bounded execution of 20 tasks in a heterogeneous cluster.

Our objective is to estimate the *steady time*. The *steady time* is the maximum value between *computation time* (*Tp*) and *communication time* (*Tc*) (22).

$$T_{Ste} = \max(Tc, Tp) \tag{22}$$

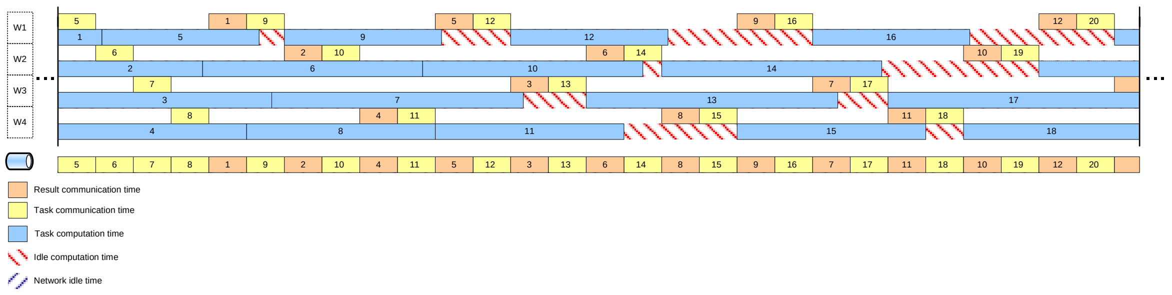


Figure 3-9: An example of a communication bounded execution of 20 tasks in a heterogeneous cluster.

The *computation time* (23) can be defined as the amount of tasks to be executed (*Tasks*) multiplied by the number of basic operations in a task (*Oper*) divided by the cluster available performance (*CPerf$_{Avail}$*) (3). The *communication time* is the amount of tasks multiplied by the size of communication (*S$_{comm}$*) divided by the local cluster LAN throughput (*TPut$_{LC}$*) (24). From this it is possible to calculate the *steady time* (25).

$$Tp = \frac{Tasks * Oper}{CPerf_{Avail}} \tag{23}$$

$$Tc = \frac{Tasks * S_{Comm}}{TPut_{LC}} \tag{24}$$

$$T_{Ste} = \max(\frac{Tasks * Oper}{CPerf_{Avail}}, \frac{Tasks * S_{Comm}}{TPut_{LC}}) \tag{25}$$

It is also important to determine the performance on which the pipeline will stabilize. This performance value represents the best system performance and can be used to dimension the resources to be used in the execution. Using computers that together have an *available performance* greater than this *steady performance* represents a waste in resources and a decrease in system efficiency.

The *steady performance* (*Perf_{Ste}*) (26) is the amount of tasks multiplied by the basic operations of a task divided by the *steady time*.

(26)

$$Perf_{Ste} = \frac{Tasks * Oper}{\max(\frac{Tasks * Oper}{CPerf_{Avail}}, \frac{Tasks * S_{Comm}}{TPut_{LC}}}$$

$$Perf_{Ste} = \min\left(CPerf_{Avail}, Oper * \frac{TPut_{LC}}{S_{Comm}} \right)$$

From the equation (26) it is possible to conclude that the *steady performance* will be the minimum value between the *available performance* and the computation bounded performance limit.

The computation bounded performance limit represents the performance on which the communication would become saturated, the best achievable performance if there was no computation resources limitation.

External cluster – One-to-One Strategy

For the one-to-one analyzed scenario, where one inter-cluster task represents one task data packet for an external cluster, both the communication time of tasks between master and sub-master and the communication time of task results between sub-master and master must also be considered for determining the steady time. It is relevant the possibility that external clusters incoming and outgoing Internet communication throughputs might have different average values.

In such a scenario the *steady time* is not just the maximum from the external cluster computation (*Tp*) and communication (*Tc*) times but also from the master to sub-master total tasks communication time ($Tc_{TotTaskMSM}$) and sub-master to master results total communication time ($Tc_{TotResultSMM}$) (27).

$$T_{Ste} = \max(Tp, Tc, Tc_{TotTaskMSM}, Tc_{TotResultSMM}) \tag{27}$$

As seen at the external cluster startup time explanation, we simplify the master to sub-master tasks communication time as being the time to communicate one task using the Internet (15). Analogously, the sub-master to master result communication time ($Tc_{ResultSMM}$) can be simplified as just being the time to send one result through the Internet, which corresponds to the size of a result divided by the external cluster outgoing throughput ($TPut_{InetOUT}$) (28).

$$Tc_{ResultSMM} = Tc_{ResultInet} = \frac{S_{Result}}{TPut_{InetOUT}} \tag{28}$$

The total time to communicate tasks through the Internet ($Tc_{TotTaskMSM}$) is the amount of tasks multiplied by the time to send one task (29) while the total time to send results ($Tc_{TotResultSMM}$) is the amount of tasks multiplied by the time to send one result (30).

$$Tc_{TotTaskMSM} = Tasks * Tc_{TaskMSM} = Tasks * \frac{S_{Task}}{TPut_{InetIN}} \tag{29}$$

$$Tc_{TotResultSMM} = Tasks * Tc_{ResultSMM} = Tasks * \frac{S_{Result}}{TPut_{InetOUT}} \tag{30}$$

Considering $TPut_{SC}$ as the sub-cluster LAN throughput, the *steady time* in this scenario is represented by equation (31) and the *steady performance* by equation (32).

$$T_{Ste} = \max\left(\frac{Tasks * Oper}{CPerf_{Avail}}, \frac{Tasks * S_{Comm}}{TPut_{SC}}, Tasks * \frac{S_{Task}}{TPut_{InetIN}}, Tasks * \frac{S_{Result}}{TPut_{InetOUT}} \right) \tag{31}$$

$$Perf_{Ste} = \min\left(CPerf_{Avail}, Oper * \frac{TPut_{EC}}{S_{Comm}}, Oper * \frac{TPut_{InetIN}}{S_{Task}}, Oper * \frac{TPut_{InetOUT}}{S_{Result}} \right) \tag{32}$$

External cluster – One-to-Many Strategy

The one-to-many scenario is similar to the previous one with the difference that one inter-cluster task and result represents **N** external sub-cluster tasks and results. The size of an inter-cluster task ($S_{TaskInter}$) and result ($S_{ResultInter}$) might be different (normally greater) than from a local task and result but the advantage is that an inter-cluster task in this scenario carries **N** local tasks.

The time to receive tasks from the Internet (33) and the time to send results at the Internet (34) are then relative to the inter-cluster task and result sizes and the **N** value.

$$Tc_{TotTaskMSM} = \frac{S_{TaskInter}}{N} * \frac{Tasks}{TPut_{InetIN}} \tag{33}$$

$$Tc_{TotResultSMM} = \frac{S_{ResultInter}}{N} * \frac{Tasks}{TPut_{InetOUT}} \tag{34}$$

In this case the *steady time* is given by equation (35) and the *steady performance* by equation (36).

$$T_{Ste} = \max\left(\frac{Tasks * Oper}{CPerf_{Avail}}, \frac{Tasks * S_{Comm}}{TPut_{EC}}, \frac{S_{TaskInter}}{N} * \frac{Tasks}{TPut_{InetIN}}, \frac{S_{ResultInter}}{N} * \frac{Tasks}{TPut_{InetOUT}} \right) \tag{35}$$

$$Perf_{Ste} = \min\left(CPerf_{Avail}, Oper * \frac{TPut_{EC}}{S_{Comm}}, Oper * \frac{N * TPut_{InetIN}}{S_{TaskInter}}, Oper * \frac{N * TPut_{InetOUT}}{S_{ResultInter}} \right) \tag{36}$$

3.3.3. End Time

The end state starts when the master has no more tasks to distribute and the first worker finishes its tasks becoming idle. The end state goes until the moment the master join the last task. There are several possible configurations for the end state depending on resources heterogeneity, amount of tasks, order of tasks distribution, communication properties, etc.

We analyze the most significant of those possible scenarios for determining the application execution time and efficiency. These scenarios are the best and worst possible end time. The best and worst possible end time will lead to the best and worst execution time and efficiency. The worst possible end time must be used when targeting to guarantee an efficiency threshold.

For determining the worst possible end time, two alternatives are evaluated. Firstly it is considered the application does not reassign tasks when the last task is distributed. The second alternative considers the master reassigns the last task to every worker that finishes its execution until the total problem is solved. It is also evaluated the possibility of the application being communication bounded and the end time be restricted to results communication.

Local cluster

At the local cluster, the best end state happens when the execution finishes perfectly balanced, just like the pipeline startup. Each worker would finish its execution in the exact time the network is available for it to send the task result data (Figure 3-10).

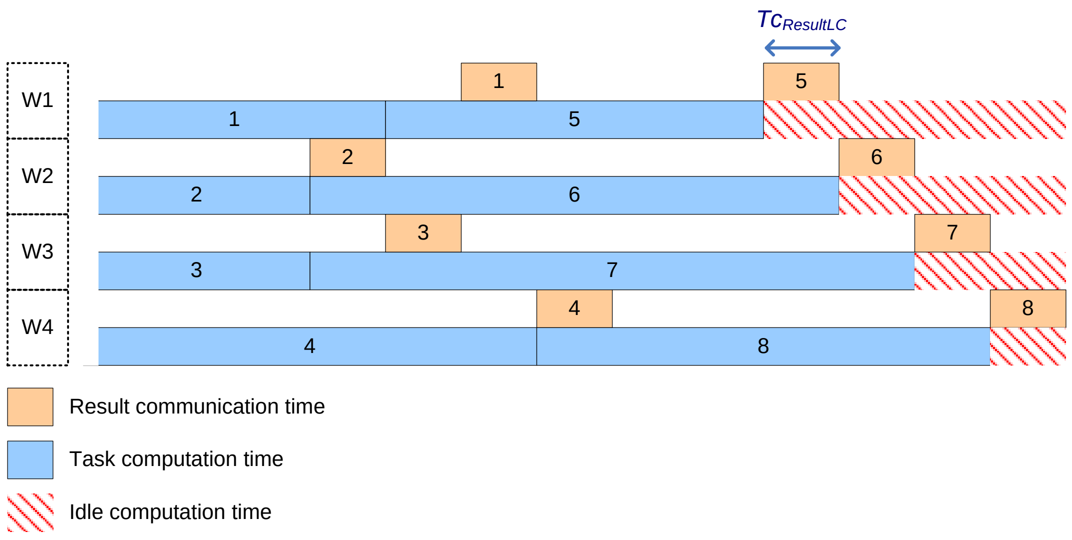


Figure 3-10: Best execution ending scenario.

In this scenario, the best end time value for the local cluster ($T_{BestEndLC}$) is similar to the startup time (14), considering the size of the task result (S_{Result}) (37).

$$T_{BestEndLC} = \frac{S_{Result} * (W + 1)}{2 * TPut_{LC}} \quad \textbf{(37)}$$

The worst end possible scenarios depend if the execution is **computation** or **communication bounded**.

The worst **computation bounded** scenario happens if the final task goes to the worst worker when all workers finished their tasks at almost same time (Figure 3-11).

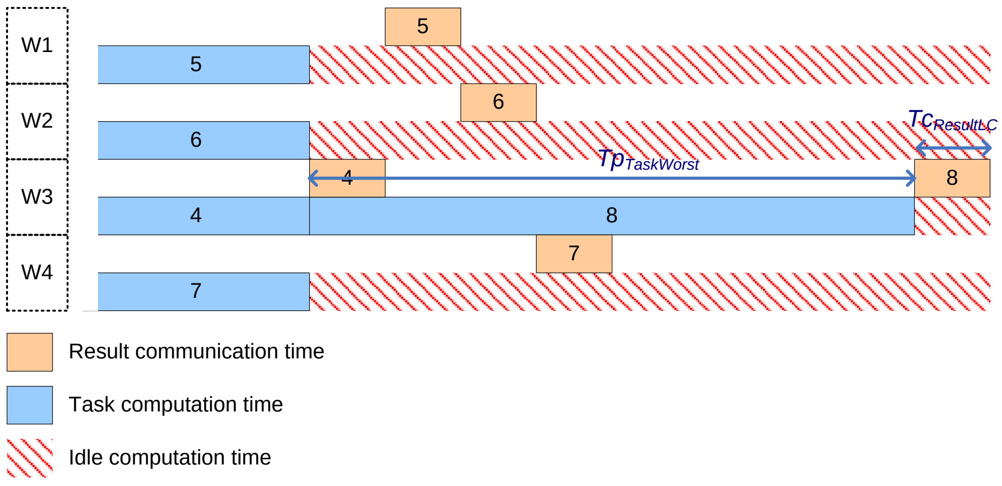


Figure 3-11: Worst execution ending scenario.

For this scenario the total computation idle time is the worst worker task execution time ($Tp_{TaskWorst}$) multiplied by the amount of worker minus one. To this value it should be added the time for each worker to send a task result in the local cluster LAN ($Tc_{ResultLC}$) (38). The worst ending time ($T_{WorstEndLC}$) (39) is then the average of the idle time for the workers.

$$Tp_{Idle} = Tp_{TaskWorst} * (W-1) + W * Tc_{ResultLC} \tag{38}$$

$$T_{WorstEndLC} = Tp_{TaskWorst} * \frac{(W-1)}{W} + Tc_{ResultLC} \tag{39}$$

It is possible for the master, when a worker finishes its tasks, to reassign the last task to this worker because in heterogeneous clusters this worker might finish faster than the one that is executing this last task.

If reassignment is considered, the worst possible **computation bounded** scenario happens when the last task is reassigned to all computers and they all finish this last task at the same time, meaning that every computer but one wasted one execution time. The worst scenario happens when the best computer is the one whose execution time is not wasted, maximizing the idle time (Figure 3-12).

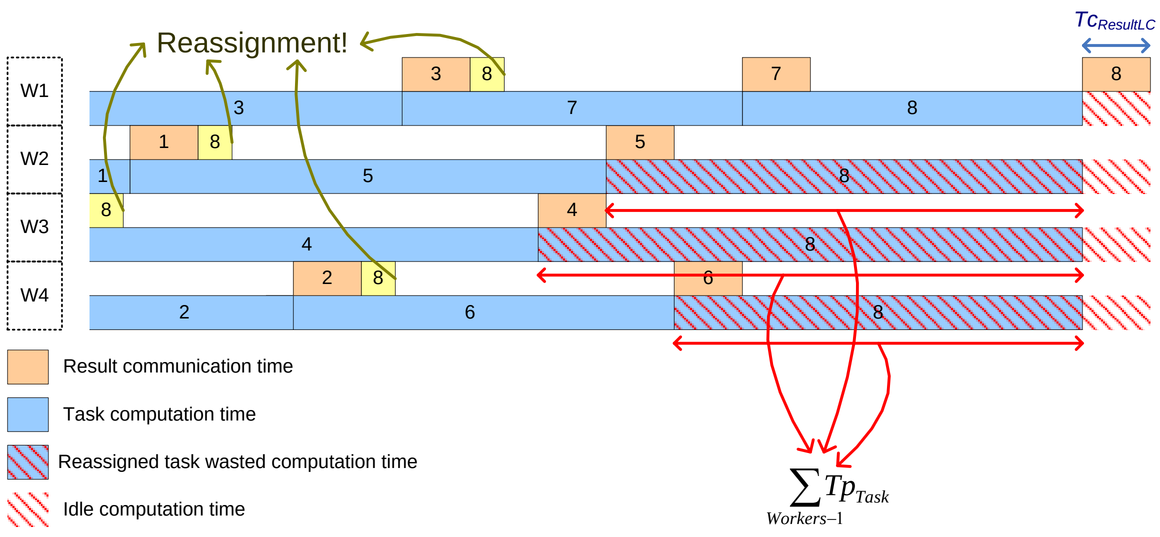

Reassignment!
$Tc_{ResultLC}$
W1
W2
W3
W4
Result communication time
Task computation time
Reassigned task wasted computation time
Idle computation time
$\sum_{Workers-1} Tp_{Task}$

Figure 3-12: Worst execution ending scenario if the last task is reassigned.

In this case the worst end time (40) is the addition of the time to send a task result data ($Tc_{ResultLC}$) and the average time each worker, except one (the worst case is that this one is the best worker), spent on processing one task (Tp_{Task}). The time to send a task result data (41) is the ratio between the task result data size (S_{Result}) and the local cluster network throughput. The time it takes to a worker to execute a task (42) is the amount of operations of the task divided by the worker available performance. Finally, the worst scenario ending time can also be seen as equation (43).

$$T_{WorstEnd} = Tc_{ResultLC} + \frac{1}{W} * \sum_{Workers-1} Tp_{Task} \tag{40}$$

$$Tc_{ResultLC} = \frac{S_{Result}}{TPut_{LC}} \tag{41}$$

$$Tp_{Task} = \frac{Oper}{Perf_{Avail}} \tag{42}$$

$$T_{WorstEnd} = \frac{S_{Result}}{TPut_{LC}} + \frac{1}{W} * \sum_{Workers-1} \frac{Oper}{Perf_{Avail}} \tag{43}$$

When the application is **communication bounded**, the worst scenario happens when all workers finish their last tasks at the same time. In this case the communications will be done without any computation (Figure 3-13).

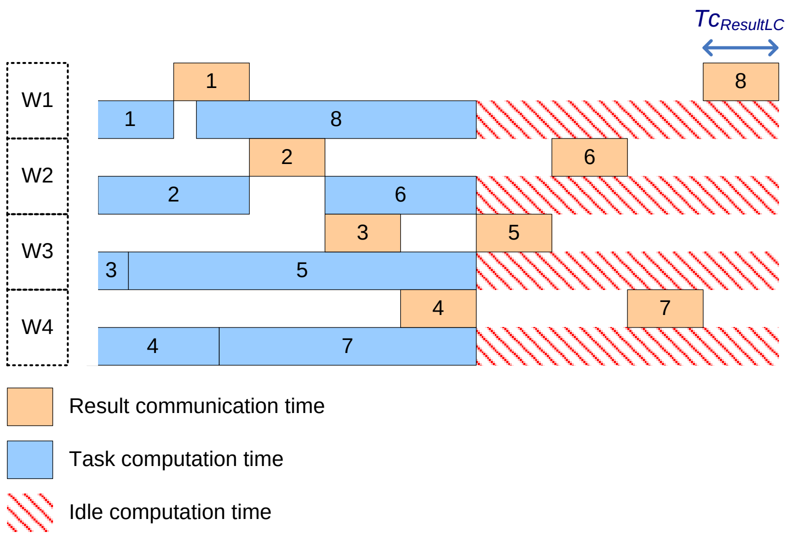


Figure 3-13: Worst execution ending scenario for communication bounded clusters.

The worst end time (44) would then be the number of workers multiplied by the time to send a task result.

$$T_{WorstEnd} = W * Tc_{Result} = W * \frac{S_{Result}}{TPut_{LC}} \tag{44}$$

In summary the best end is the perfect balanced pipeline and the worst end time depends if the application is computation or communication bounded and if it reassigns the last task.

External cluster – One-to-One Strategy

Considering that every task result for an external cluster needs to be communicated back to the main cluster, both the best and worst ending scenarios time differs from the local cluster by the inter-cluster communication pipeline.

The best ending scenario occurs when the pipeline finishes perfectly balanced, considering the time for a task result to be communicated from the sub-master to the master (Figure 3-14).

The idle processing time for the last computer in this scenario is the time to send a result in the external cluster LAN plus ($Tc_{ResultSC}$) the time to send a result from sub-master to main master ($Tc_{ResultSMM}$). For the previous worker, the idle processing time it is time to send a result in the external cluster LAN plus two times the time to send it to the main cluster.

This value increases in one inter-cluster communication until the first worker (45). The total idle time is then defined by equation (46).

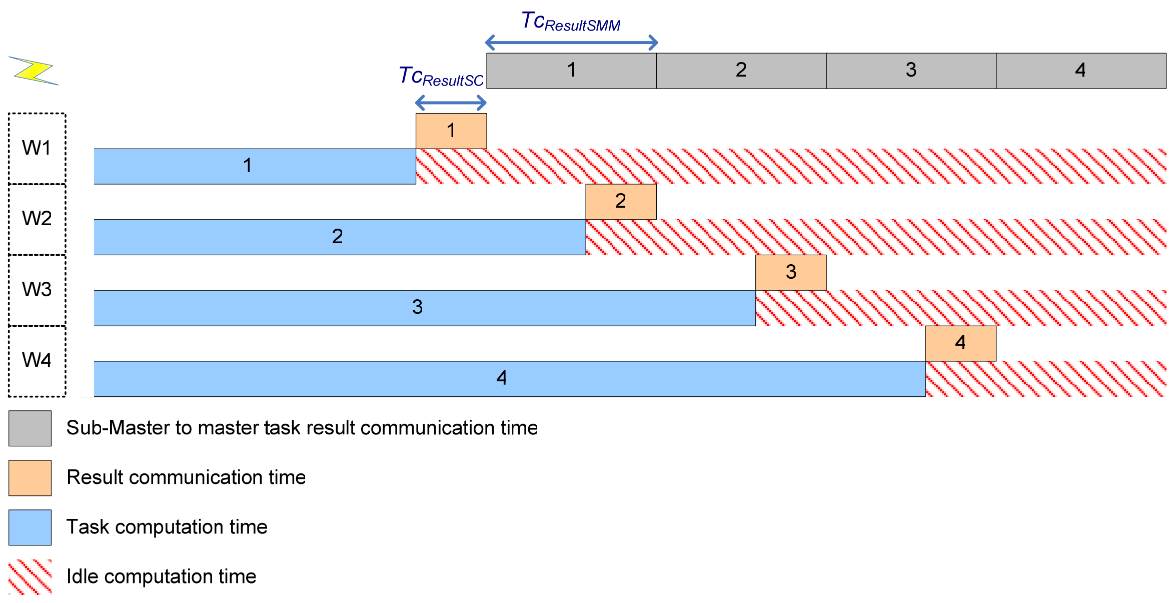


Figure 3-14: Best execution ending for external clusters in the first scenario.

$$Tp_{Idle} = Tc_{ResultSC} + Tc_{ResultSMM} + \\ Tc_{ResultSC} + 2 * Tc_{ResultSMM} + \\ \ldots + Tc_{ResultSC} + W_{SC} * Tc_{ResultSMM} \tag{45}$$

$$Tp_{Idle} = W_{SC} * Tc_{ResultSC} + \frac{W_{SC} * (W_{SC} - 1)}{2} * Tc_{ResultSMM} \tag{46}$$

The end time (47) is then the average of the idle time for the workers. Similarly to the time to send a task, the time the result takes from the sub-master to the master ($Tc_{ResultSMM}$) is simplified as the time it takes to send a result between the communication managers in the Internet ($Tc_{ResultInet}$) (48).

$$T_{BestEnd} = Tc_{ResultSC} + \frac{(W_{SC} - 1)}{2} * Tc_{ResultSMM} \tag{47}$$

$$Tc_{ResultSMM} = Tc_{ResultInet} = \frac{S_{Result}}{TPut_{InetOUT}} \tag{48}$$

The worst end cases are also similar to the ones of the local cluster analysis. If the application is **computational** bounded with no reassignment of the last task than the worst scenario is when workers finish at the same tame and the last task goes to the worst worker (Figure 3-15).

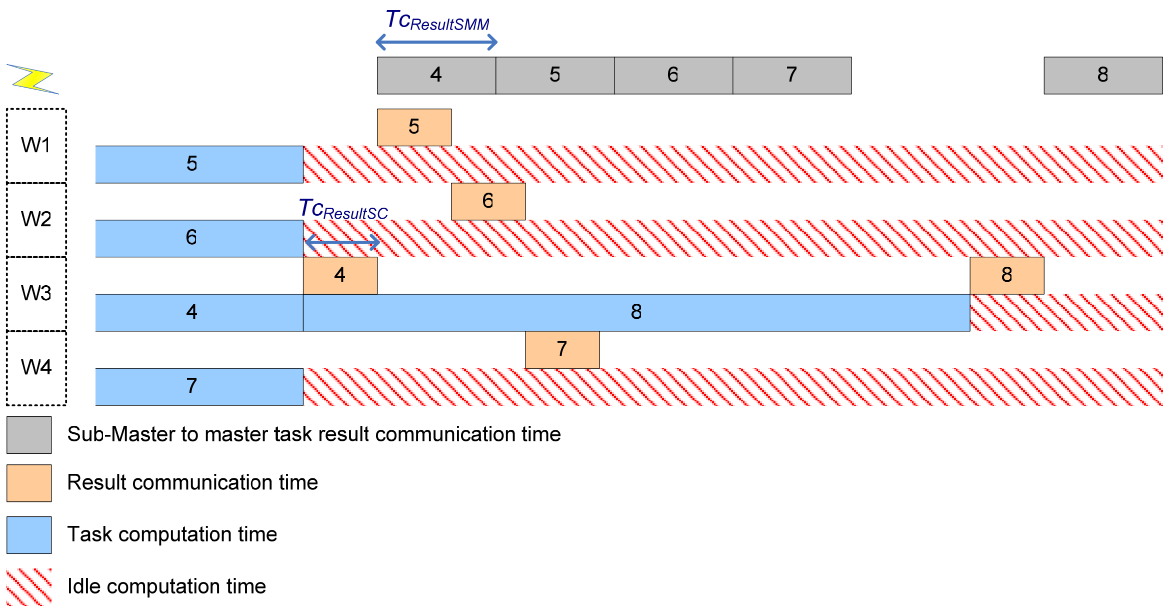


Figure 3-15: Worst execution ending for external clusters in the first scenario when the last task is not reassigned.

In this case the worst end time (49) is similar to the local cluster just adding the time to send a result from the sub-master to the main master.

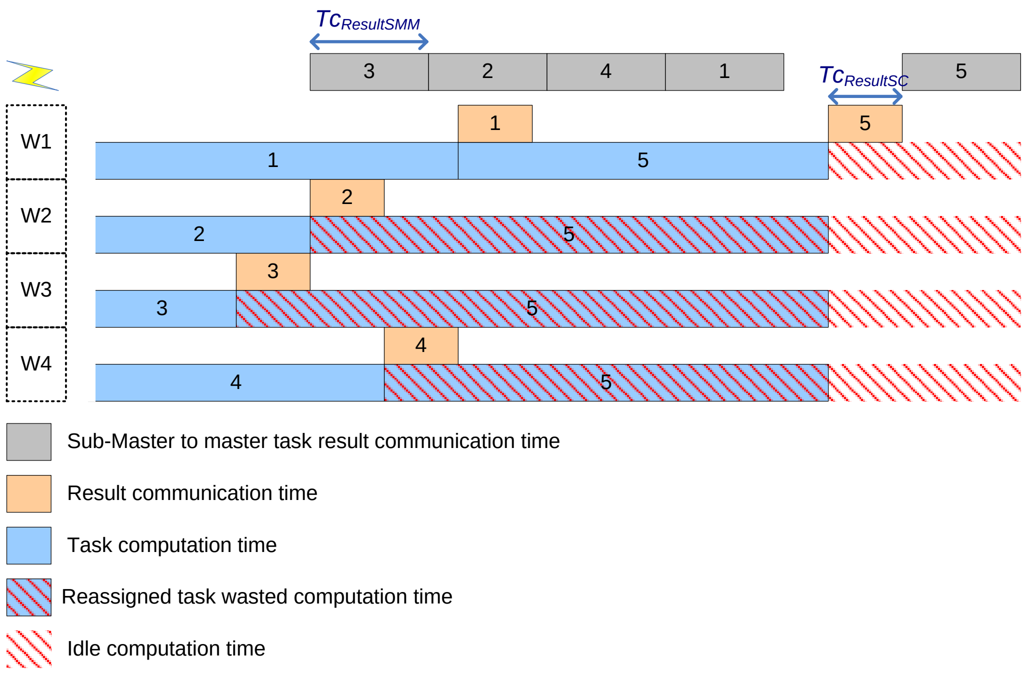

$Tc_{ResultSMM}$
3
2
4
1
$Tc_{ResultSC}$
5
W1
W2
W3
W4
Sub-Master to master task result communication time
Result communication time
Task computation time
Reassigned task wasted computation time
Idle computation time

Figure 3-16: Worst execution ending for external clusters in the first scenario when the last task is reassigned.

$$T_{WorstEnd} = Tp_{TaskWorst} * \frac{(W_{SC} - 1)}{W_{SC}} + Tc_{ResultSC} + Tc_{ResultSMM} \tag{49}$$

The same situation happens when tasks are reassigned (Figure 3-16) and to reach the worst end value it is just necessary to add the time for the result to be communicated from the sub-master to the main master (50).

$$T_{WorstEnd} = Tc_{ResultSC} + \frac{1}{W_{SC}} * \sum_{Workers-1} Tp_{Task} + Tc_{ResultSMM} \tag{50}$$

In the case it is **communication** bounded, it is possible that at the worst end all the workers wait for the slower sub-master to master communication of results (Figure 3-17). The communication bounded worst end time value is seen in equation (51).

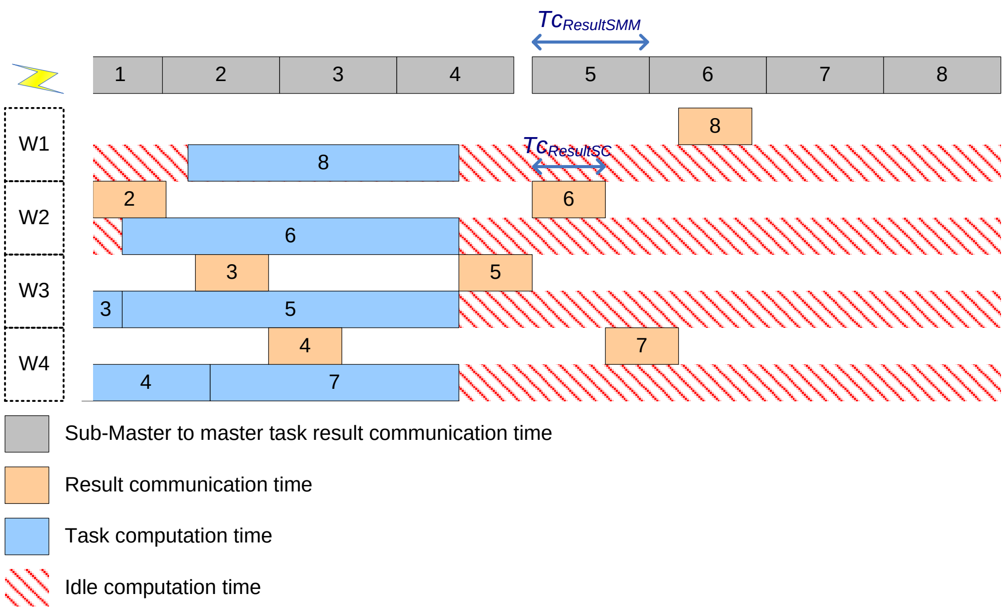


Figure 3-17: Worst execution ending for external clusters in the first scenario when the application is communication bounded.

$$T_{WorstEnd} = Tc_{ResultEC} + W * Tc_{ResultSMM} \tag{51}$$

For **communication** bounded executions the worst end time would be the maximum between the computation bounded – (49) or (50) – and the communication bounded (51) worst end times.

External cluster – One-to-Many Strategy

At the second studied scenario, one task result sent from the external cluster to the main represents the joint of N tasks results executed inside the external cluster. The size of the inter-cluster result ($S_{ResultInter}$) is then different from the intra-cluster result (S_{Result}).

The best ending scenario occurs when the pipeline finishes perfectly balanced (Figure 3-18). The idle computation time for the first computer in this scenario is the time to send N results in the external cluster LAN ($Tc_{ResultSC}$) plus W/N times the time to send a result from the sub-master to the main master ($Tc_{ResultSMM}$).

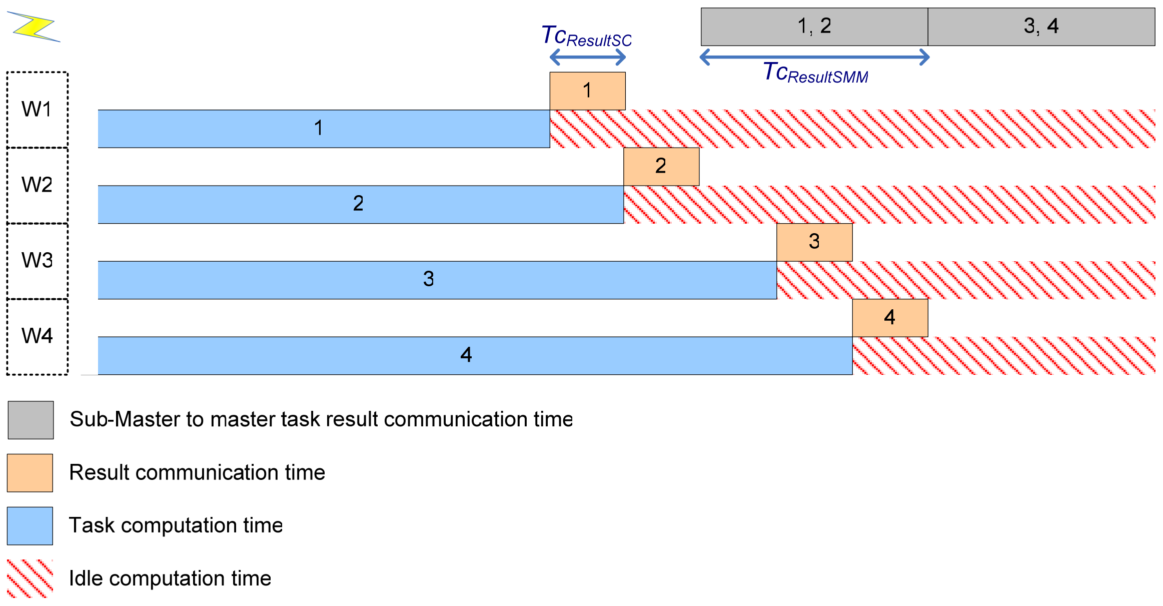


Figure 3-18: Best execution ending for external clusters in the second scenario.

For the second worker it is the time to send N-1 results in the external cluster LAN plus W/N times the time to send it to the main cluster. This goes successively for the first N workers. At this point the next worker idle time will be the time of N LAN results plus (W/N - 1) sub-master to master result. The total idle computation time is then (52).

$$Tp_{Idle} = N * Tc_{ResultSC} + \frac{W_{SC}}{N} * Tc_{ResultSMM} + (N-1) * Tc_{ResultSC} + \frac{W_{SC}}{N} * Tc_{ResultSMM} + \\ \ldots + Tc_{ResultSC} + \frac{W_{SC}}{N} Tc_{ResultSMM} + N * Tc_{ResultSC} + \left(\frac{W_{SC}}{N} - 1\right) Tc_{ResultSMM} + \\ \ldots + Tc_{ResultSC} + Tc_{ResultSMM} \tag{52}$$

The best end time is then the average of the idle time for the workers (53).

(53)

$$T_{BestEnd} = \frac{Tp_{Idle}}{W_{SC}}$$

The worst end cases are similar to the ones of the first studied scenario. If the application is **computational** bounded with no reassignment of the last task than the worst scenario is when the workers finish at the same tame and the last task goes to the worst worker (Figure 3-19).

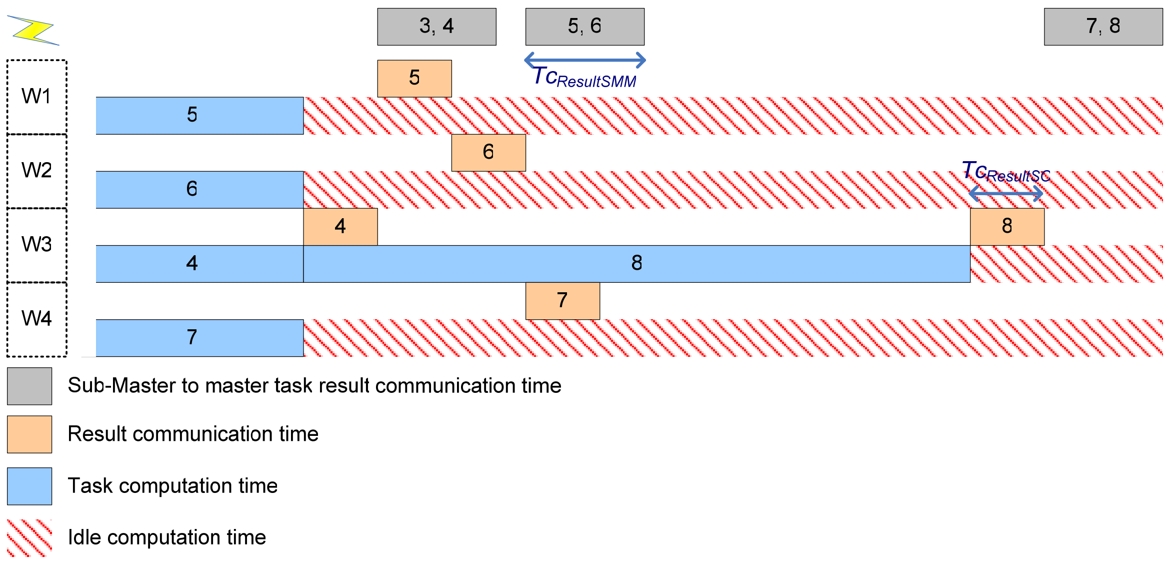


Figure 3-19: Worst execution ending for external clusters in the second scenario when the last task is not reassigned.

In this case the worst end time is similar to the first scenario (54).

(54)

$$T_{WorstEnd} = Tp_{TaskWorst} * \frac{(W_{SC} - 1)}{W} + Tc_{ResultSC} + Tc_{ResultSMM}$$

The same situation happens when tasks are reassigned (Figure 3-20), the worst end equation is similar to the one of the first scenario (55).

(55)

$$T_{WorstEnd} = Tc_{ResultSC} + \frac{1}{W_{SC}} * \sum_{Workers-1} Tp_{Task} + Tc_{ResultSMM}$$

In the case it is **communication** bounded, it is possible that at the worst end all the workers wait for the slower sub-master to master communication of results, with the difference that this communication starts after the first N local task communications and repeats W/N times (Figure 3-21). The worst end time value would then be (56).

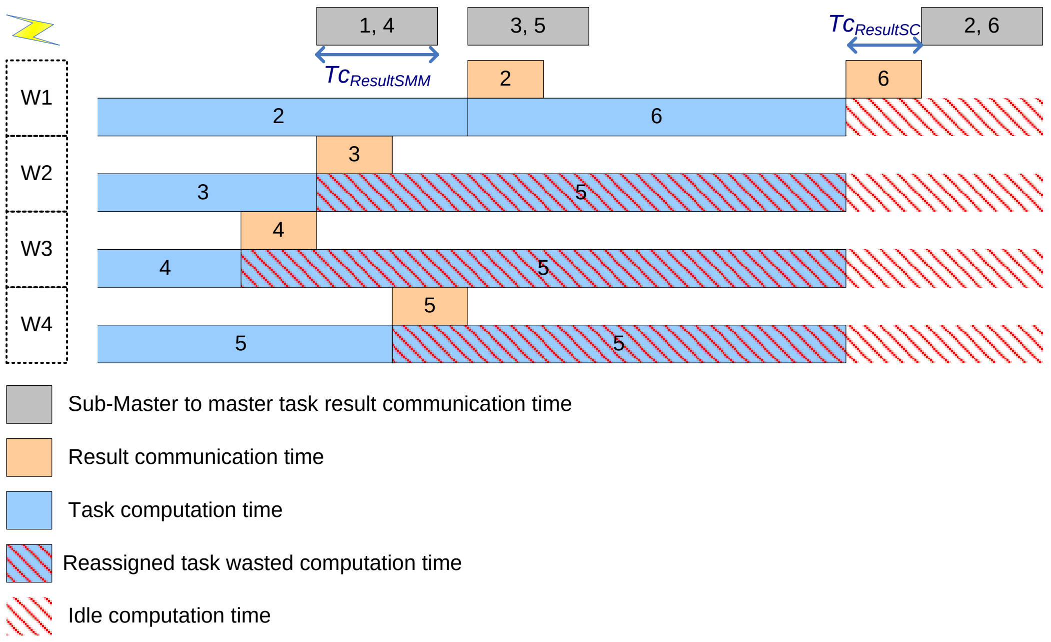

1, 4
3, 5
2, 6
$Tc_{ResultSMM}$
$Tc_{ResultSC}$
W1
W2
W3
W4
Sub-Master to master task result communication time
Result communication time
Task computation time
Reassigned task wasted computation time
Idle computation time

Figure 3-20: Worst execution ending for external clusters in the second scenario when the last task is reassigned.

$$T_{WorstEnd} = N * Tc_{ResultSC} + \frac{W_{SC}}{N} * Tc_{ResultSMM} \tag{56}$$

The worst end time would be the maximum between the computation bounded – (54) or (55) – and the communication bounded (56) worst end times.

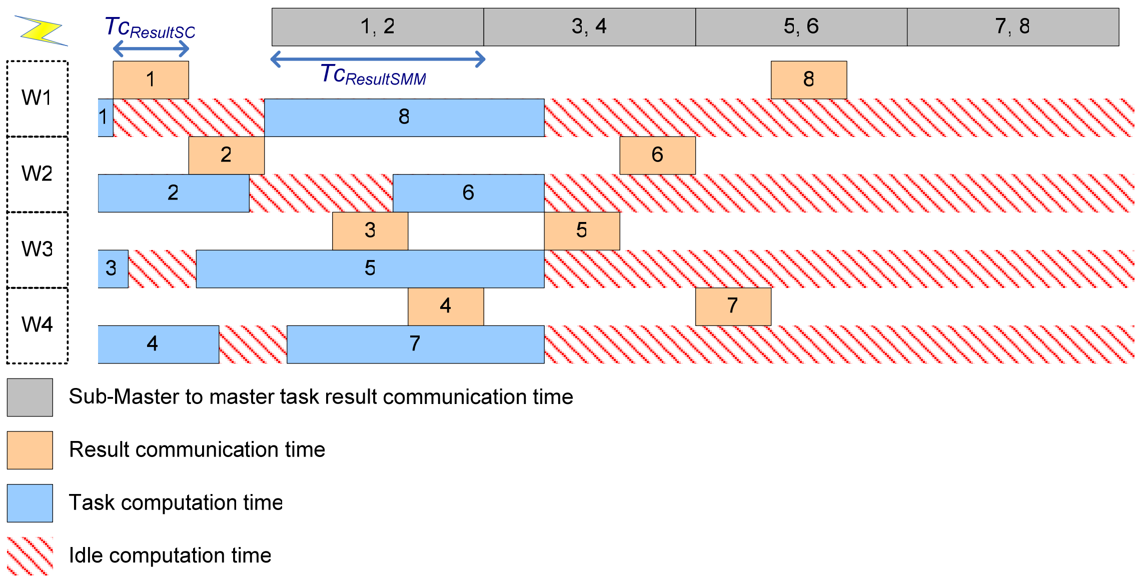


Figure 3-21: Worst execution ending for external clusters in the second scenario when the application is communication bounded.

3.4. Conclusions

In this chapter the proposed analytical model was presented. The analytical model is based in computation-communication analysis. The proposed analytical model evaluates the execution time and efficiency for an application execution in a multi-cluster. The model has as inputs some characteristics of the application (tasks of a workload, basic operations of a task, and basic operations of a workload) and multi-cluster system (available performance and network throughput).

In the analytical model, the two-stage master-worker pipelined execution is divided in three states: startup, pipeline and end. Equations are developed to estimate the duration of each of these states for a local cluster distribution and for an external cluster. In the case of the external cluster, two possible data distribution strategies are evaluated. It is important to remark that there are other different possible data strategies distributions for external clusters that would require new synthesis of equations.

If more than one scenario was possible for each execution part then the best and worst times were evaluated. This leads to the best and worst possible execution times.

The model serves as base to analyze applications and evaluate their efficiency in multi-clusters.

Chapter 4

Performance Prediction and System Tuning

4.1. Introduction

Two are our main goals on using multi-cluster environments. Firstly it is to decrease the application execution time in a multi-cluster when compared to the time spent for the application to run in a single cluster. Secondly is to guarantee that this speedup is obtained without a significant waste of local and remote resources, guaranteeing a chosen minimum level of efficiency.

At Chapter 3 an analytical performance model for multi-cluster execution of applications was introduced. This performance model analyzes the multi-cluster master-worker execution, based on the system architecture proposed at Chapter 2. The analytical model evaluates the application execution time and efficiency dividing the execution in three different states (startup, steady and end).

The analytical model can be used as the basis for *prediction*, *efficiency obtainment* and *application tuning* in multi-cluster environments.

At the *prediction* level it is possible to evaluate if it is possible to reduce the application execution time using a multi-cluster and to estimate the execution time reduction. It is also possible to evaluate the efficiency of each cluster and of the multi-cluster as a whole.

For the *efficiency obtainment*, the model can be used to select which clusters and computers in each cluster should be used to reduce the execution time if, for example, a minimum efficiency of 90% is desired in each cluster. Once the resources are selected it is possible to predict the application execution time for this selection.

In other words, it is possible to evaluate what would be the adequate configuration of resources that could be efficiently used to diminish the execution time of the application.

The model can be used to give support to the *application tuning* in case the obtainable execution time is not satisfactory. The application tuning process can be supported by the evaluation of the stages that represent the performance and efficiency bottleneck, identifying the model parameters that need to be tuned in order to achieve better levels of usage of the multi-cluster resources.

A performance prediction and system tuning methodology is proposed [10] to guide through the process of evaluating a parallel application execution in a multi-cluster, estimating the execution time and determining the clusters and resources on which the execution efficiency is superior to a specified threshold. The methodology has as inputs the parallel application, the desired threshold efficiency and the target multi-cluster.

On another hand, the methodology can be used for giving guidance to the application tuning, giving support to changes in the application workload distribution in a way that the execution time is decreased with an increment on the efficiency.

In this chapter the proposed performance prediction and system tuning methodology is described through its three basic phases: Local Cluster Analysis, Multi-Cluster Analysis, and Application Tuning

It is also presented in this chapter a developed hierarchical master-worker framework. This framework aims to provide tools to support the methodological process, implementing the proposed system architecture performance improvements, and monitoring the analytical model parameters during the execution to make available accurate time prediction.

4.2. Methodological phases

The performance prediction and system tuning methodology is divided in three basic phases providing an estimated speedup, efficiency, and the selected resources to be used in each cluster. The basic phases are Local Cluster Analysis, Multi-Cluster Analysis, and Application Tuning. Figure 4-1 shows the methodology basic flow diagram.

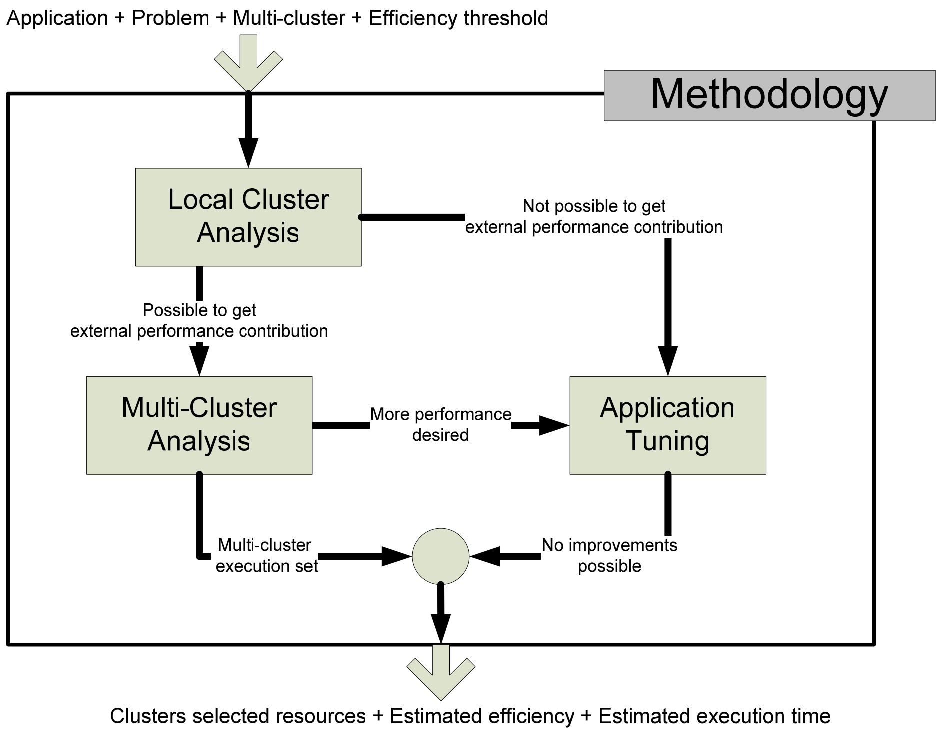

Application + Problem + Multi-cluster + Efficiency threshold
Methodology
Local Cluster Analysis
Not possible to get external performance contribution
Possible to get external performance contribution
Multi-Cluster Analysis
More performance desired
Application Tuning
Multi-cluster execution set
No improvements possible
Clusters selected resources + Estimated efficiency + Estimated execution time

Figure 4-1: Flow diagram of the multi-cluster performance prediction and application tuning methodology.

The methodology starts with the existence of a parallel application, a problem to be solved, a local cluster and external clusters (multi-cluster) that could help in the parallel application execution.

The Local Cluster Analysis is the first phase and evaluates if the application in the local cluster is apt to obtain external cluster's performance contribution, reducing the execution time. When external clusters can collaborate raising up the speedup then the Multi-Cluster Analysis is the following phase. On the contrary, if speedup is not possible, the application should be tuned to be able to use the multi-cluster resources.

The Multi-Cluster Analysis evaluates the speedup that each available external cluster can add to the local one executing the specific application. Each cluster's resources are then selected to reach the evaluated speedup respecting the target efficiency. The Multi-Cluster Analysis also provides the attainable speedup for the application execution using these resources.

There are two possible outputs from the Multi-Cluster Analysis. The first output, when the obtainable performance is satisfactory, leads to the end of the methodological process. The other output, for improving the performance, is the Application Tuning phase.

The Application Tuning evaluates the possibilities of adapting the application data distribution strategy for improving the collaborative performance. Some performance recommendations are presented to guide the possible application changes. If the application is changed the application needs to be re-evaluated and the Application Tuning is followed by the Local Cluster Analysis phase.

4.3. Local Cluster Analysis

The main process inside the Local Cluster Analysis (LCA) is to apply the analytical performance model, to evaluate the collaboration possibility of external cluster in the application execution.

This analysis can be divided in some steps as shown at Figure 4-2. The first step is the *application parameters analysis*. In this step the application is studied, the basic operation metric and the computation and communication analytical model application parameters are defined. If possible the application parameters should be defined as a function of the granularity.

The following step is then the *system parameters obtainment*. In this step it needs to be determined the analytical model system parameters for the local cluster. It is important that the parameters obtainment is as precise as possible, acquired in similar conditions of the application execution. If different granularities are possible for the problem execution then the parameters can be obtained for some possible granularities or, at this moment, an arbitrary granularity can be fixed.

Once obtained the system parameters (computers performance and network throughput) it is possible to do the *architecture roles mapping*. The *architecture roles mapping* is oriented to the selection of computers that will have master and communication manager processes. The communication manager process needs to be in a computer that is connected to both cluster local area network and external network. It is preferable that the communication manager computer is connected to both networks with different network cards so that inter-cluster does not interfere in intra-cluster communication. If more than one computer follows this option, the computer with lower computational power should be selected.

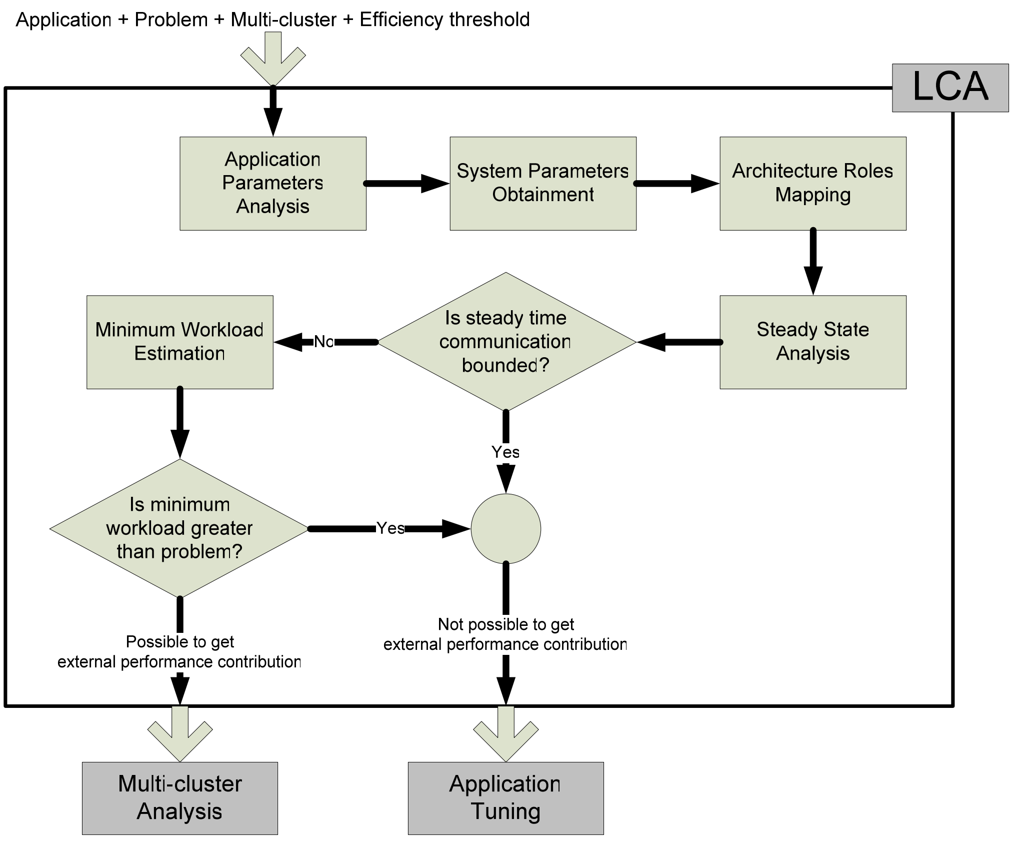

Application + Problem + Multi-cluster + Efficiency threshold
LCA
Application Parameters Analysis
System Parameters Obtainment
Architecture Roles Mapping
Steady State Analysis
Is steady time communication bounded?
No
Minimum Workload Estimation
Yes
Is minimum workload greater than problem?
Yes
Possible to get external performance contribution
Not possible to get external performance contribution
Multi-cluster Analysis
Application Tuning

Figure 4-2: Local Cluster Analysis flow diagram.

Mapping the master process requires a different approach, not to turn master into an unnecessary bottleneck. It is common in master-worker applications for a master to do the task joining work which might demand some processing power. The master workstation should be the less processing power one that can do the task joining work keeping LAN throughput in its maximum.

This can be determined simply by checking if for the less processing power workstation it is possible to reach, in a parallel application execution, the expected steady performance. If the expected performance is reached then the master is not the bottleneck.

If the cluster uses a network file system (NFS), the possibility of avoiding NFS communication traffic should be studied. This can be done by setting the NFS server to be the master computer or by storing application data in the master computer local hard drive. All the computers that do not have master or communication manager roles are set to have worker processes.

After finishing the roles mapping it is possible to start the analyses. The next step is *steady state analysis*. In the *steady state analysis* the execution performance for the steady state is estimated. From this estimation it can be concluded if the application using all resources of the local cluster is **computation** or **communication** bounded.

If the application in the steady state is **communication** bounded (communication time is greater than computation time) then it is not possible for this application to use all its local resources. In this case it is useless to target external clusters collaboration. The Local Cluster Analysis should then lead to the Application Tuning phase.

If the application is **computation** bounded in the steady state then the next step is the *minimum workload estimation*. The objective of this step is to use the performance model to estimate the *minimum workload* for which the efficiency threshold is respected for the local cluster. This estimation is done setting that the worst execution efficiency needs to be greater than or equal to the threshold (57) deducing the workload. This deduction process is shown in (58).

$$Efficiency \geq Eff_{Threshold} \tag{57}$$

$$\begin{gathered}
Efficiency \geq Eff_{Threshold}, \text{using (7)} \\
\frac{T_{best}}{T_{Up} + T_{Ste} + T_{WorstEnd}} \geq Eff_{Threshold}, \text{using (6)} \\
\frac{\dfrac{Workload}{CPerf_{Avail}}}{T_{Up} + \dfrac{Workload}{Perf_{Ste}} + T_{WorstEnd}} \geq Eff_{Threshold}, \text{inferring the workload} \\
Workload \geq \left(T_{Up} + T_{WorstEnd}\right) * \frac{Perf_{Ste} * CPerf_{Avail} * Eff_{Threshold}}{\left(Perf_{Ste} - CPerf_{Avail} * Eff_{Threshold}\right)}
\end{gathered} \tag{58}$$

In the particular case of **computation** bounded applications where the *steady performance* is the *cluster available performance*, the minimum workload equation is (59). If it is desired to know the minimum amount of tasks, the minimum workload should be divided by the amount of basic operations in a task.

$$Workload \geq (T_{st} + T_{WorstEnd}) * CPerf_{Avail} * \frac{Eff_{Threshold}}{(1 - * Eff_{Threshold})} \tag{59}$$

If the minimum workload is greater than the total workload of the problem being solved then it would just be possible to achieve the efficiency threshold in the local cluster by releasing computers of the execution. In this case it would not be possible for the local cluster to attain external clusters performance and the LCA should be followed by the Application Tuning phase.

In the case that the minimum workload is just part of the total problem than it is possible to attain external clusters help and the LCA precedes the Multi-Cluster Analysis.

4.4. Multi-Cluster Analysis

The Multi-Cluster Analysis (MCA) targets to evaluate the external clusters and establish the resources that can be used in each cluster for diminishing the execution time, respecting the efficiency threshold. The MCA smaller steps flow diagram can be seen in the Figure 4-3.

The first step of the MCA is the *inter-cluster application parameters analysis* which should be done in case the data distribution policy or granularity between clusters is different than in the local cluster. The objective of this analysis is to define the application parameters for the local to external cluster data distribution.

The following steps are the *system parameters obtainment* and *architecture roles mapping* that have the same procedure as their LCA analogous but applied to each external cluster.

The *steady stage analysis* is also similar with the correspondent step in the LCA with the difference that in the MCA this analysis needs to verify if each external cluster is not just communication bounded by the LAN but also by the Internet incoming and outgoing communication with the local cluster.

After evaluated if the application in the external cluster is computation or communication bounded the MCA goes to the *resources selection* step. The target of the *resources* *selection* is to evaluate if and which computers should be dropped of the execution because of the efficiency threshold target ($Eff_{Threshold}$).

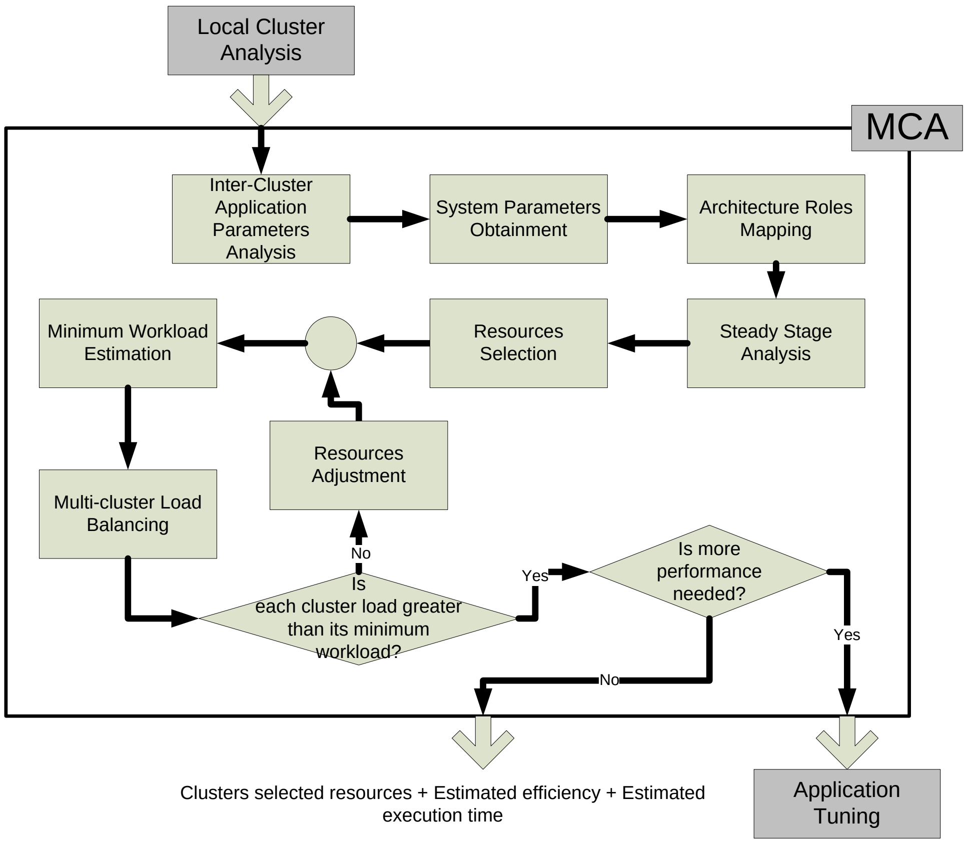

Local Cluster Analysis
MCA
Inter-Cluster Application Parameters Analysis
System Parameters Obtainment
Architecture Roles Mapping
Minimum Workload Estimation
Resources Selection
Steady Stage Analysis
Resources Adjustment
Multi-cluster Load Balancing
No
Is each cluster load greater than its minimum workload?
Yes
Is more performance needed?
Yes
No
Clusters selected resources + Estimated efficiency + Estimated execution time
Application Tuning

Figure 4-3: Multi-Cluster Analysis flow diagram.

Supposing the startup and ending times values dischargeable, it is possible to conclude that the system efficiency is upper bounded by the ratio between the steady and available performances as deduced in the equation (60).

$$
\begin{aligned}
&Efficiency \geq Eff_{Threshold}, \text{using}\,(7)\\
&\frac{T_{best}}{T_{Up}+T_{Ste}+T_{End}} \geq Eff_{Threshold}, \therefore T_{Up}=T_{End}=0\\
&\frac{T_{best}}{T_{Ste}} \geq Eff_{Threshold}\\
&Eff_{Threshold} \leq \frac{Perf_{Ste}}{CPerf_{Avail}}
\end{aligned}
\qquad \textbf{(60)}
$$

Computers should be released from the external cluster until this performances ratio is greater than the efficiency threshold. Because of clusters heterogeneity several different computers sets might be possible. The best set is the one with the fewer amount of computers because this would lower both the startup and ending times.

If the performances ratio is not greater than the efficiency threshold, even when just the worst computer of the external cluster is selected, then the collaboration of this cluster is not possible.

The next step on the Multi-Cluster Analysis is then the *minimum workload estimation*. This step is similar to the one in the LCA and is done for each external cluster to determine the minimum workload on which the efficiency target is reached.

The following step is then the *multi-cluster load balancing*. The purpose of this step is to statically determine the approximate workload each cluster will be responsible to solve. This is done by balancing each cluster execution time, assuming all the clusters will end up their execution at the same time knowing that the sum of each cluster workload is the total workload of the problem (61).

$$\begin{cases} T_{ex_{Local}} = T_{ex_{Remote1}} = T_{ex_{Remote2}} = \ldots = T_{ex_{RemoteN}} \\ Workload = \sum_{Clusters} Workload \end{cases} \tag{61}$$

At this point each cluster estimated workload and each cluster minimum workload are known. If for any cluster the estimated workload is lower than the minimum workload, this means that for this cluster with the selected computers it is not possible to reach the efficiency target. In this case the methodological process next step is the *resources adjustment*.

At the *resources adjustment* computers should be taken out the execution in order to reach the efficiency threshold. There might be several possible ways to do this adjustment, releasing computers from different clusters. The objective is to find the resources configuration that admits the minimum execution time.

After the *resources adjustment* step, the *minimum workload estimation* and the *multi-cluster load balancing* are re-evaluated until the efficiency threshold is respected.

After the efficiency threshold target is reached, some resources might be left out of the execution. If a better execution time is needed, then the MCA should lead to the Application Tuning. If the execution time is satisfactory then no more analysis are necessary and the methodology reaches its end.

4.5. Application Tuning

The Application Tuning phase is oriented to evaluating the possibility of changing the application to decrease the execution time in the multi-cluster system. The objective is to increase the execution performance using more available resources. The Application Tuning flow diagram can be seen in Figure 4-4.

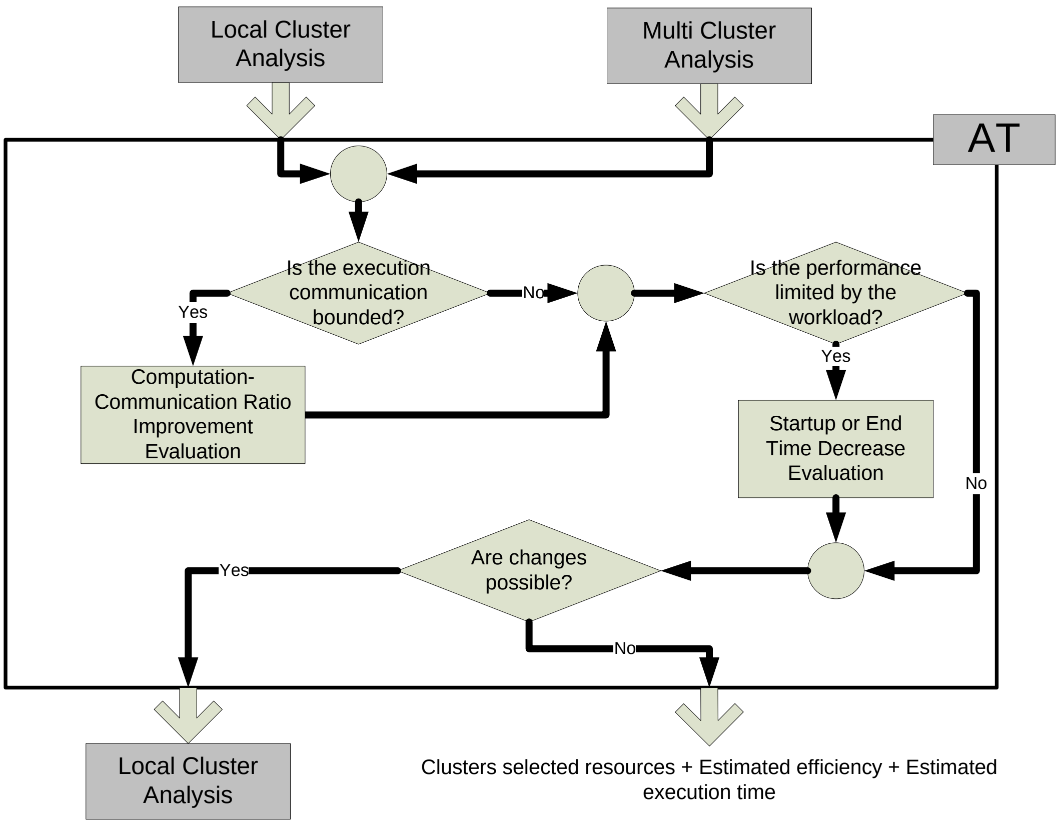

Local Cluster Analysis
Multi Cluster Analysis
AT
Is the execution communication bounded?
Yes
No
Computation-Communication Ratio Improvement Evaluation
Is the performance limited by the workload?
Yes
No
Startup or End Time Decrease Evaluation
Are changes possible?
Yes
No
Local Cluster Analysis
Clusters selected resources + Estimated efficiency + Estimated execution time

Figure 4-4: Application Tuning flow diagram.

Two are the possible causes for releasing resources or clusters from the multi-cluster execution. Firstly the application execution in a cluster might be communication bounded, either in the cluster local area or in the inter-clusters Internet network, and resources were taken out to guarantee the threshold efficiency.

The second reason for releasing resources from an execution is the impossibility of surpassing the efficiency threshold caused by the startup and end time. Even if the application is computation bounded it might happen that the startup and end time together are relatively big enough, when compared to the steady time, to reduce the efficiency down the threshold.

The first step of the Application Tuning phase is to verify if the application is communication bounded in any of the clusters. This information is known from the incoming phase (LCA or MCA) *steady state analysis* step.

If the application is communication bounded then the next step is the *computation-communication ratio improvement evaluation*. At this step first it is evaluated for the communication bottleneck (the LAN, the Internet incoming or the Internet outgoing communication) the possibility of changing the communication granularity, improving the computation-communication ratio in a way it is possible to use more resources from the evaluated cluster.

The *computation-communication ratio improvement evaluation* depends on the application and the possibility of changing the application execution granularity inside a cluster or between clusters. The granularity can be changed, for example, by dividing the workload in smaller-sized or in bigger-sized tasks, by aggregating several tasks in one bigger task or by aggregating several results in one bigger result.

The next step on the AT phase is to verify if the performance is limited by the workload the cluster is responsible for. This information is known from the LCA and MCA *minimum workload estimation* and from the MCA *multi-cluster load balancing* steps.

If for any of the clusters the assigned workload was smaller than the minimum workload necessary for this cluster to surpass the efficiency threshold then resources were taken out of the execution to guarantee the minimum efficiency.

In this case the startup and end time values inhibit the efficiency obtainment and it is necessary to do the *startup and end time evaluation* step. The objective of the *startup and end time evaluation* step is to evaluate the possibility of reducing the size of tasks or results at intra-cluster or inter-cluster levels in order to reduce the startup and end time.

There is a natural tradeoff in the analysis of the Application Tuning phase. For most of the applications the communication-computation ratio will be increased by increasing the task and result sizes. This will cause longer startup and end time for whose smaller tasks and results are necessary to guarantee the efficiency.

It is interesting to use these two analyses to discover the lower and upper granularity bounds for each cluster so that decisions about resources and execution time can be taken.

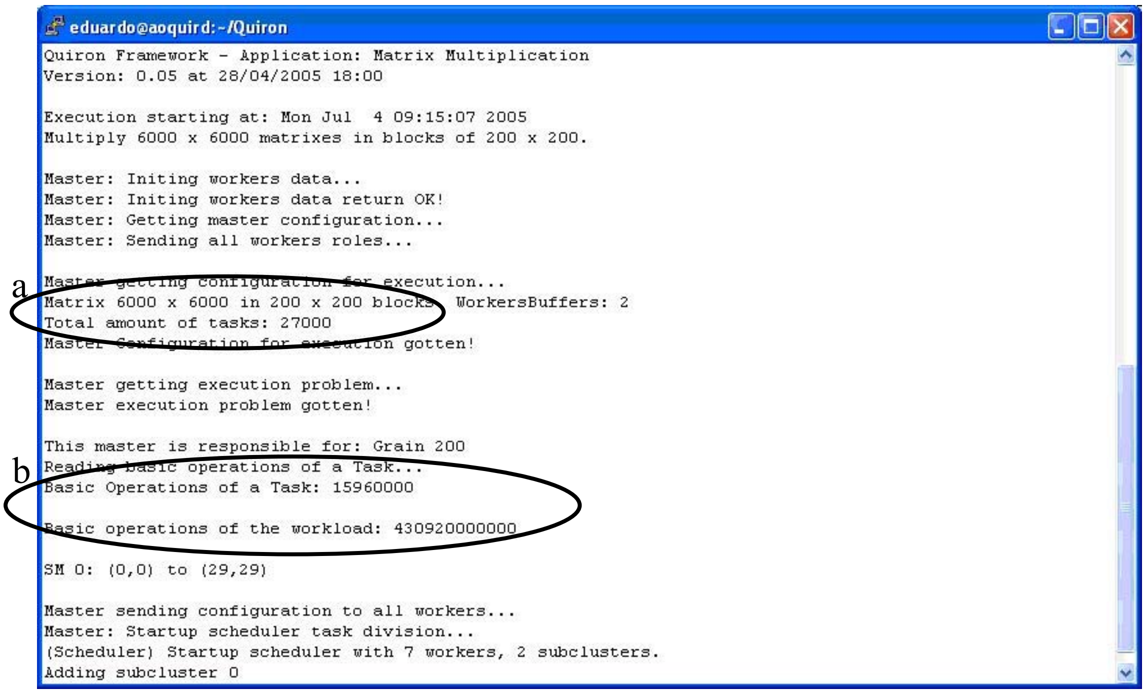
If changes were decided at the AT phase then it is necessary to re-evaluate the application starting by the LCA. In no changes could be made then the methodological processes ends with the results that were reached at the MCA.

4.6. Hierarchical Master-Worker Framework

A hierarchical master-worker framework was developed with the objective of providing tools to support the methodological process, implementing the proposed system architecture performance improvements, and monitoring the analytical model parameters during the execution to provide accurate time prediction.

In order to use the proposed analytical model through the performance prediction and tuning methodology it is necessary to obtain the application and system parameters. Application parameters are related to the application communication and computation needs while system parameters are related to the multi-cluster system computation and communication capacity.

The developed framework provides, at the execution startup or in a workload benchmark previously to the execution, the application parameters such as the *amount of tasks of a workload/problem* and the *basic operations of a workload*. The amount of *basic operations of a task* is application specific and is considered as an input. In case this input is not set the framework performs the measurements considering one task as the basic operation.



```
Quiron Framework - Application: Matrix Multiplication
Version: 0.05 at 28/04/2005 18:00

Execution starting at: Mon Jul  4 09:15:07 2005
Multiply 6000 x 6000 matrixes in blocks of 200 x 200.

Master: Initing workers data...
Master: Initing workers data return OK!
Master: Getting master configuration...
Master: Sending all workers roles...

Master getting configuration for execution...
Matrix 6000 x 6000 in 200 x 200 blocks. WorkersBuffers: 2
Total amount of tasks: 27000
Master Configuration for execution gotten!

Master getting execution problem...
Master execution problem gotten!

This master is responsible for: Grain 200
Reading basic operations of a Task...
Basic Operations of a Task: 15960000

Basic operations of the workload: 430920000000

SM 0: (0,0) to (29,29)

Master sending configuration to all workers...
Master: Startup scheduler task division...
(Scheduler) Startup scheduler with 7 workers, 2 subclusters.
Adding subcluster 0
```

Figure 4-5: Example of the framework workload initialization for a matrix multiplication.

An example of the initialization, using the framework, for a 6,000 x 6,000 matrix multiplication execution in 200 x 200 blocks can be seen in Figure 4-5. In this figure can be seen the *amount of tasks of the workload* (Figure 4-5 a) and the user input value for the *amount of basic operations for a task* as well as the *amount of basic operations for a workload* (Figure 4-5 b).

When an application is written in the framework it is possible to benchmark multi-cluster computers, executing application tasks to obtain each computer *available performance*; and to benchmark local and wide-area networks, achieving the communication throughputs. An example of the computers benchmark results can be seen in Table 4-1.

Table 4-1: Example of the framework benchmark results for the matrix multiplication.

Benchmark Date		Wed Sep 14 20:09:44 2005					
Benchmark Configuration		Matrix 2000 x 2000 in 200 x 200 blocks.					
Machine	**Total Time (sec)**	**Read Time (sec)**	**Write Time (sec)**	**Work Time (sec)**	**Join Time (sec)**	**Other Time (sec)**	**Perf (Tasks/sec)**
aoquir8	216.9150	0.0092	0.0031	216.7773	0.0980	0.0274	4.6130
aoquir3	126.5020	0.0108	0.0043	126.2889	0.1708	0.0273	7.9184
aoquir12	126.5029	0.0115	0.0038	126.2929	0.1659	0.0287	7.9181
aoquir6	87.5568	0.0141	0.0056	85.6038	1.8285	0.1047	11.6817
aoquir9	125.5992	0.0206	0.0073	124.2014	1.2776	0.0923	8.0514
aoquir5	88.4233	0.0138	0.0053	86.2676	2.0378	0.0987	11.5918
aoquir2	138.0031	0.0206	0.0072	136.7841	1.1123	0.0790	7.3108
aoquir4	137.1708	0.0209	0.0072	136.0529	1.0163	0.0735	7.3501
aoquir11	137.6371	0.0207	0.0073	136.4532	1.0857	0.0702	7.3285
aoquir1	137.8504	0.0208	0.0078	136.5439	1.1698	0.1081	7.3237
aoquir7	152.3769	0.0231	0.0079	151.3552	0.9099	0.0809	6.6070

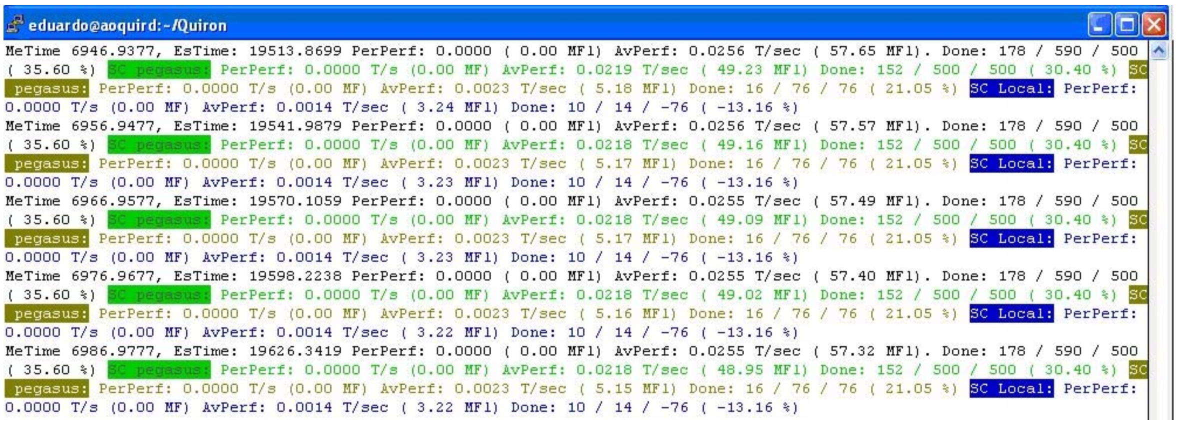
Through the usage of these tools it is possible to, previously to the execution, follow the methodological steps, described at the present chapter, obtaining an execution time and efficiency prediction.

The hierarchical master-worker framework is a set of C++ classes. A parallel master-worker application can be adapted to the framework by adding to it the application specific features: *task execution, results joining* and *task division functionalities* as well as *tasks and results serialization*.

For example, to add a master-worker matrix multiplication in blocks to the framework it would be necessary to add the blocks multiplication algorithm (*task execution*), the matrix sum to join two blocks multiplications (*results joining*), the division of a matrix into blocks representing smaller tasks (*task division*), and the way to read and communicate the matrix blocks (*tasks and results serialization*).

The framework them implements all the hierarchical master-worker functionalities as described in the system architecture. Workers will have two buffers and two threads, one for computation and one for communication. A communication manager process is added to the local cluster for each sub-cluster in the execution and sub-masters receive their work from their communication managers. The framework also enables the usage of different granularities for intra and inter cluster.

During the execution the framework measures the intra and inter-cluster communication throughputs as well as the execution performance. This monitoring feeds the analytical model with information, updating the execution time prediction along the execution. An example of the execution time prediction for the Stochastic Resonant Memory Storage Device (SRMSD) application with three clusters can be seen in Figure 4-6. The multi-cluster statistics are shown in white, while the statistics for each cluster are shown with different colors. The statistics were updated every 10 seconds.

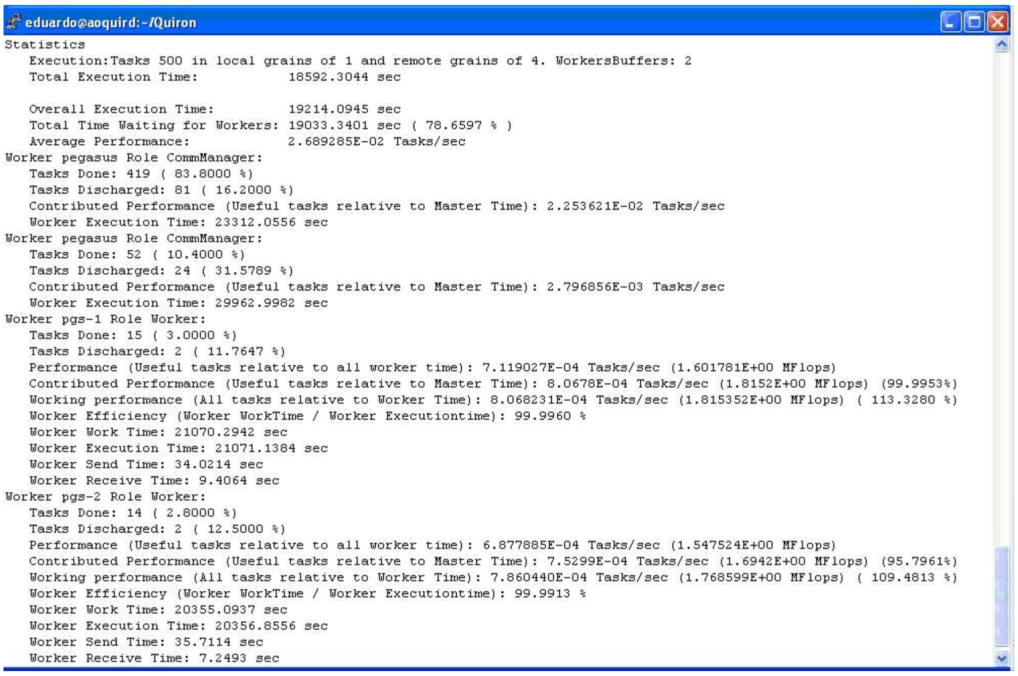


```
MeTime 6946.9377, EsTime: 19513.8699 PerPerf: 0.0000 ( 0.00 MFl) AvPerf: 0.0256 T/sec ( 57.65 MFl). Done: 178 / 590 / 500
( 35.60 %) SC pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0219 T/sec ( 49.23 MFl) Done: 152 / 500 / 500 ( 30.40 %) SC
 pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0023 T/sec ( 5.18 MFl) Done: 16 / 76 / 76 ( 21.05 %) SC Local: PerPerf:
0.0000 T/s (0.00 MF) AvPerf: 0.0014 T/sec ( 3.24 MFl) Done: 10 / 14 / -76 ( -13.16 %)
MeTime 6956.9477, EsTime: 19541.9879 PerPerf: 0.0000 ( 0.00 MFl) AvPerf: 0.0256 T/sec ( 57.57 MFl). Done: 178 / 590 / 500
( 35.60 %) SC pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0218 T/sec ( 49.16 MFl) Done: 152 / 500 / 500 ( 30.40 %) SC
 pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0023 T/sec ( 5.17 MFl) Done: 16 / 76 / 76 ( 21.05 %) SC Local: PerPerf:
0.0000 T/s (0.00 MF) AvPerf: 0.0014 T/sec ( 3.23 MFl) Done: 10 / 14 / -76 ( -13.16 %)
MeTime 6966.9577, EsTime: 19570.1059 PerPerf: 0.0000 ( 0.00 MFl) AvPerf: 0.0255 T/sec ( 57.49 MFl). Done: 178 / 590 / 500
( 35.60 %) SC pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0218 T/sec ( 49.09 MFl) Done: 152 / 500 / 500 ( 30.40 %) SC
 pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0023 T/sec ( 5.17 MFl) Done: 16 / 76 / 76 ( 21.05 %) SC Local: PerPerf:
0.0000 T/s (0.00 MF) AvPerf: 0.0014 T/sec ( 3.23 MFl) Done: 10 / 14 / -76 ( -13.16 %)
MeTime 6976.9677, EsTime: 19598.2238 PerPerf: 0.0000 ( 0.00 MFl) AvPerf: 0.0255 T/sec ( 57.40 MFl). Done: 178 / 590 / 500
( 35.60 %) SC pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0218 T/sec ( 49.02 MFl) Done: 152 / 500 / 500 ( 30.40 %) SC
 pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0023 T/sec ( 5.16 MFl) Done: 16 / 76 / 76 ( 21.05 %) SC Local: PerPerf:
0.0000 T/s (0.00 MF) AvPerf: 0.0014 T/sec ( 3.22 MFl) Done: 10 / 14 / -76 ( -13.16 %)
MeTime 6986.9777, EsTime: 19626.3419 PerPerf: 0.0000 ( 0.00 MFl) AvPerf: 0.0255 T/sec ( 57.32 MFl). Done: 178 / 590 / 500
( 35.60 %) SC pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0218 T/sec ( 48.95 MFl) Done: 152 / 500 / 500 ( 30.40 %) SC
 pegasus: PerPerf: 0.0000 T/s (0.00 MF) AvPerf: 0.0023 T/sec ( 5.15 MFl) Done: 16 / 76 / 76 ( 21.05 %) SC Local: PerPerf:
0.0000 T/s (0.00 MF) AvPerf: 0.0014 T/sec ( 3.22 MFl) Done: 10 / 14 / -76 ( -13.16 %)
```

Figure 4-6: Example of the statistics provided along the execution for the SRMSD simulation with three clusters.

At the execution end the framework provides a summary of workers and network obtained performances enabling the possibility of comparing with the predicted values and analyzing the possible causes of prediction errors. An example for the execution end statistics for the SRMSD simulation with three clusters can be seen in Figure 4-7.

The applications used in this thesis to experimentally validate the precision of the proposed analytical model were adapted to multi-clusters through the use of the developed framework.



```
Statistics
   Execution:Tasks 500 in local grains of 1 and remote grains of 4. WorkersBuffers: 2
   Total Execution Time:          18592.3044 sec

   Overall Execution Time:        19214.0945 sec
   Total Time Waiting for Workers: 19033.3401 sec ( 78.6597 % )
   Average Performance:           2.689285E-02 Tasks/sec
Worker pegasus Role CommManager:
   Tasks Done: 419 ( 83.8000 %)
   Tasks Discharged: 81 ( 16.2000 %)
   Contributed Performance (Useful tasks relative to Master Time): 2.253621E-02 Tasks/sec
   Worker Execution Time: 23312.0556 sec
Worker pegasus Role CommManager:
   Tasks Done: 52 ( 10.4000 %)
   Tasks Discharged: 24 ( 31.5789 %)
   Contributed Performance (Useful tasks relative to Master Time): 2.796856E-03 Tasks/sec
   Worker Execution Time: 29962.9982 sec
Worker pgs-1 Role Worker:
   Tasks Done: 15 ( 3.0000 %)
   Tasks Discharged: 2 ( 11.7647 %)
   Performance (Useful tasks relative to all worker time): 7.119027E-04 Tasks/sec (1.601781E+00 MFlops)
   Contributed Performance (Useful tasks relative to Master Time): 8.0678E-04 Tasks/sec (1.8152E+00 MFlops) (99.9953%)
   Working performance (All tasks relative to Worker Time): 8.068231E-04 Tasks/sec (1.815352E+00 MFlops) ( 113.3280 %)
   Worker Efficiency (Worker WorkTime / Worker Executiontime): 99.9960 %
   Worker Work Time: 21070.2942 sec
   Worker Execution Time: 21071.1384 sec
   Worker Send Time: 34.0214 sec
   Worker Receive Time: 9.4064 sec
Worker pgs-2 Role Worker:
   Tasks Done: 14 ( 2.8000 %)
   Tasks Discharged: 2 ( 12.5000 %)
   Performance (Useful tasks relative to all worker time): 6.877885E-04 Tasks/sec (1.547524E+00 MFlops)
   Contributed Performance (Useful tasks relative to Master Time): 7.5299E-04 Tasks/sec (1.6942E+00 MFlops) (95.7961%)
   Working performance (All tasks relative to Worker Time): 7.860440E-04 Tasks/sec (1.768599E+00 MFlops) ( 109.4813 %)
   Worker Efficiency (Worker WorkTime / Worker Executiontime): 99.9913 %
   Worker Work Time: 20355.0937 sec
   Worker Execution Time: 20356.8556 sec
   Worker Send Time: 35.7114 sec
   Worker Receive Time: 7.2493 sec
```

Figure 4-7: Example of the execution end statistics for the SRMSD simulation with three clusters.

The hierarchical master-worker framework can be divided in three basic modules: *master-worker engine*, *application functionalities* and *I/O middleware*. A layered model of the framework basic modules and their interfaces can be seen at Figure 4-8.

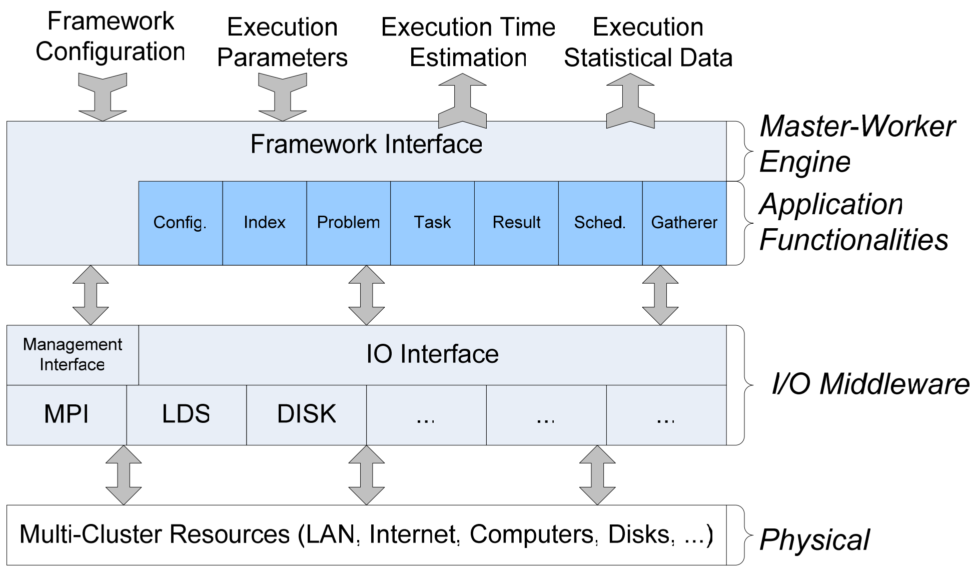


Figure 4-8: Framework layered model and basic modules.

4.6.1. Master-Worker Engine

The master-worker engine provides all the functionalities of the proposed hierarchical master-worker system architecture such as workers pipeline, communication managers, sub-masters buffering, etc.

The framework interface receives the framework configuration and execution parameters that are set in configuration files. The framework configuration corresponds to the *statistics interval*, *worker behavior* and *inter-cluster load balancing*.

The *statistics interval* corresponds to the interval in seconds where the execution progress and time estimation is show in the main master standard output. The value zero to this parameter sets the framework for a "silent" execution.

The *worker behavior* sets if workers will have two buffers and threads, overlapping computation and communication, or if they will have just one buffer and one thread without the overlapping feature.

The *inter-cluster load balance* sets if the framework is responsible for doing the load balance as seen in the Multi-Cluster Analysis or if the user wants to have the load balance programmed in the application functionalities.

The framework configuration is set for the main master in the main cluster only. An example of the configuration file entries for the framework configuration can be seen in Figure 4-9.

Configuration File

```
StatisticsInterval = 10 #10 seconds
WorkerBehaviour = 2     #2 buffers
LoadBalance = True      #FW does the Load balance
```

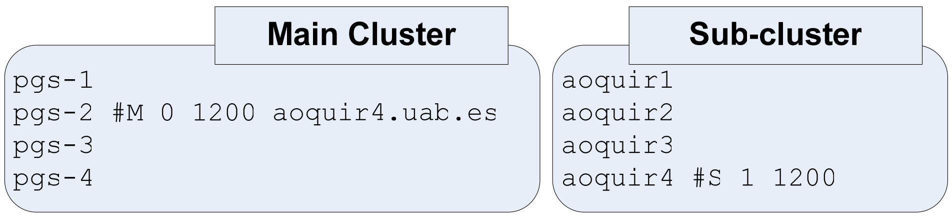
Figure 4-9: Example of the configuration file entries for the framework configuration.

The execution parameters are set for each cluster. The execution parameters indicate the computers names, processes roles and communication manager (CM) configurations. These parameters are set in an enhanced version of the MPI machine file. The first computer in this machine file represents the master. All other computers are workers. For the communication manager some special parameters need to be set following the settings:

```
<local machine name> # <M/S> <0/1> <socket_port> <server machine name>
```

The first communication manager parameter is M/S indicating it is connected to a master or to a sub-master behaving as a worker or as the I/O input respectively. The second parameter 0/1 sets if this communication manager is the socket server or client (0 for socket client and 1 for socket server).The socket port is the number of the port on which the CMs connect and the server machine name is required when the CM is a client, representing the socket server machine this CM needs to connect.

An example of this machine file for a main cluster and a sub-cluster can be seen at Figure 4-10. In this example the machine pgs-2 is the CM for the main master and socket client connecting to the aoquir4 through the port 1200. The machine aoquir4 has the sub-cluster communication manager process and is the socket server.


Main Cluster
Sub-cluster

```
pgs-1
pgs-2 #M 0 1200 aoquir4.uab.es
pgs-3
pgs-4
```

```
aoquir1
aoquir2
aoquir3
aoquir4 #S 1 1200
```

Figure 4-10: Example of the enhanced MPI machine file for a main cluster and a sub-cluster.

4.6.2. Application Functionalities

The application functionalities module represents the code the application programmer needs to pass to the framework for enabling its execution. For adding this code the user must inherit some of the framework classes. These classes might have some standard behaviors that may or may not be changed, depending on the application needs.

The application classes are **configuration**, **index**, **problem**, **task**, **result**, **scheduler**, and **gatherer**. The **configuration** represents the factory class on which the framework is enabled to create instances of all other application functionality classes. There is just one instance of the configuration class and it might contain specific application parameters for the creation of instances. For example in a squared matrix multiplication, the configuration class might have the number of rows and columns a multiplication task and result must allocate.

The **task** class must have the task serialization routine and also the task execution functionality. The **result** class must have the result joining functionality as well as the result serialization routine.

Through the task and result serialization routines it is possible for the framework to set the application communication parameters.

The framework uses a default task and result **index** class. This class associates an integer unique ID value for each task. Optionally the user might inherit the **index** class in order to change a task/result **index** for having for example a row and column number.

The **problem** class has the problem amount of tasks, configuration and division in a multi-cluster. The standard procedure is to divide the problem tasks proportionally to each cluster. If another concept is applied to this division, like different granularities for intra and inter-cluster communication, it is necessary to inherit the **problem** class and change the problem division behavior.

The **scheduler** is responsible for dividing the **problem** into **tasks** selecting the **task** to be sent to each worker. The division of the **problem** needs to be defined and from this division the framework obtains the application computation parameters.

The default functionality for the **scheduler** to distribute tasks is to select the next task for the next worker without making differences between workers. It is possible to change this functionality by rewriting the selection method of the **scheduler**.

The **gatherer** is responsible for gathering task results, joining them and checking for the end of the algorithm when all tasks are done. It is also possible to change its behavior.

4.6.3. I/O Middleware

All the I/O for the framework is made through a standard middleware interface that is translated to the necessary I/O system device. With this it is possible for the framework to perform the I/O in the correct device depending on the necessity. The Internet communication in the CM is done through the LDS library while the results storing in the disk is done through the disk device. The communication in a LAN is done through MPI calls. The application programmer describes the I/O through standard calls to the I/O middleware.

Through this I/O structure the framework is able to obtain the application communication parameters. The basic I/O interface has functions to open and close devices, store and load data from the devices. The I/O functions to the programmer and its description are illustrated at Table 4-2.

Table 4-2: Illustration of the I/O middleware interface for the application classes' serialization.

I/O Function	Description
OpenDevice	Opens a device. Depending on the received parameters it opens a file, an MPI communication, or a LDS communication.
CloseDevice	Closes and specific device.
CloseAllDevices	Closes all opened devices.
ReadBlock	Reads a data block from a device.
WriteBlock	Writes a data block into a device.

The I/O interface also provides the master-worker engine with functionalities for serializing objects and getting statistics. These functions are illustrated at Table 4-3.

Table 4-3: Illustration of the I/O middleware interface for the master-worker engine.

I/O Function	Description
Load	Calls the serialization function for loading the data of a certain serializable object.
Store	Calls the serialization for storing the data of a certain serializable object
GetDeviceStatistics	Get the actual time statistics for a device: time spent reading and writing data in the device.
GetDeviceVolStatistics	Function to get the data volume statistics for the device: Amount of bytes read or written in the device.

4.7. Conclusions

In this chapter a performance prediction and system tuning methodology was proposed and its three phases were described. The objective of this methodology is to use the analytical model to evaluate an application execution in a multi-cluster environment, estimating the execution time and efficiency and selecting the clusters and computers that should be used at this execution. The resources selection is done in a way that a minimum efficiency is guaranteed.

The methodology was divided in three related phases: Local Cluster Analysis, Multi-Cluster Analysis and Application Tuning. The Local Cluster Analysis is responsible for evaluating if the local cluster is apt to receive, for this execution, extra performance from external clusters.

The Multi-Cluster Analysis evaluates each external cluster and selects the ones that can collaborate in the execution keeping the system efficiency over a threshold. If it is desirable a better execution time than the estimated by the LCA and MCA phases the methodology needs to go to the Application Tuning phase.

At the Application Tuning the possible causes of the release of resources are analyzed and changes on the application communication granularity that could lead to the usage of more resources are evaluated.

If changes are possible then the methodological steps are reevaluated and a new resources selection, efficiency and execution time estimation is done.

It was also presented in this chapter a hierarchical master-worker framework that provides tools to the methodological steps. The developed framework implements the proposed system architecture performance enhancements and provides the application and system parameters as well as execution time predictions based on the analytical model.

Chapter 5

Experimental Validation

5.1. Introduction

This thesis studies the problem of improving the execution performance of parallel applications adapting them from their original single cluster to a heterogeneous multi-cluster environment. The goal is to reduce the execution time guaranteeing a minimum level of efficiency on both local and external resources usage.

In order to reach this target, at Chapter 2 it is proposed an architecture that can efficiently interconnect clusters providing reliability to Internet communication and allowing a multi-cluster to be seen as a single system. An analytical performance model was then developed in Chapter 3, based on this system architecture.

From this analytical model it is possible to, providing some computation and communication parameters of an application and of a target multi-cluster, evaluate the application efficiency and execution time for a certain problem. This model also gives support to the selection of resources for which efficiency can be improved.

At Chapter 4 are described the methodological steps on which an application originally written for a single cluster can be evaluated and adapted to a multi-cluster. The methodological phases guide the obtainment of application parameters, the usage of the analytical model to evaluate resources, efficiency and execution time in the multi-cluster, and the tuning process.

The performance analysis and system tuning methodology can be used to predict the execution time and efficiency for an application execution in a multi-cluster. Another possible use of the methodology is to support the selection of resources for which an execution will take place with efficiency over a threshold. The methodology can also be used to assist the application tuning process to improve the level of usability of the multi-cluster.

The objective of the present chapter is to validate the analytical model and methodological approach. For doing this, three applications were chosen: the matrix multiplication (MM), the Stochastic Resonant Memory Storage Device (SRMSD) simulation and the traveling salesman problem (TSP).

Throughout this chapter, each application implementation for a single cluster is described. The applications are adapted to the multi-cluster through the usage of the hierarchical master-worker framework presented at Chapter 4.

The matrix multiplication (MM) is the first evaluated application and represents an important algebra kernel [35]. The MM is used as an initial benchmark application selected because of its scalability, known computation and communication volume as well as high communication requirement for multi-cluster collaboration.

There are many possible execution variations for the MM within quite different computation communication ratios and granularities. The matrix multiplication is briefly analyzed through the methodological steps in order to verify the precision of the analytical model predictions and its capability on supporting an application design and granularities selection.

The Stochastic Resonant Memory Storage Device (SRMSD) simulation application [28] represents a complex numerical simulation to study the response of a system of bistable oscillators coupled unidirectional driven by a source of external noise and a periodic and temporal stimulus.

This application is composed by a few – in the order of hundreds – long-duration atomic simulations on which each simulation's result represents a large data communication packet of 2.256 MBytes. Each simulation task may last for hundreds up to thousands of seconds. For this application the balance of load becomes an important issue.

For this application a problem and a target multi-cluster are presented and the methodological process phases for this application/multi-cluster scenario are explained step by step. The target is to verify the methodology capabilities of prediction, resources selection for guaranteeing an efficient execution as well as performance tuning.

The traveling salesman problem (TSP) application was selected to prove the analytical performance model robustness. Three different algorithms were developed for the TSP: *exhaustive*, *local pruning* and *global pruning*. These algorithms differ on the level of predictability of tasks execution time.

5.2. Matrix Multiplication

The matrix multiplication (MM) is a key operation of the linear algebra kernel [34][35] used by a wide range of scientific applications. There are efficient solutions for the MM problem over homogeneous processors, e.g. ScalaPACK [30]. However, the MM execution over different-speed processors turns out to be surprisingly difficult. Actually, its NP-completeness over heterogeneous platforms was proved [19]. There are studies for the MM execution optimization over heterogeneous processors as seen in [20].

The analysis done in this thesis does not intend to be a specific optimization study for the matrix multiplication. For this thesis the MM represents a highly scalable and fine-grained application that present scalability difficulties for heterogeneous multi-clusters. No optimization library, like ATLAS [L1], was used for the matrix multiplication codification present in this work.

A master-worker blocked algorithm is the studied approach for this thesis. The main master has two operands M x M matrixes and one result M x M matrix. For our approach no distribution of any of the operand matrixes is done previously to the execution.

A blocked multiplication algorithm was chosen because it does not require previous distribution of any of the operand matrixes. This approach is also highly adaptable to heterogeneous computation. For the blocked multiplication, the three M x M operand matrixes are organized in B x B blocks as shown in Figure 5-1.

The master sends to workers a block from the first operand matrix and a block from the second operand matrix representing a task. Workers multiply the block matrixes and send back to the master the blocked multiplication result that is joined (summed) in the result matrix (Figure 5-2). The master repeats this process until every first operand blocked row and second operand blocked column are multiplied and the final result matrix is achieved.

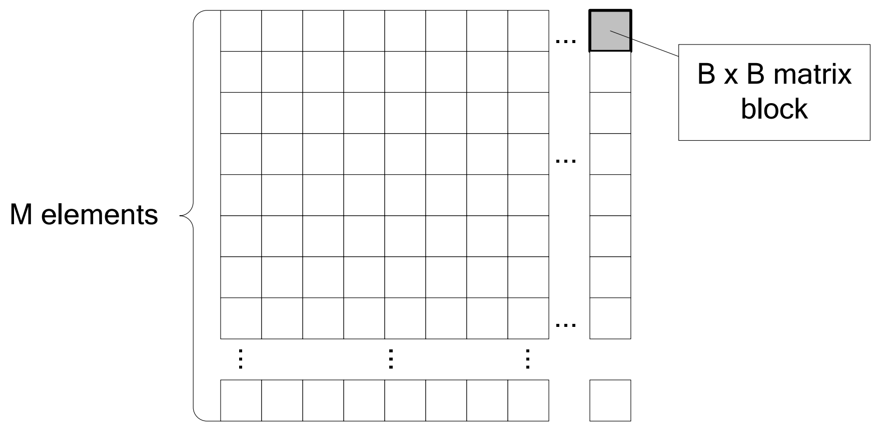


Figure 5-1: M x M elements matrix divided in B x B blocks.

This multiplication approach allows the analysis of large matrixes multiplication (like 10,000 x 10,000) with fine-grained tasks (blocks of 100 x 100). It is a challenge to attain external clusters collaboration in a fine-grained application like the matrix multiplication.

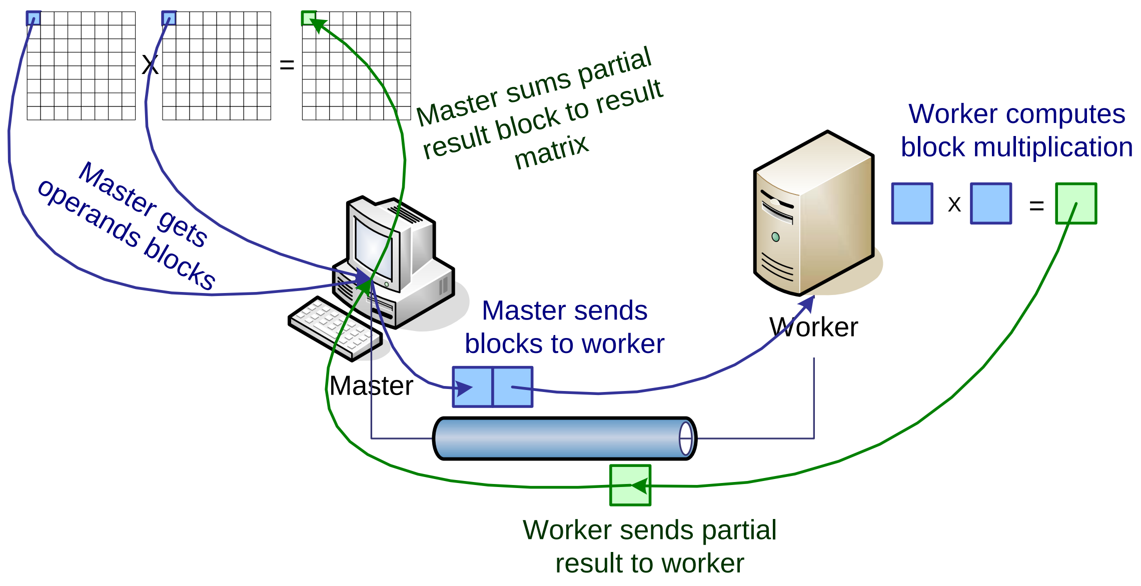

X
=
Master sums partial result block to result matrix
Worker computes block multiplication
x
=
Master gets operands blocks
Master sends blocks to worker
Worker
Master
Worker sends partial result to worker

Figure 5-2: Schematic diagram of a master sending a task of matrix blocks to a worker and receiving from this worker a block result.

This section briefly describes each methodological phase for the matrix multiplication, focusing on the main challenges for its execution using efficiently the available resources. The target multi-cluster is formed by two clusters: the first cluster is considered the local cluster and is located in Brazil while the external cluster is located in Spain.

5.2.1. Local Cluster Analysis

The blocked approach to solve the matrix multiplication enables the division of the problem in several different granularities, i.e., it is possible to divide the M x M matrixes in different B x B sized blocks.

There are then two possibilities to select a granularity for the execution in a cluster. The first possibility is to arbitrarily choose a granularity and, if this granularity does not allow an efficient usage, adjust it in the Application Tuning phase.

The second possibility is to select some different granularities and evaluate the effect of the granularity in the steady performance and system efficiency. Through this analysis the best granularity for a cluster can be inferred. This second analysis was chose because of its completeness.

In this section we briefly determine the application and system parameters and then we perform the granularity analysis.

Application and System Parameters Obtainment

The main determinant basic operation of the matrix multiplication is the float point operation (either multiplication or addition). All the performance measurements will then be done in float point operations per second (FLOPS) or in million of FLOPS (MFLOPS).

The application parameters are the *amount of tasks of a workload*, the *amount of basic operations of a task* and the *amount of basic operations for a workload*. For the MM, a task is the multiplication of two B x B sub-matrixes.

The *amount of tasks of a workload* is the amount of B x B blocks multiplications of the M x M operand matrixes. Each matrix row has K=M/B blocks. The result matrix has K^2 blocks that are the result of row per column blocked multiplications. Each row per column multiplication has K blocked operations. From this it is possible to deduce that the amount of tasks is K^3 (62). The value of B represents the granularity of the task.

$$Tasks(M,B) = K^3 = \left(\frac{M}{B}\right)^3 \tag{62}$$

In the process of multiplying two B x B matrixes, the result matrix is composed by B^2 line per columns multiplications. Each line per column multiplication has B multiplications and B – 1 additions, meaning 2*B – 1 float point operations. This means that the *amount of basic operations of a B x B blocked task* in is given by equation (63).

$$Oper_{Task}(B) = 2 * B^3 - B^2 \tag{63}$$

The *amount of basic operations for an M x M matrix workload divided in B x B blocks* in our master-worker approach is then the total amount of tasks multiplied by each task amount of basic operations (64).

$$Workload(M, B) = \left(\frac{M}{B}\right)^3 * \left(2 * B^3 - B^2\right) = M^3 * \left(2 - \frac{1}{B}\right) \tag{64}$$

On the communication side, the amount of bytes necessary to be sent for a B x B multiplication (*size of a task*) is two times the B x B matrix of float points. Considering that a float point has α bytes, the *size of a task* (S_{Task}) is given by equation (65). A task result data has one B x B block; the *size of a result* (S_{Result}) is then equation (66). The *total communication of a task* (S_{Comm}) (67) is the sum of these values.

$$S_{Task}(B) = 2 * \alpha * B^2 \tag{65}$$

$$S_{Result}(B) = \alpha * B^2 \tag{66}$$

$$S_{Comm}(B) = 3 * \alpha * B^2 \tag{67}$$

The granularities B=25, 50, 100, 200, 300, 400, 500 and 1000 were selected for the local-cluster performance analysis. The system parameters were measured for these granularities. Table 5-1 shows the Brazilian cluster local area network throughput for the different granularities with different amount of workers. The average value of 1,012,391 bytes/sec is used for the Brazilian cluster estimations.

Figure 5-3 shows the Brazilian *cluster available performance* for the MM application with different granularities. It can be seen that the performance decreases for bigger grains. This can be explained with the fact that the increment of the granularity implies a squared increase on the size of the task which might improve the cache misses.

Table 5-1: Average throughput in bytes/sec for the Brazilian cluster LAN with different amount of workers and different granularities.

Workers	100 x 100	200 x 200	300 x 300	400 x 400	500 x 500	Average
1	1,075,341	1,042,389	1,051,342	1,035,495	1,037,723	1,048,458
2	992,490	981,755	988,112	988,755	998,153	989,853
3	995,754	982,516	1,002,688	1,006,843	1,004,562	998,472
4	1,026,582	1,005,519	1,011,797	994,758	997,432	1,007,218
5	1,012,046	1,016,555	1,018,914	1,005,807	1,019,032	1,014,471
6	1,001,692	1,010,856	1,023,699	1,024,551	1,018,561	1,015,871
Average	1,017,318	1,006,598	1,016,092	1,009,368	1,012,577	**1,012,391**

For the matrix multiplication, the steady performance depends on the granularity and can be defined as equation (68). The performance for which the execution becomes computational bounded at the LAN is proportional to the B block index. This can be explained because the MM computation has cubic and the communication has squared complexity.

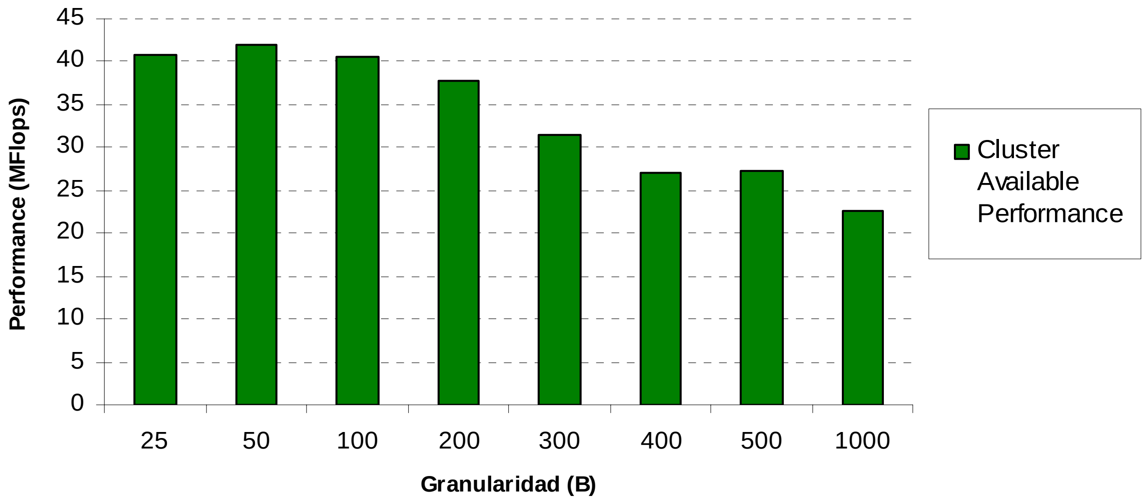


Figure 5-3: Brazilian cluster available performance for different granularities.

(68)

$$Perf_{Ste} = \min\left(CPerf_{Avail}, Oper * \frac{TPut_{LC}}{S_{Comm}} \right)$$

$$Perf_{Ste} = \min\left(CPerf_{Avail}, (2*B^3 - B^2) * \frac{TPut_{LC}}{(3*\alpha*B^2)} \right)$$

$$Perf_{Ste} = \min\left(CPerf_{Avail}, (2*B-1) * \frac{TPut_{LC}}{3*\alpha} \right)$$

$$\therefore \left\langle \begin{array}{l} TPut_{LC} = 1{,}012{,}391 \text{bytes/sec} \\ \alpha = 4 \text{bytes} \end{array} \right.$$

Figure 5-4 shows the Brazilian *cluster available performance*, expected steady performance and maximum cluster efficiency for the different studied granularities. The steady time is communication bounded for the studied granularities up to B=200. From B=300 on the steady time is computation bounded.

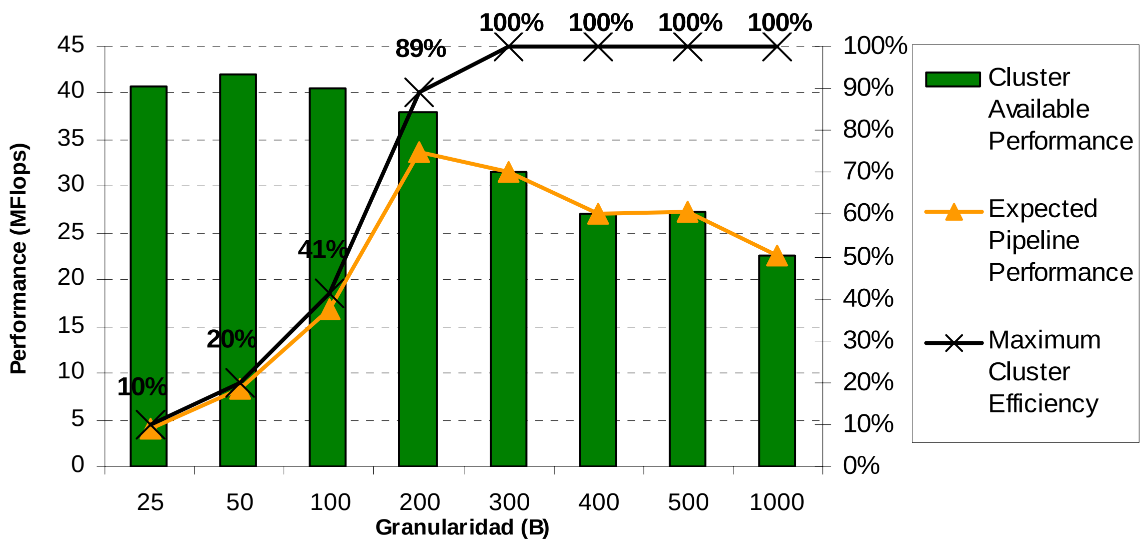


Figure 5-4: Brazilian cluster available performance for different granularities.

Figure 5-5 shows the estimated startup and worst end time for the different granularities in the Brazilian cluster. It can be seen that an increase in the granularity causes an increment of these times. The best granularity is the smaller one on which the application is computation bounded.

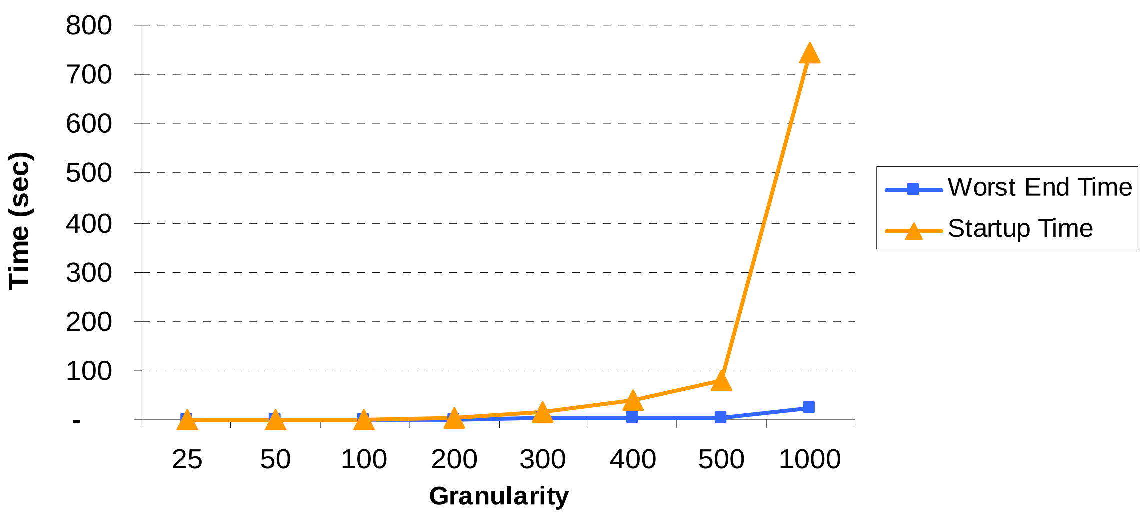


Figure 5-5: Startup and worst end time for different granularities in the Brazilian cluster.

5.2.2. Multi-Cluster Analysis

Once the Brazilian cluster parameters were obtained, it is time to analyze the system parameters for the Spanish cluster. The Spanish *cluster available performance*, steady performance and local maximum efficiency, for the MM application can be seen in Figure 5-6. The Spanish cluster is computation bounded in the local area network for granularities greater than or equal to B=400.

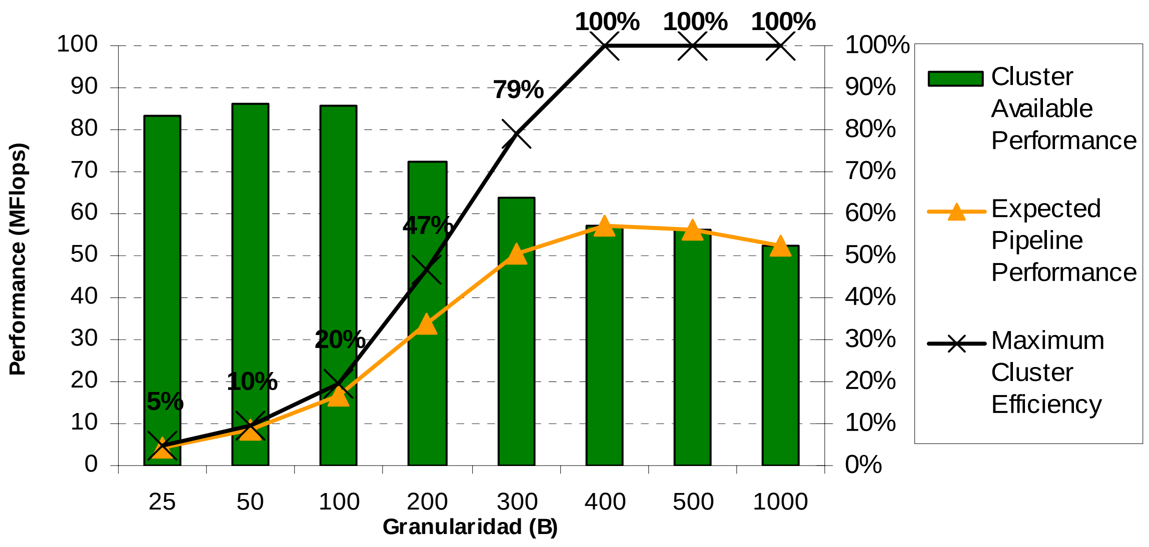


Figure 5-6: Available performance, expected steady performance and maximum efficiency for different granularities locally in the Spanish cluster.

The obtained experimental Internet throughput values for the Spanish cluster relative to the Brazilian cluster was of 53.95 Kbytes/sec with no significant difference for incoming or outgoing.

The objective now is to evaluate the possible steady performance for the Spanish cluster, considering also Internet incoming and outgoing communication. The expected steady performance for the Spanish cluster is obtained by the equation (69).

$$Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ Oper * \frac{TPut_{EC}}{S_{Comm}}, \\ Oper * \frac{TPut_{InetIN}}{S_{Task}}, \\ Oper * \frac{TPut_{InetOUT}}{S_{Result}} \end{pmatrix} \quad Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ (2*B-1)*\frac{TPut_{EC}}{3*\alpha}, \\ (2*B-1)*\frac{TPut_{InetIN}}{2*\alpha}, \\ (2*B-1)*\frac{TPut_{InetOUT}}{\alpha} \end{pmatrix} \qquad \textbf{(69)}$$

Since the incoming and outgoing Internet throughputs are similar, the steady performance is limited by the incoming Internet communication. The graphic in Figure 5-7 shows the *cluster available performance*, the expected steady performance (result of the equation (69)) and the maximum cluster efficiency for the Spanish cluster as an external cluster.

It can be concluded that the application is communication bounded for all the studied granularities. The granularity B=400 is the smaller studied one that is computational bounded for both clusters. This granularity is the selected one for intra-cluster computation because bigger granularities would represent longer startup and end times.

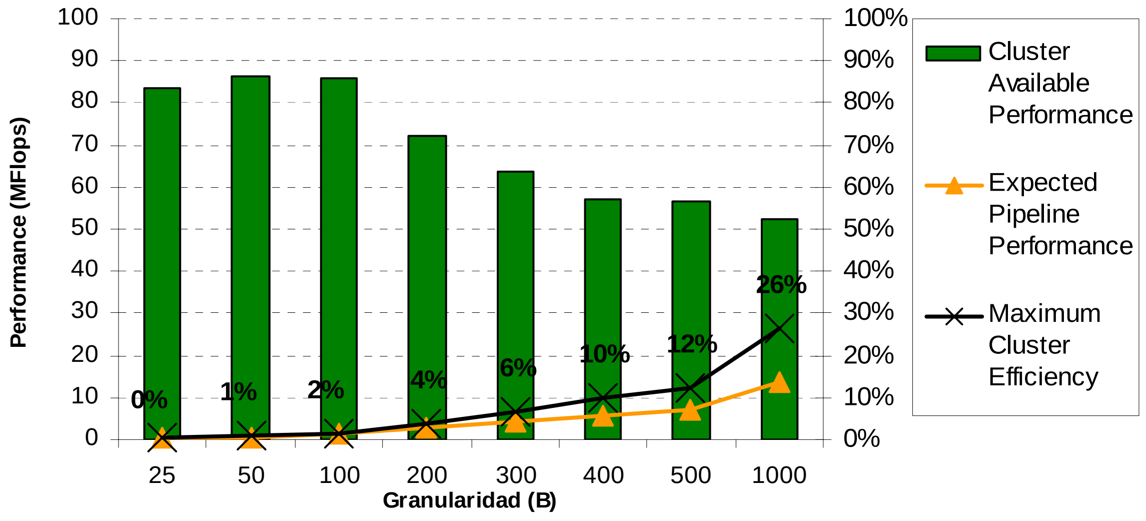


Figure 5-7: Spanish cluster available performance, expected steady performance and maximum efficiency for different granularities.

For the B=400 granularity it is not profitable the Spanish cluster collaboration. There are no possible worker configurations for which the execution would be over an efficiency threshold of 80%.

The alternative of increasing the granularity until the application becomes computational bounded considering the Internet communication is not a viable because of two reasons. Firstly the workers performance would be drastically reduced because of the size of the task. A task of B=2000 grain would be approximately 46 Mbytes. Secondly, the size of the task would also drastically reduce the efficiency because of the high startup and end times. It is then necessary to tune the application for the multi-cluster.

5.2.3. Application Tuning

The objective of the MM Application Tuning is to increase the computation-communication ratio for the inter-cluster communication. In order to do that two different inter-cluster distribution strategies are analyzed. Both strategies try to take advantage of the MM cubic computation complexity against its squared communication complexity.

The first analyzed strategy is a generalization of the blocked approach, increasing the inter-cluster granularity by sending to the Spanish cluster tasks with matrixes of G x G elements, being G a multiple of B. The second strategy is to send lines and columns of blocks of the operand matrixes to the sub-cluster. This strategy dynamically improves the inter-cluster granularity by the use of the data locality.

The second strategy is the one selected for the inter-cluster distribution because it is the one with lower startup and end time. The following sections explain each strategy showing how the analytical model can be used to compare different tuning possibilities.

Increase Inter-Cluster Matrix

The objective of this strategy is to increase the external cluster steady performance by sending a bigger inter-cluster multiplication G x G.

It is possible to calculate the minimum inter-cluster granularity value assuming that at the equation (69) the Internet incoming communication bound performance for the Spanish cluster, with the inter-cluster G granularity, needs to be greater than or equal to the Spanish *cluster available performance* (70).

$$(2*G-1)*\frac{TPut_{InetIN}}{2*\alpha} \geq CPerf_{Avail} \tag{70}$$

The minimum inter-cluster granularity G for reaching at the steady time the Spanish available performance is then given by equation (71).

$$G \geq \frac{\alpha * CPerf_{Avail}}{TPut_{InetIN}} + \frac{1}{2} \tag{71}$$

Applying this equation to the Spanish cluster (72) we obtain the minimum value of G=4,123. The next B=400 multiple is the value of G=4,400. This means that the inter-cluster tasks have to be in the minimum blocks of 4,400 x 4,400 to be able to attain the totality of the Spanish cluster performance at the steady time.

$$G \geq \frac{\alpha * CPerf_{Avail}}{TPut_{InetIN}} + \frac{1}{2} \Rightarrow G \geq \frac{4 * 56{,}939{,}364}{55{,}245} + \frac{1}{2} \Rightarrow G \geq 4123.18 \tag{72}$$

A 4400 x 4400 matrix has approximately 148Mbytes of size, increasing the startup (21) and end times (55) to an order of 45 and 24 minutes respectively. Using the analytical model we can calculate the minimum workload of 6 inter-cluster tasks for reaching a minimum of 80% efficiency from the Spanish cluster.

Increase Inter-Cluster Matrix

Another possible data distribution strategy for the matrix multiplication is to send to external clusters complete operands' sets of rows/columns. Each new row/column (r/c) pair that reaches the sub-master can be computed with the previously received columns/rows.

The granularity of the inter-cluster tasks continuously grows along time the more rows/columns are sent to the external cluster. An example of the growing granularity can be seen in Figure 5-8. When the first r/c arrives it is possible to multiply one row of blocks by one column of blocks. Once the second r/c arrives it is possible to perform four blocked row per column multiplications. For the third r/c, nine row per columns multiplications can be done and this ratio increases the more data arrives.

This approach has some advantages when compared with the strategy of increasing the inter-cluster granularity. The first advantage is that once a row/column arrives in the external cluster, this cluster can already perform some computation even though with reduced efficiency until the cluster available performance can be reached.

Another advantage is that the external cluster result is an already computed row per column multiplication, improving the end time. This result represents just one 400 x 400 matrix block for which the size is quite small when compared to a 4400 x 4400 bigger granularity block.

This would reduce the end time from 20 minutes to 30 seconds requiring a smaller minimum workload to obtain the desired efficiency for 400 x 400 blocks independently of the matrix size.

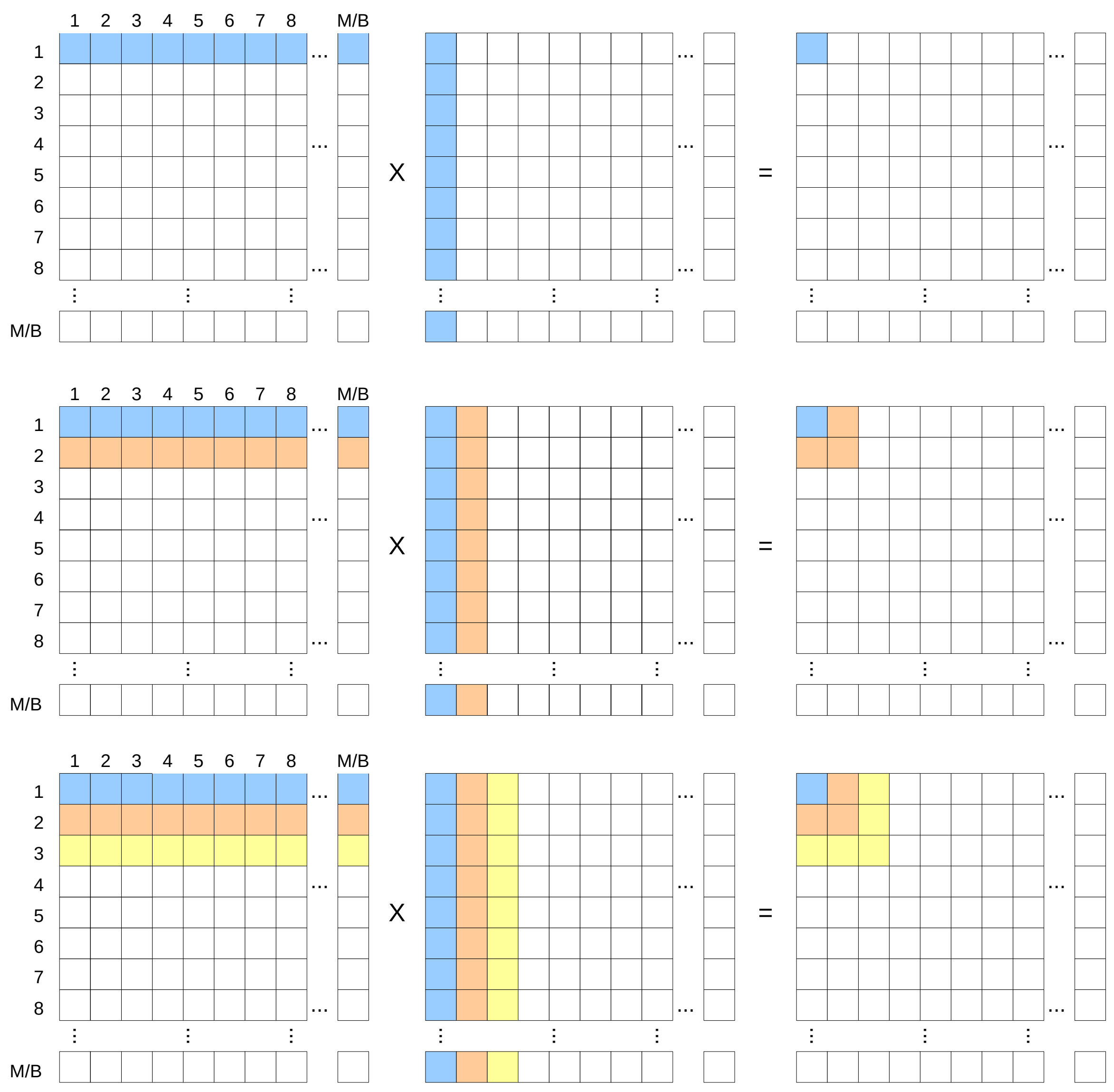


Figure 5-8: Schematic diagram of the grown in the granularity by the strategy of sending operands row/columns.

5.2.4. Multi-Cluster Analysis – Tuned Application

Considering the modified inter-cluster distribution, the application parameters for the communication between clusters have changed. An inter-cluster task can be redefined as being one row and column while the row per column multiplication can be considered as being an inter-cluster result.

As shown at Figure 5-8, the first row/column sent to an external cluster represents one row per one column blocks multiplications. This means that the first inter-cluster task carries M/B (the amount of blocks in a row) tasks to be solved by the external cluster workers.

When the second r/c is received by the external cluster, three new rows per columns multiplications are possible, as illustrated with the orange color in the Figure 5-8. When a RC row/column is received it is possible to do 2*RC – 1 row per columns multiplications.

It is possible to conclude that the RC inter-cluster task has (73) operations. The *amount of operations* of an inter-cluster task grows linearly the more tasks are sent and can be defined by (74).

$$Oper_{Inter}(RC, M, B) = 2 * (RC - 1) * \frac{M}{B} * Oper(B) \tag{73}$$

$$Oper_{Inter}(RC, M, B) = 2 * (RC - 1) * M * (2 * B^2 - B) \tag{74}$$

The *size of a task* is the amount of bytes of one row of blocks plus one column of blocks. Both a row and a column of blocks have M x B float point elements; the *size of a task* is then (75). Similarly to an intra-cluster result, an inter-cluster result is one B x B block (76). Each sent result carries one row per column multiplication operations (77).

$$S_{Task}(B) = 2 * \alpha * M * B \tag{75}$$

$$S_{Result}(B) = \alpha * B^2 \tag{76}$$

$$\begin{aligned} Oper_{InterResult}(M, B) &= \frac{M}{B} * Oper(B) \\ Oper_{InterResult}(M, B) &= \frac{M}{B} * (2 * B^3 - B^2) \\ Oper_{InterResult}(M, B) &= M * (2 * B^2 - B) \end{aligned} \tag{77}$$

For this MM distribution, different amount of operations are used for intra-cluster, incoming and for outgoing communication bounded limits.

Once defined the application parameters it is possible to analyze the steady performance. The steady performance for the Spanish cluster is then given by the equation (78).

For a 10,000 x 10,000 multiplication in 400 x 400 blocks, for example, the Spanish cluster steady performance would be of (79). The obtainable performance is mainly limited by the incoming communication and it is possible to find out for which row/column the application would become computational bounded (80).

(78)

$$Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ (2*B^3 - B^2)*\dfrac{TPut_{EC}}{3*\alpha*B^2}, \\ 2*(RC-1)*M*(2*B^2-B)*\dfrac{TPut_{InetIN}}{2*\alpha*M*B}, \\ M*(2*B^2-B)*\dfrac{TPut_{InetOUT}}{\alpha*B^2} \end{pmatrix}$$

$$Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ (2*B-1)*\dfrac{TPut_{EC}}{3*\alpha}, \\ (RC-1)*(2*B-1)*\dfrac{TPut_{InetIN}}{\alpha}, \\ M*(2*B-1)*\dfrac{TPut_{InetOUT}}{\alpha*B} \end{pmatrix}$$

(79)

$$Perf_{Ste} = \min\begin{pmatrix} 56.939x10^6, \\ 67.408x10^6, \\ (RC-1)*11,035x10^6, \\ 275.878x10^6 \end{pmatrix}\text{MFlops}$$

(80)

$$(RC-1)*11,035x10^6 \geq 56.939x10^6$$
$$RC \geq 6.1$$

In this example after the 7th inter-cluster task the external cluster would become computation bounded and the available performance would be reached. To guarantee a minimum efficiency it is necessary to evaluate this startup time and the end time, reaching the minimum workload or selecting resources.

The startup time is the time spent to send the row/columns until the cluster available performance is reached. The best and worst end times can be calculated by the second external cluster studied scenario equations (53) and (55) respectively. For a 10,000 x 10,000 matrix in 100 x 100 blocks the best end time would be 0.46 seconds and the worst time 0.68 seconds.

5.2.5. Experimental Validation

The objective with the matrix multiplication was to analyze a complex fine grained application for which it was necessary a different inter-cluster distribution strategy to attain an efficient collaboration. This different strategy implies in different parameters for the multi-cluster communications. The simple computation-communication analysis, that is the base of the analytical model, could be used for achieving more complex information like the minimum row column for which the steady performance reaches its maximum.

The designed experiments aim to validate the performance analysis of the MM application in two levels: Firstly at the intra-cluster level, demonstrating the necessity of the local cluster granularity analysis. The selection of a computation bounded grain avoids the local area network saturation and the consequent efficiency reduction.

Secondly at the inter-cluster level, demonstrating that it is possible to predict the application performance behavior along time, the moment where the pipeline gets in its best performance, the execution time and efficiency.

Intra-Cluster Validation

The Brazilian and Spanish cluster expected performance (26) in the local MM execution for different granularities can be seen in Figure 5-4 and Figure 5-6 respectively.

In order to validate this analysis several executions of the parallel MM application at both clusters with the selected granularities and different matrixes sizes were done. The comparison between available, expected and obtained steady performance for the Brazilian and Spanish clusters can be seen in Figure 5-9 and Figure 5-10 respectively.

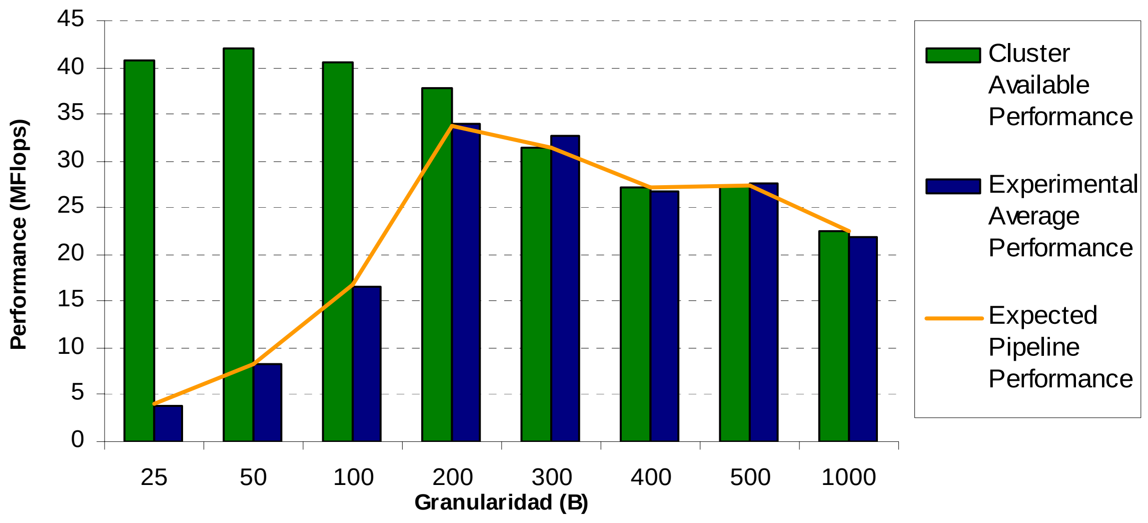


Figure 5-9: Brazilian cluster's available, expected and experimentally obtained steady performance for different granularities.

The prediction is over 98% precision for both the computation and communication bounded granularities.

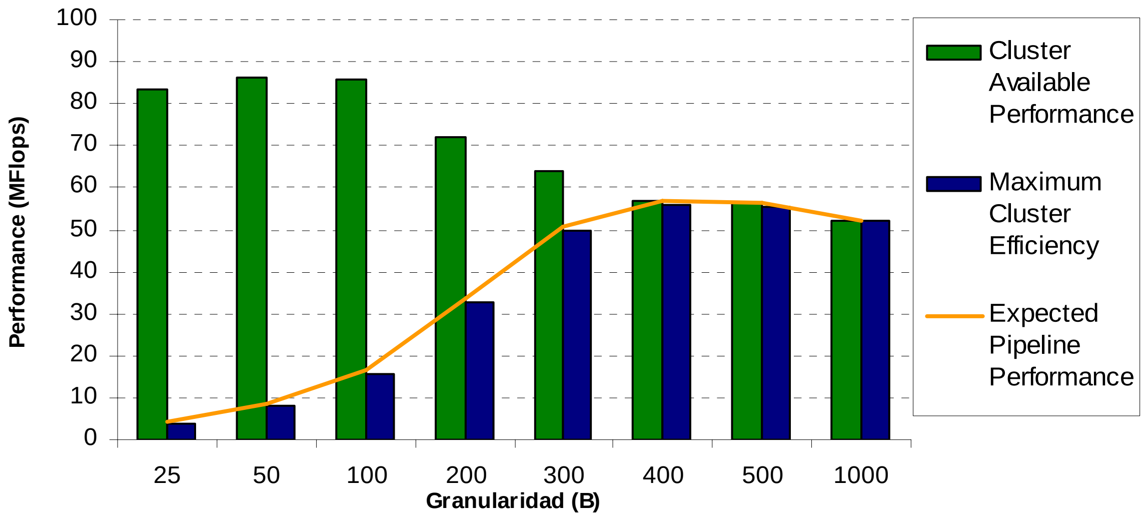


Figure 5-10: Spanish cluster's available, expected and experimentally obtained steady performance for different granularities.

Inter-Cluster Validation

For the inter-cluster validation two experiments are analyzed in details. For the first experiment a 10,000 x 10,000 multiplication is done in blocks of 100 x 100 elements. In this experiment both clusters steady state are communication bounded causing a reduction in the system efficiency.

For the second experiment the 10,000 x 10,000 matrixes are multiplied in 400 x 400 blocks. This experiment targets a minimum of 80% efficiency using all resources of both clusters.

The analytical model was used to predict the efficiency and execution time for both experiments. The model was also used to determine the amount of row/columns needed to be sent to the Spanish cluster in order to turn its execution into computation bounded and this startup time.

The 10,000 x 10,000 multiplication in 100 x 100 blocks experiment prediction data can be seen at Table 5-2. From this data it can be inferred that because of the fine-grained inter-clusters communication the best and worst predicted times are very similar.

Table 5-2: Prediction information for a multi-cluster execution of a 10,000 x 10,000 matrix in 100 x 100 blocks.

	Startup		Estimated Execution Time (sec)		Estimated Efficiency	
Cluster	Row / Columns	Time (min)	Best	Worst	Best	Worst
Brazil		0.004	16h 41'24sec	16h 41'24sec	61.51%	61.51%
Spain	7	16.89	16h 41'24sec	16h 41'25sec	28.48%	28.48%
Multi-Cluster			**16h 41'24sec**	**16h 41'25sec**	**39.13%**	**39.13%**

The Spanish average performance along time grows linearly through the startup phase until the 7th row/column arrives after 16.89 min. At this moment the pipeline stabilizes in the expected steady performance until the execution end. The expected execution performance for both clusters along time can be seen in Figure 5-11.

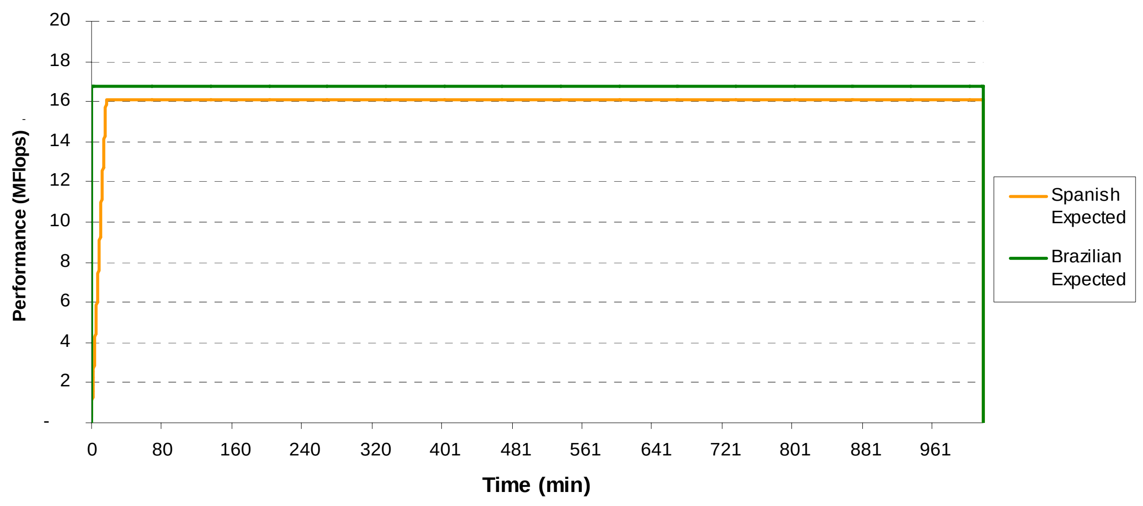


Figure 5-11: Performance behavior along time estimation for the Brazilian and Spanish clusters.

It is important to remark that, once both clusters execution is communication bounded, at Figure 5-11 the Brazilian cluster predicted steady performance is 62% of its capacity while the Spanish steady performance is 28%.

Table 5-3 shows the comparison between the estimated and the experimentally obtained startup row/column, startup time, execution time and efficiency. Figure 5-12 shows the comparison between expected and obtained performance behavior along time.

Table 5-3: Comparison between estimated and experimentally obtained values for the multi-cluster execution of a 10,000 x 10,000 matrix in 100 x 100 blocks.

	Estimated	Experimental	Error
Stabilize Row Column	7	7	0%
Spanish Startup Time (min)	16.89	16.3	4%
Execution Time	16h 41'24sec	17h 18'00sec	4%
Efficiency	39.13%	38.02%	3%

Analyzing the performance graphic it can be concluded that for both clusters the obtained performance along time was slightly inferior of the predicted value. It can be inferred that the obtained local area network throughput was lower than the measured value.

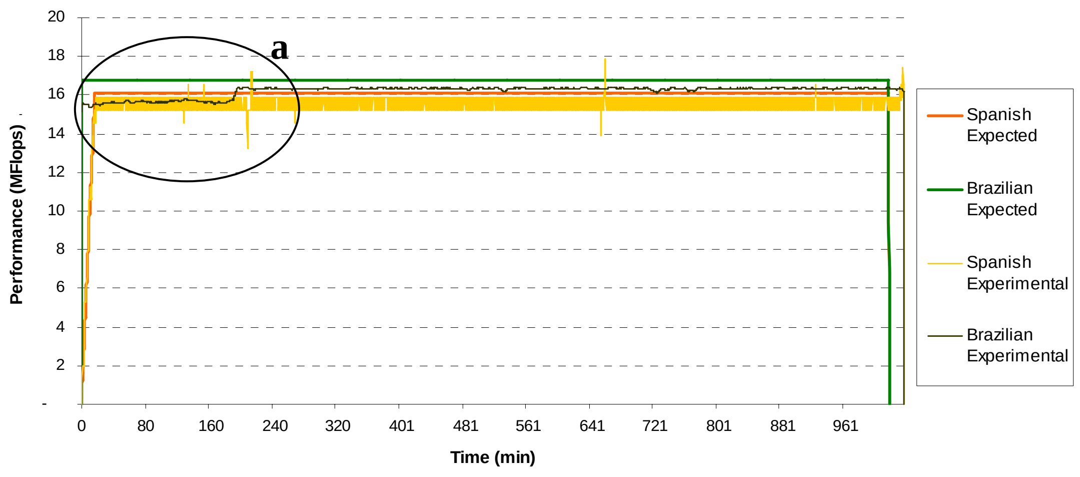


Figure 5-12: Expected and obtained performance behavior along time for the Brazilian and Spanish clusters.

It can be also seen in the graphic a curious behavior in the Brazilian cluster experimental performance for the first 200 minutes of the execution (Figure 5-12 a). During this period the Brazilian cluster performance is stabilized in a lower level of performance then the level for the rest of the execution.

This behavior can be explained through the trace of the inter-cluster communication shown at Figure 5-13. Brazilian lower performance stage coincides with the higher throughput outgoing communication period between Brazil and Spain. The reason for this behavior is that because of the local area network saturation the communication between master and communication manager causes impact on the attainable performance.

For the second experiment the 10,000 x 10,000 multiplication is done in 400 x 400 blocks and it is expected an efficient execution with all the resources. The prediction information for this execution can be seen at Table 5-4.

Table 5-4: Prediction information for a multi-cluster execution of a 10,000 x 10,000 matrix in 400 x 400 blocks.

Cluster	Startup		Estimated Execution Time (sec)		Estimated Efficiency	
	Row / Columns	Time (min)	Best	Worst	Best	Worst
Brazil		0.063	7h 07'26sec	7h 07'49sec	94.64%	94.50%
Spain	6	57.92	7h 07'25sec	7h 07'49sec	84.45%	84.39%
Multi-Cluster			**7h 07'26sec**	**7h 07'49sec**	**87.73%**	**87.65%**

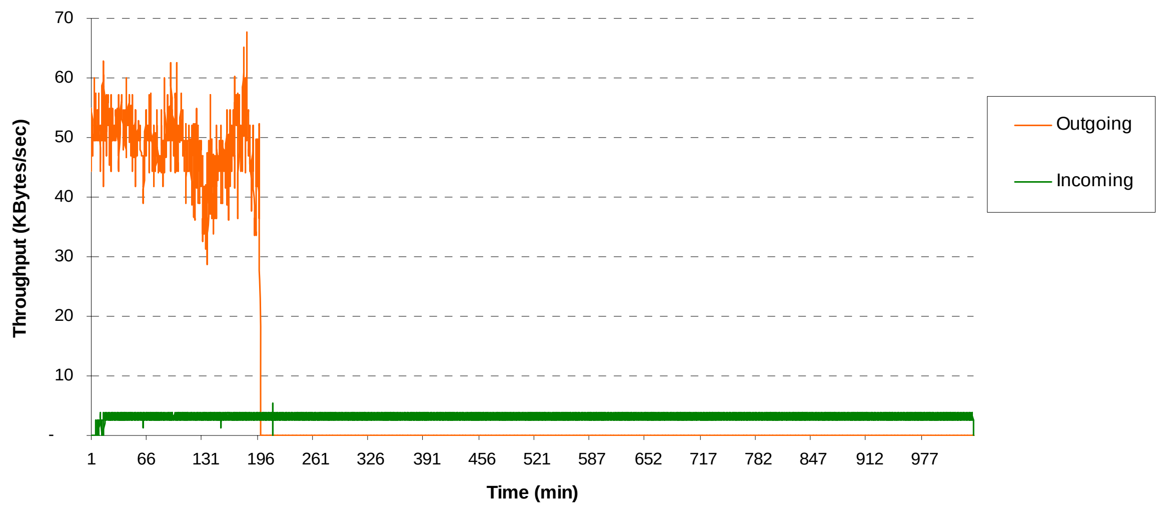


Figure 5-13: Brazilian cluster outgoing and incoming communication with the Spanish cluster during the experiment.

The expected execution time was reduced from 16 to 7 hours and it is expected more than 87% efficiency. The Spanish average performance along time grows linearly through the startup phase until the 6th row/column arrives after 57.92 min. The startup moment was delayed for this execution because of the grown in the Spanish cluster steady performance. The expected execution performance for both clusters along time for this experiment can be seen in Figure 5-14.

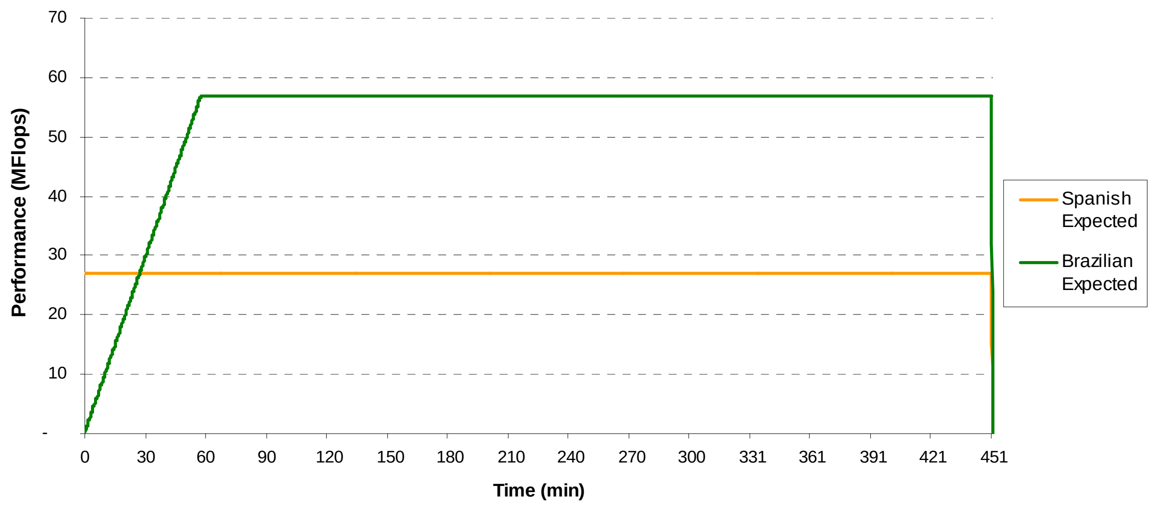


Figure 5-14: Performance behavior along time estimation for the Brazilian and Spanish clusters.

Table 5-5 shows the comparison between the estimated and the experimentally obtained startup row/column, startup time, execution time and efficiency.

Table 5-5: Comparison between estimated and experimentally obtained values for the multi-cluster execution of a 10,000 x 10,000 matrix in 400 x 400 blocks.

	Estimated	Experimental	Error
Stabilize Row Column	6	6	0%
Spanish Startup Time	57.92	53.24	8%
Execution Time	7h 07'26sec	7h 28'36sec	5%
Efficiency	87.73%	88.31%	1%

Figure 5-15 shows the comparison between expected and obtained performance behavior along time. At the performance graphic it can be seen a moment with a sudden decrease of performance followed by a great performance peak around the minute 130 (Figure 5-15 a). Analyzing the inter-cluster communication graphic at Figure 5-16 it can be seen that there was an interruption of the internet communication between the clusters at this moment (Figure 5-16 a).

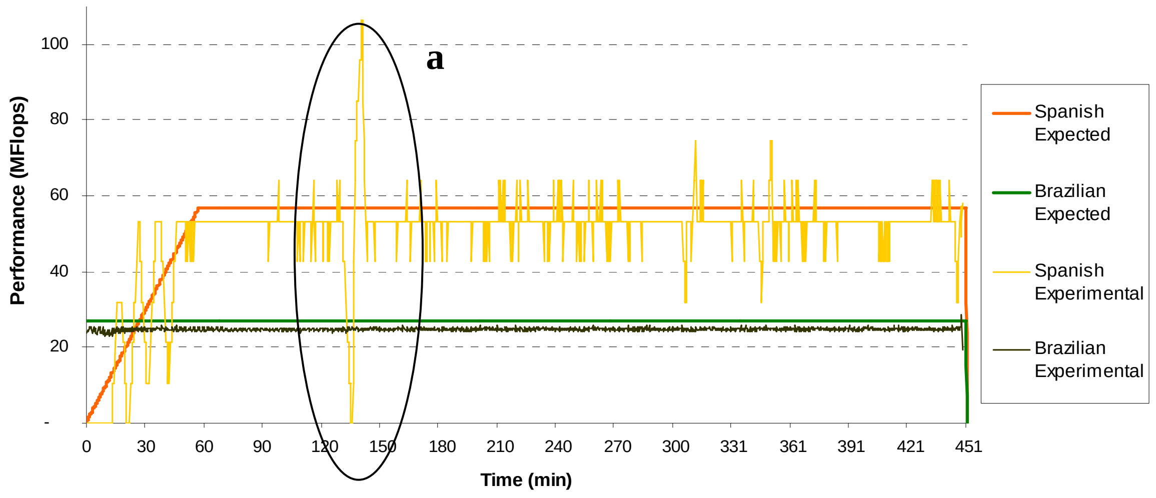


Figure 5-15: Expected and obtained performance behavior along time for the Brazilian and Spanish clusters.

The following increase in the Spanish performance is a result of the communication manager buffering strategy. During the Internet disconnection the Spanish cluster did not stop its execution and the results were kept at the communication manager. When the connection was reestablished, these results were sent to the Brazilian cluster resulting in a sudden increase of the attained performance (Figure 5-15 a).

It is also possible to conclude from the inter-cluster communication graphic that the startup time for the Spanish cluster was reduced because the effective attained Internet performance was greater than the 53.95 Kbytes/sec used for the estimation.

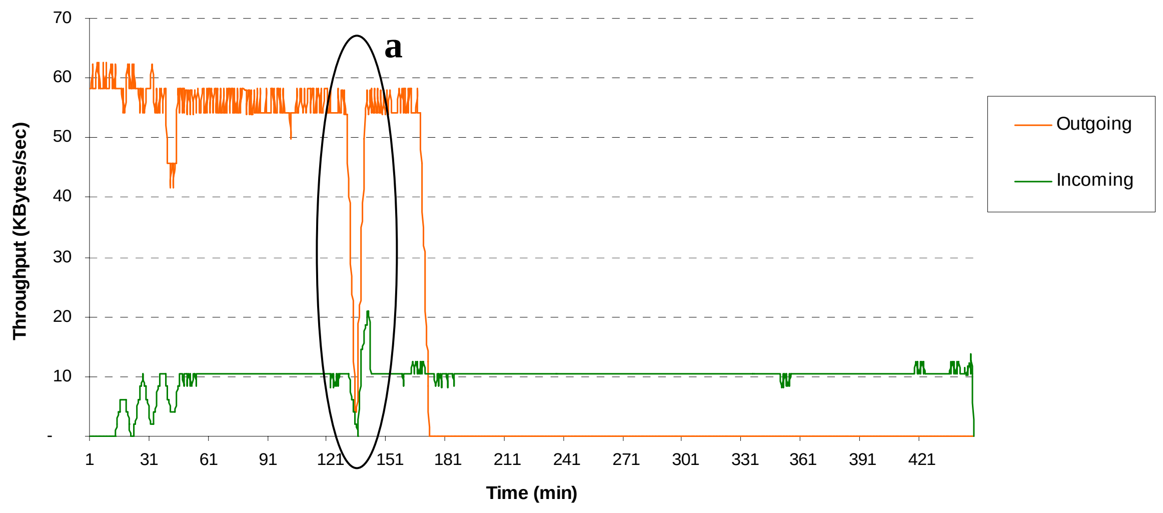


Figure 5-16: Brazilian cluster outgoing and incoming communication with the Spanish cluster during the experiment.

From these executions it is possible to conclude that the model is also accurate (with over 95% of precision) for finer-grained applications and that the estimation is influenced by the dynamic variation of the model parameters like the LAN or Internet throughputs. The accuracy of the estimation could be increased dynamically using the methodology.

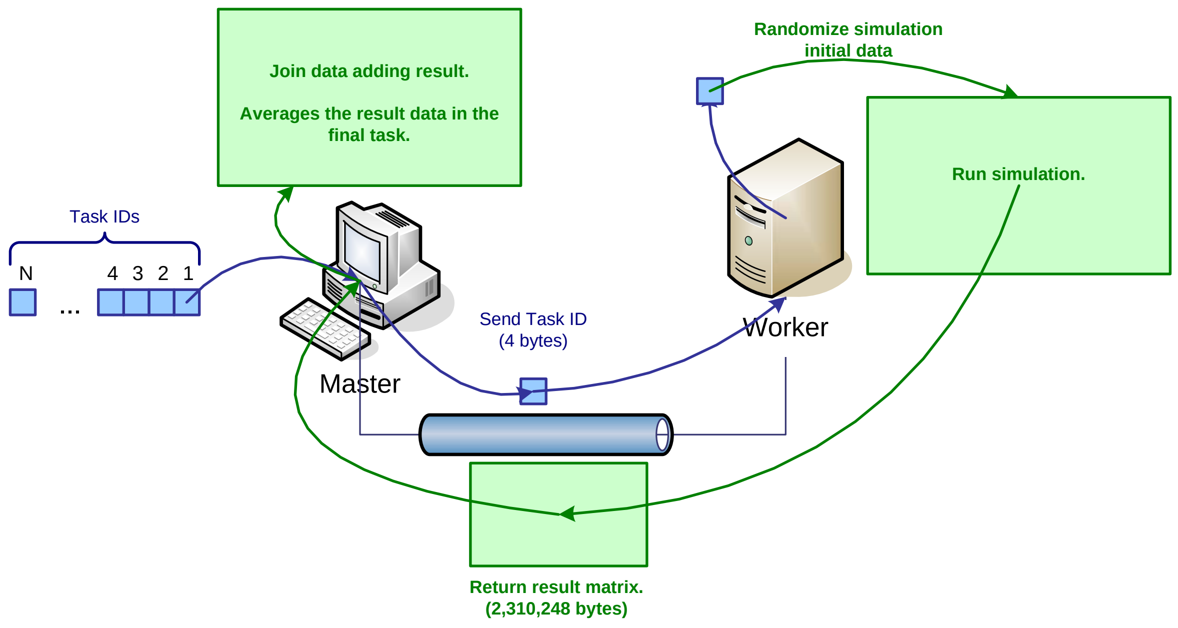
5.3. Stochastic Resonant Memory Storage Device

The Stochastic Resonant Memory Storage Device SRMSD is our example application used to follow the methodological steps. The SRMSD application [28] represents a numerical simulation to study the response of a system of bistable oscillators coupled unidirectional driven by a source of external noise and a periodic and temporal stimulus.

The physical phenomena underlying such short-time storage is that of stochastic resonant (SR) [60], that is, the presence of external noise is essential in sustaining the stored information for an appreciable time once the external stimulus has disappeared. Under this condition the set up acts as a SRMSD.

The application performs several numerical simulations involving rings with different numbers of links and delay times in the coupling. The result is the evaluation of the power-spectral density of the first oscillator during the travelling signal loops, aver-aging it over a suitable ensemble of N initial conditions in order to hinder fluctuations. The windowed Fourier transform is used to provide a picture of the decaying process.

The SRMSD model and its simulation in a single-cluster environment were developed using the master-worker paradigm. The master distributes to all workers the command to execute one simulation. The worker then generates a random set of initial conditions and simulates the traveling signal, sending back to the master a tri-dimensional matrix representing the simulation result. When the master receives one result back, it sends to this worker another simulation command (Figure 5-17).



Join data adding result.
Averages the result data in the final task.
Randomize simulation initial data
Run simulation.
Task IDs
N
...
4 3 2 1
Master
Send Task ID (4 bytes)
Worker
Return result matrix. (2,310,248 bytes)

Figure 5-17: Schematic diagram of a master sending a task simulation to a worker and receiving from this worker the simulation result.

When all the N simulations matrices results are received and added by the master, the master calculates the average value of these tri-dimensional matrices, writes it in a file and terminates the program execution.

The application is originally executed in a heterogeneous cluster located in the Universidad Nacional de General Sarmiento, Argentina. Two external clusters are available to collaborate in the execution of the SRMSD application. The first is a heterogeneous cluster located in the University Catholic of Salvador in Brazil and the other is a heterogeneous cluster located in the University Autonoma of Barcelona, Spain. The clusters are interconnected through Internet.

Our objective with this application is to demonstrate the usage of the methodology for a scientific application in more details. This application is also used to validate the analytical model and methodology, showing the efficiency increase through the resources selection and the precision of the model estimations.

The resources are selected to use the three clusters multi-cluster, keeping the efficiency over 80%. The application is tuned to reduce even more the execution time using all available resources.

On the following sections we go along the methodological steps for a simulation composed of 500 tasks, targeting a minimum of 80% efficiency. The application execution time in the cluster Argentina is of approximately 66 hours.

5.3.1. Local Cluster Analysis

The Local Cluster Analysis is focused on evaluating if it is possible for this application in the local cluster to attain external clusters performance. In order to do this analysis the application parameters and the local cluster dynamic parameters are set.

The LCA steps are *application parameters analysis*, *system parameters obtainment*, *architecture roles mapping*, *steady state analysis* and *minimum workload estimation*.

Application Parameters Analysis

The first step for doing the Local Cluster Analysis is to analyze the application and determine the basic operation and the application parameters.

It is not a simple matter to determine a basic operation for the SRMSD application since several different numerical operations are done for each simulation process. Because of this and because of the fact that the application was built in a way that one simulation is not dividable, the basic operation is set to be **one simulation task** itself and the performance metric will then be the amount of **tasks per second**.

The computational application parameters are the *amount of tasks of a certain workload* or problem, the *amount of basic operations of a certain task* and the *amount of basic operations of a certain workload*.

Since the task is one simulation itself, the *amount of tasks of a workload* of N simulations is N (81), the *amount of basic operations of a task* is equal to 1 (82) and the *amount of basic operations of a workload* is also the N tasks this workload has (83).

$$Tasks(N) = N \tag{81}$$

$$Oper_{Task} = 1 \tag{82}$$

$$Workload(N) = N \tag{83}$$

The communication application parameters are the amount of bytes necessary to be communicated for sending a task to a worker (*size of a task*), for receiving tasks result from the worker (*size of a result*) and the *total communication of a task.*

A task is simply one integer value representing a task ID. The amount of bytes necessary to be sent for a task, the *size of a task*, is then the β bytes of an integer (84). The simulation result is a matrix of 601 x 31 x 31 elements. Considering that a float point has α bytes, the *size of a result* is (85). The *total communication of a task* is the sum of these values (86). In the execution platform used both α and β have 4 bytes.

It is remarkable that for the SMRSD application the communication volume for sending a task is quite small when compared to the one of the result. This fact will have an influence on reducing the startup time and rising significantly the end time. Differently from the matrix multiplication application, the end time will them have a great importance on the system efficiency.

$$S_{Task} = \beta = 4 \tag{84}$$

$$S_{Result} = \alpha * 601 * 31 * 31 = 577{,}561 * \alpha = 2{,}310{,}244 \tag{85}$$

$$S_{Comm} = \beta + 577{,}561 * \alpha = 2{,}310{,}248 \tag{86}$$

System Parameters Obtainment

Once determined the application parameters, it is necessary to determine the system parameters for the local cluster. For the computation, the system parameter is each *computer available performance* and for the communication it is the *local area network throughput*.

Each computer available performance can be determined by executing several tasks in each worker stand alone. Table 5-6 shows for each computer in the Argentinean cluster (local cluster) the processor type, main memory size and the available performance executing the SRMSD application in 10^{-3} tasks per second.

Table 5-6: Argentinean cluster computers' processor type, memory and available performance for the SRMSD application.

Computer	Processor		Memory (MBytes)	Available Performance (10^{-3} tasks/sec)
pgs-1	Pentium III	500	512	0.7909
pgs-3	AMD	500	32	0.7951
pgs-4	AMD	300	128	0.5417
pegasus	Pentium III	800	512	2.0969
Total general				4.2246

The cluster in Argentina has its workstations connected by a 10Mbits hub. For precision reasons, communication experiments were done to determine an experimental throughput for this network. These experiments were done setting the worst processing computer (pgs-4) to act as a master and the other computers to act as workers. The workers continuously send packets of 2,310,244 bytes (the size of task result) to the master that meters the average throughput in bytes per second.

Table 5-7 shows the average throughput for the Brazilian LAN with different amount of workers and the variance of this value when compared to the global average. The variance of the average throughput when compared to the global average throughput is not considerable (it does not exceed 3%). The global average throughput of 1,068,674 is used for all the estimations.

Table 5-7: Average throughput and variance compared to the global average for continuous communication of 80,000 bytes packets with different number of workers of the Brazilian cluster.

Workers	Average Throughput (bytes/sec)	Variance to Global Average
1	1,105,107	3%
2	1,047,449	-2%
3	1,053,465	-1%
Global Average	1,068,674	

Architecture Role Mapping

All the workstations of the Argentinean cluster are interconnected through a hub while the computer **pegasus** has two network cards and is the gateway for the cluster to the Internet. Originally the computer **pegasus** has the master role because it represents the network file system server.

According with the system architecture the computer **pegasus** will have the communication manager role because it is the Internet gateway. The problem for this cluster is that **pegasus** is the best computer and would represent the best worker. In this situation it would be recommendable if possible to switch the Internet gateway to another computer.

The best candidate for having the master role is the computer **pgs-4** that has the worst processing power. In order to verify if this computer would not represent a bottleneck as a master a similar throughput experiment, with the difference that this time the computer **pgs-4** also executes the join function, was done. The result of this experiment shows no difference in the throughputs when compared with the version that does not compute task joins. The computer **pgs-4** was then selected as the master.

It is now possible to find out the value for the *cluster available performance* that represents the sum of the workers performance. The *cluster available performance* for the cluster in Argentina is the sum of **pgs-1** and **pgs-3** performances: 1.5861×10^{-3} tasks/sec.

Steady State Analysis

Once determined all the parameters it is possible to proceed with the *steady stage analysis* to find out if the steady execution is computation or communication bounded. This is done by evaluating if the *steady performance* (26) is limited by the computation or by the communication.

It can be demonstrated in (87) that the execution is computation bounded because its steady performance is limited by the computation performance. Consequently, the next step in the LCA is the *minimum workload estimation*.

(87)

$$Perf_{Ste} = \min\left(CPerf_{Avail}, Oper * \frac{TPut_{LC}}{S_{Comm}} \right)$$

$$\therefore \left\langle \begin{array}{l} CPerf_{Avail} = 1.5861\times10^{-3} \text{ tasks/sec} \\ Oper = 1 \text{task} \\ TPut_{LC} = 1{,}068{,}674 \text{bytes/sec} \\ S_{Comm} = 2{,}310{,}248 \text{bytes} \end{array} \right.$$

$$Perf_{Ste} = \min\left(1.5861\times10^{-3}, 462.5796\times10^{-3}\right)$$

Minimum Workload Estimation

At this step we will determine the minimum workload for reaching in the local cluster the desired 80% threshold efficiency. The minimum workload for a computation bounded application is defined by (88) and depends on the startup and worst end time.

(88)

$$Workload \geq CPerf_{Avail} * \left(T_{Up} + T_{WorstEnd}\right) * \frac{Eff_{Threshold}}{\left(1 - Eff_{Threshold}\right)}$$

The startup time value for the local cluster is of 5.61x10^{-6} sec (89) while the worst end time is of 634.38 sec (90), considering reassignment of the last task.

(89)

$$T_{Str} = \frac{S_{Task}}{TPut_{LC}} * \frac{(W+1)}{2} \therefore \left\langle \begin{array}{l} S_{Task} = 4 \text{bytes;} \\ \text{W} = 2; \end{array} \right.$$

$$T_{Str} = 5.61\times10^{-6} \text{ sec}$$

(90)

$$T_{WorstEnd} = \frac{S_{Result}}{TPut_{LC}} + \frac{1}{W} * \sum_{Workers-1} \frac{Oper}{Perf_{Avail}}$$

$$\therefore \left\langle \begin{array}{l} S_{Result} = 2{,}310{,}244 \text{bytes;} \\ \text{Workers - 1} \Rightarrow \text{pgs - 3} \end{array} \right.$$

$$T_{WorstEnd} = 634.38 \text{seg}$$

Based on these values, the minimum workload is then of 4.02 tasks and, since one task is not dividable, the minimum workload in the local cluster is of 5 tasks.

Once the minimum workload is not greater than the problem, the Local Cluster Analysis leads to the Multi-Cluster Analysis.

5.3.2. Multi Cluster Analysis

Inter-cluster Application Parameters Analysis

The first step of the MCA is to identify the application parameters for the communication between clusters. At this time we are analyzing the possibility to pass the SRMSD unchanged to a multi-cluster environment. In this case the application parameters between clusters are the same as inside a cluster.

System Parameters Obtainment

In this step the system parameters for the external clusters in Brazil and Spain are obtained. The system parameters are the *available performance* for each computer, each cluster *local area network throughput* and the *incoming and outgoing Internet throughput* between the main cluster and each external cluster.

At Table 5-8 it can be seen each computer processor, memory and *available performance* executing SRMSD application's tasks for the Brazilian cluster.

Table 5-8: Each computer performance for the Brazilian cluster.

Computer	Processor		Memory (MBytes)	Available Performance (10^{-3} tasks/sec)
infoquir1	Pentium	166	32	0.4683
infoquir2	Pentium	166	32	0.4590
infoquir3	Pentium III	500	64	1.4845
infoquir5	Pentium	166	32	0.3642
infoquir6	Pentium	133	32	0.3750
infoquir7	Pentium	133	32	0.3743
infoquir8	Pentium	133	32	0.2530
Total general				3.7784

Similarly to the Argentinean cluster, the cluster located in Brazil is connected with a 10 Mbits hub with the difference that this cluster has more computers. Table 5-9 shows the LAN throughput experiments for the Brazilian cluster from 1 up to 6 workers. The *average LAN throughput* for this cluster is of 987,614 bytes per second.

The Spanish cluster is the biggest and has the best performance of the three target clusters. Each computer processor, memory and *available performance* of the Spanish cluster can be seen at Table 5-10.

Table 5-9: Brazilian cluster local area network throughput with different amount of workers.

Workers	Average Throughput (bytes/sec)	Variance to Global Average
1	1,045,279	6%
2	972,011	-2%
3	971,283	-2%
4	974,110	-1%
5	979,988	-1%
6	983,014	0%
Global Average	987,614	

The Spanish cluster is interconnected through a 100 Mbits hub. Table 5-11 shows the results for the throughput experiments with different amount of workers in this cluster. The global average value is used as input for the estimation.

Table 5-10: Each computer performance for the Spanish cluster.

Computer	Processor		Memory (MBytes)	Available Performance (10^{-3} tasks/sec)
aoquir1	Pentium III	500	128	1.1312
aoquir2	Pentium III	500	128	2.0018
aoquir3	AMD Athlon	2600	256	4.6354
aoquir4	Pentium III	500	128	1.8681
aoquir5	Pentium III	800	128	1.4706
aoquir6	Pentium III	800	128	1.5286
aoquir7	Pentium III	450	128	2.0293
aoquir8	Intel Pentium 4	2600	256	6.1301
aoquir10	Pentium III	450	128	2.0507
aoquir11	Pentium III	500	128	1.9997
Total general				24.8454

For an external cluster it is also a parameter the incoming and outgoing *Internet average throughput* between the clusters. Several long duration experiments were made and the incoming and outgoing average throughput value between the external clusters and the Argentinean cluster is shown in Table 5-12.

Architecture Roles Mapping

The Brazilian cluster is interconnected to the Internet through the computer **infoquir2** that is mapped with communication manager role. The computer with the lowest performance is **infoquir8** and it has the master role. The Brazilian *cluster available performance* is then of 3.066x10^{-3} tasks/sec.

Table 5-11: Spanish cluster local area network throughput with different amount of workers.

Workers	Average Throughput (bytes/sec)	Variance to Global Average
1	10,281,379	7%
2	10,023,744	4%
3	9,598,730	0%
4	9,341,646	-3%
5	9,461,904	-1%
6	9,410,681	-2%
7	9,245,924	-4%
8	9,424,798	-2%
9	9,603,672	0%
Global Average	9,599,164	

The Internet gateway for the Spanish cluster is the computer **aoquir2** while the computer with the lowest performance for this application is **aoquir1**. These computers are mapped with communication manager and master roles respectively and the Spanish *cluster available performance* is then of 21.712x10^{-3} tasks/sec.

Table 5-12: Average Internet throughput between the external clusters (Brazilian and Spanish) and the local cluster (Argentinean).

External Cluster	Internet Throughput (bytes/sec)	
	Incoming	Outgoing
Brazil	26,384	25,430
Spain	21,802	21,206

The *multi-cluster available performance* is the sum of the *cluster available performance* and for this execution the *multi-cluster available performance* value is 26.365x10^{-3} tasks/sec. This value represents a maximum speedup of 16.623 for the multi-cluster relative to the local cluster performance.

Steady State Analysis

All the parameters are set for the clusters and the *steady state analysis* evaluates if the external clusters are computation bounded or communication bounded. It is possible for the external clusters to be communication bounded by its local area networks, by the incoming or outgoing Internet communication.

The evaluation of this step is done by analyzing the steady performance and which of the equation elements has the minimum value, representing the bottleneck. The actual approach for the inter-cluster communication is that one inter-cluster task is one task inside the external cluster. This corresponds to the first data distribution evaluated scenario for external clusters studied at the Chapter 3.

In this case the steady performance equation is (32) and the results for the Brazilian cluster can be seen in (91). From these values it can be concluded that the *estimated steady performance* is the *cluster available performance* meaning the Brazilian cluster is computation bounded. It can also be concluded that more computers could be added to the Brazilian cluster until the performance reaches the lowest limitation represented by the Internet outgoing communication.

$$Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ Oper * \dfrac{TPut_{EC}}{S_{Comm}}, \\ Oper * \dfrac{TPut_{InetIN}}{S_{Task}}, \\ Oper * \dfrac{TPut_{InetOUT}}{S_{Result}} \end{pmatrix} = \min\begin{pmatrix} 3.066\times10^{-3}, \\ 427.49\times10^{-3}, \\ 6596, \\ 11.007\times10^{-3} \end{pmatrix} \text{tasks/sec} \tag{91}$$

The same analysis applied to the Spanish cluster can be seen in (92). Differently from the Brazilian cluster, the Spanish cluster is communication bounded by the Internet outgoing communication in a performance of 9.179x10^{-3} tasks/sec.

$$Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ Oper * \dfrac{TPut_{EC}}{S_{Comm}}, \\ Oper * \dfrac{TPut_{InetIN}}{S_{Task}}, \\ Oper * \dfrac{TPut_{InetOUT}}{S_{Result}} \end{pmatrix} = \min\begin{pmatrix} 21.712\times10^{-3}, \\ 4115.04\times10^{-3}, \\ 5450, \\ 9.179\times10^{-3} \end{pmatrix} \text{tasks/sec} \tag{92}$$

Resources Selection

The *resources selection* step evaluates the computers that should be used in the external clusters respecting the efficiency threshold. The system efficiency of a cluster is upper limited to efficiency in the steady time that is the ratio between the *steady performance* and the *cluster available performance* (60).

In Table 5-13 it can be seen for the external clusters the estimated *steady performance*, the *cluster available performance* and the expected steady efficiency. Since the Brazilian cluster is computation bounded, the estimated steady performance is the available performance leading to an expected efficiency of 100% during the steady time. The conclusion is that all the resources are selected for the Brazilian cluster.

Table 5-13: Estimated steady performance, cluster available performance and estimated steady efficiency for the external clusters.

Cluster	Estimated Steady Performance (10^{-3} tasks/sec)	Cluster Available Performance (10^{-3} tasks/sec)	Expected Steady Efficiency
Brazil	3.106	3.106	100%
Spain	9.179	23.570	42%

A different situation happens with the Spanish cluster because this cluster is communication bounded. Using all the resources the best reachable efficiency for the Spanish cluster would be of 42%. Since our minimum target efficiency for this execution is 80% it is necessary to release some of the Spanish computers out of the execution.

When computers are released from the execution the Spanish *cluster available performance* value reduces and the expected efficiency increases. Our target is to find out the group of computers for which the efficiency is over 80%. Several might be the computers combination; the best choice is the one with the least amount of computers meaning lower startup and end time.

The selection of the computers **aoquir3**, **aoquir7** and **aoquir10** would decrease the *cluster available performance* to $8.715\text{x}10^{-3}$ tasks/sec. The steady period is now computation bounded and the estimated steady efficiency rises up to 100%.

Minimum Workload Estimation

In order to estimate the minimum workload each external cluster needs to receive for surpassing the efficiency threshold it is necessary to calculate for these clusters the startup time and worst end time and apply these times in the equation (58).

The startup time and worst end time equations for this inter-cluster distribution are defined in the first external cluster scenario study of the Chapter 3 respectively as the equations

(18) and (50). Table 5-14 shows for each external cluster the startup time, the worst end time and the minimum workload to reach the efficiency target.

Table 5-14: External clusters' startup time, worst end time and minimum workload for reaching efficiency threshold.

External Cluster	Startup Time (sec)	Worst End Time (sec)	Minimum Workload (Tasks)
Brazil	0.000459	2137	27
Spain	0.000367	436	16

Multi-cluster Load Balancing

The objective of this step is to estimate the amount of tasks to be solved by each cluster in order to reach a perfect balance of the load at the execution. The workload estimation is done by solving the equations system in (93). This system derives to the equation system in (94) where the variables are the workloads.

$$\begin{cases} T_{Ex_{Argentina}} = T_{Ex_{Brazil}} = T_{Ex_{Spain}} \\ Workload = Workload_{Argentina} + Workload_{Brazil} + Workload_{Spain} \end{cases} \quad \textbf{(93)}$$

$$\begin{cases} T_{Up_{Argentina}} + \dfrac{Workload_{Argentina}}{Perf_{Ste_{Argentina}}} T_{End_{Argentina}} = T_{Up_{Brazil}} + \dfrac{Workload_{Brazil}}{Perf_{Ste_{Brazil}}} T_{End_{Brazil}} \\ T_{Up_{Argentina}} + \dfrac{Workload_{Argentina}}{Perf_{Ste_{Argentina}}} T_{End_{Argentina}} = T_{Up_{Spain}} + \dfrac{Workload_{Spain}}{Perf_{Ste_{Spain}}} T_{End_{Spain}} \\ T_{Up_{Brazil}} + \dfrac{Workload_{Brazil}}{Perf_{Ste_{Brazil}}} T_{End_{Brazil}} = T_{Up_{Spain}} + \dfrac{Workload_{Spain}}{Perf_{Ste_{Spain}}} T_{End_{Spain}} \\ Workload = Workload_{Argentina} + Workload_{Brazil} + Workload_{Spain} \end{cases} \quad \textbf{(94)}$$

The balance of the load is done supposing two situations: the clusters will execute in their best time or in their worst time. Supposing the clusters will have their execution with the best time, the balanced execution will provide the *best execution time* for the application. On the other hand, the balanced load supposing clusters will execute with their worst time will provide the *worst execution time*.

Table 5-15 shows for each cluster the *startup time*, the *best end time*, the *worst end time* and the balanced workload for the best and worst possibilities.

The balanced workload is not yet the estimated workload for each cluster because tasks can not be divided. It is necessary to redistribute the tasks subdivisions generating the smallest possible load imbalance.

Table 5-15: Each cluster startup time, best end time, worst end time and balanced workload for the best and worst execution time possibilities.

Cluster	Startup Time (sec)	Best End Time (sec)	Worst End Time (sec)	Balanced Workload (tasks)	
				Best	Worst
Argentina	0.0000056	3.24	634.38	59.64	59.66
Brazil	0.0004589	274.88	2136.99	114.48	110.74
Spain	0.0003674	218.13	435.99	325.88	329.59

The best solution for the possible combinations of this redistribution can be seen in Table 5-16. This table also shows the minimum workload, the best and worst estimated execution time for each cluster and the best and worst estimated execution time for the multi-cluster system. The estimated execution time for the application in the multi-cluster is the maximum from each cluster execution time.

Table 5-16: Estimated workload and execution time for each cluster and estimated execution time for the multi-cluster.

Cluster	Minimum Workload (tasks)	Estimated Workload (tasks)		Estimated Execution Time (sec)	
		Best	Worst	Best	Worst
Argentina	5	59	59	10h 20'04sec	10h 30'35sec
Brazil	27	114	111	10h 24'12sec	10h 38'56sec
Spain	16	327	330	10h 28'58sec	10h 38'20sec
Estimated multi-cluster execution time				10h 28'58sec	10h 38'56sec

Since for all the clusters the best and worst estimated workloads are greater than the minimum workload, the Multi-Cluster Analysis reaches its end.

Two are the possible outputs from the MCA. If the estimated execution time is satisfactory then the methodological process reaches its end. The results of the methodological process are the selected resources, best and worst execution time and efficiency for each cluster and the best and worst execution time and efficiency for the multi-cluster. These data can be seen in Table 5-17.

Comparing this estimation with the local cluster stand alone execution, the worst execution time would represent a speedup of 6.13, reducing the application execution time from 65h 16min to 10h 38min.

Table 5-17: Methodological process results: selected resources, estimated best and worst execution time and efficiency for each cluster and for the multi-cluster as a whole.

Cluster	Resources	Estimated Execution Time (sec)		Estimated Efficiency	
		Best	Worst	Best	Worst
Argentina	All Cluster	10h 20'04sec	10h 30'35sec	98.58%	97.04%
Brazil	All Cluster	10h 24'12sec	10h 38'56sec	98.52%	94.43%
Spain	aoquir3, 7 and 10	10h 28'58sec	10h 38'20sec	99.42%	98.77%
Multi-Cluster		**10h 28'58sec**	**10h 38'56sec**	**99.11%**	**97.57%**

Despite that reduction, most of the resources from the cluster in Spain were taken out of the execution. There is a potential speedup of 16.6 if all resources from the three clusters could be used. If several executions of the application are expected than it would be interesting to evaluate the possibility of tuning the application in order to improve the execution speedup. Targeting this goal, the methodology will proceed with the Application Tuning phase.

5.3.3. Application Tuning

The Application Tuning phase targets to increase the attainable performance analyzing the possible bottlenecks and evaluating the possible changes in the application to surpass the problem.

The first step on the AT is to verify if the application is communication bounded in any of the clusters. For our example execution the application with all the resources is communication bounded for the Spanish cluster. It is then necessary to proceed with the *computation-communication ratio improvement evaluation* step. Because no cluster has its performance limited by the workload, it is not necessary to proceed with the *startup or end time decrease evaluation* step.

Computation-Communication Ratio Improvement Evaluation

It was seen at the MCA *steady state analysis* that the SRMSD application is communication bounded on the Internet outgoing communication for the Spanish cluster. Our objective is then to try to increase the computation-communication ratio for the Spanish cluster task results.

The SRMSD application final result is the average of each simulation task result. The main master keeps a partial sum from the received results and when the last result arrives it computes the average and save the result data.

It is then possible for a sub-cluster to send back to the main cluster results with the sum of some tasks, avoiding the communication of every single task result, improving the computation-communication ratio.

In order to do this it is necessary to change the sub-master to sum a certain number R of results from the sub-cluster before sending the result sum to the main master. It is also necessary to change the main master in order to consider a result from the external cluster as N computed results.

Before performing the necessary changes the methodology proceeds with a new evaluation of the application to determine if these changes represent a gain in the execution time without reducing significantly the efficiency.

5.3.4. Local Cluster Analysis – Tuned Application

There are no changes in the local data distribution or in any of the application or system parameters. It is not necessary to re-evaluate the Local Cluster Analysis and it is possible to proceed with the Multi Cluster Analysis.

5.3.5. Multi Cluster Analysis – Tuned Application

Inter-cluster Application Parameters Analysis

Applying the tuning strategy, the only inter-cluster computation application parameter that changes is the *amount of basic operations* for an inter-cluster task. An inter-cluster task after the tuning represents the value R of results that are joined in the sub-master before communication (95).

$$Oper_{InterTask} = R \tag{95}$$

The communication volume for sending one result or R results does not change. The difference is that for R results the task result matrix has the R added values. This means there is no difference on the communication application parameters.

Inside the clusters the task distribution and execution is not changed with the tuning alternative. This means that the system parameters on performance and throughput and the architecture roles are not changed. It is not necessary to do the *system parameters obtainment* and *architecture roles mapping* steps.

Steady State Analysis

The evaluation if the external clusters are computation bounded or communication bounded has changed for the tuning alternative. With the tuning alternative the steady time follows the second distribution scenario studied for external clusters for the analytical model (Chapter 3).

In this case the steady performance equation is (36) and the results for the Brazilian cluster can be seen in (96). It can be concluded that for the Brazilian cluster it is not necessary to aggregate tasks since for R=1 the cluster is already computation bounded.

$$Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ Oper * \frac{TPut_{EC}}{S_{Comm}}, \\ Oper * \frac{R * TPut_{InetIN}}{S_{Task}}, \\ Oper * \frac{R * TPut_{InetOUT}}{S_{Result}} \end{pmatrix} = \min\begin{pmatrix} 3.066\times10^{-3}, \\ 427.49\times10^{-3}, \\ 6596 * R, \\ 11.007\times10^{-3} * R \end{pmatrix} \tag{96}$$

The same analysis applied to the Spanish cluster can be seen in (97). Differently from the Brazilian cluster, the Spanish cluster might be limited by the outgoing communication.

$$Perf_{Ste} = \min\begin{pmatrix} CPerf_{Avail}, \\ Oper * \frac{TPut_{EC}}{S_{Comm}}, \\ Oper * \frac{R * TPut_{InetIN}}{S_{Task}}, \\ Oper * \frac{R * TPut_{InetOUT}}{S_{Result}} \end{pmatrix} = \min\begin{pmatrix} 21.712\times10^{-3}, \\ 4115.04\times10^{-3}, \\ 5450 * R, \\ 9.179\times10^{-3} * R \end{pmatrix} \text{tasks/sec} \tag{97}$$

The Spanish cluster is turns to be computational bounded when the Internet outgoing value is greater than or equal to the available performance (98). The amount of joined tasks before communication for turning to computation bounded the Spanish cluster needs to be R≥2.37.

$$9.179\times10^{-3} * R \geq 21.712\times10^{-3}$$
$$\mathrm{R} \geq 2.37 \quad \textbf{(98)}$$

Since tasks are atomic we can conclude that for values or R greater than or equal to 3 the Spanish cluster will be communication bounded. The best value of R is the minimum value 3 because this value minimizes the possible unbalance of the load. This happens because the Spanish cluster will just answer results after three tasks are computed which takes a longer duration.

Resources Selection

The *resources selection* step evaluates the computers that should be used in the external clusters respecting the efficiency threshold. Once there are no communication bounded clusters, the *steady performance* is limited to the *cluster available performance* meaning maximum efficiency with all the resources. All the multi-cluster resources are then selected for the execution.

Minimum Workload Estimation

In order to estimate the minimum workload the new startup time and worst end time are calculated following the second external cluster scenario study of the Chapter 3. The equations used are respectively (21) and (55).

Table 5-18 shows for each external cluster the startup time, the worst end time and the minimum workload to reach the efficiency target with the tuned application.

Table 5-18: External clusters' startup time, worst end time and minimum workload for reaching efficiency threshold with the tuned application.

External cluster	Startup Time (sec)	Worst End Time (sec)	Minimum Workload (Tasks)
Brazil	0.000459	2137	27
Spain	0.000368	981	90

Multi-cluster Load Balancing

Following the same procedure of the *multi-cluster load balancing* step for the application without tuning, it is possible to reach the balanced workload results for the tuned application for both the best and worst execution times.

Table 5-19 shows for each cluster and the tuned application the *startup time*, the *best end time*, the *worst end time* and the balanced workload for the best and worst possibilities.

Table 5-19: Each cluster startup time, best end time, worst end time and balanced workload for the best and worst execution time possibilities.

Cluster	Startup Time (sec)	Best End Time (sec)	Worst End Time (sec)	Balanced Workload (tasks)	
				Best	Worst
Argentina	0.0000056	3.24	634.38	29.17	29.56
Brazil	0.0004589	274.88	2136.99	55.56	52.54
Spain	0.0003678	218.37	980.73	415.28	417.91

The balanced workload is not yet the estimated workload for each cluster because tasks can not be divided and also because for the Spanish cluster the amount of tasks needs to be a multiple of R=3.

Redistributing tasks, the best solution for the possible combinations can be seen in Table 5-20. This table also shows the minimum workload, the best and worst estimated execution time for each cluster and the best and worst estimated execution time for the multi-cluster system. The estimated execution time for the application in the multi-cluster is the maximum from each cluster execution time.

Table 5-20: Estimated workload and execution time for each cluster and estimated execution time for the multi-cluster with the tuned application.

Cluster	Minimum Workload (tasks)	Estimated Workload (tasks)		Estimated Execution Time (sec)	
		Best	Worst	Best	Worst
Argentina	5	29	29	5h 04'48sec	5h 15'19sec
Brazil	27	54	51	4h 58'05sec	5h 12'49sec
Spain	90	417	420	5h 07'48sec	5h 22'41sec
Estimated multi-cluster execution time				5h 07'48sec	5h 22'41sec

Since for all the clusters the best and worst estimated workloads are greater than the minimum workload, the Multi-Cluster Analysis reaches its end.

The results of the methodological process are the selected resources, best and worst execution time and efficiency for each cluster and the best and worst execution time and efficiency for the multi-cluster. These data can be seen in Table 5-21.

Comparing this estimation with the multi-cluster execution without tuning, the speedup is of almost 2, reducing the execution time from 10h 38min to 5h 22min. If the comparison is made with the original application in the local cluster, the worst execution time would represent a speedup of 12.14, reducing the application execution time from 65h 16min to 5h 22min.

Table 5-21: Methodological process results: selected resources, estimated best and worst execution time and efficiency for each cluster and for the multi-cluster with the tuned application.

Cluster	Resources	Estimated Execution Time (sec)		Estimated Efficiency	
		Best	Worst	Best	Worst
Argentina	All Cluster	5h 04'48sec	5h 15'19sec	99.01%	94.44%
Brazil	All Cluster	4h 58'05sec	5h 12'49sec	95.36%	85.90%
Spain	All Cluster	5h 07'48sec	5h 22'41sec	98.82%	94.93%
Multi-Cluster		**5h 07'48sec**	**5h 22'41sec**	**98.44%**	**93.90%**

5.3.6. Experimental Validation

The validation of the performance analysis and system tuning methodology applied to the Stochastic Resonant Memory Storage Device application is explained in two parts: first the proposed problem experiments and then several experiments varying workload and clusters configurations.

At the first part three experiments corresponding to proposed problem used to explain the methodology are described in more details. The proposed problem is the solution of 500 simulations in three geographically distributed clusters located in Argentina, Brazil and Spain and connected by Internet.

The experiments correspond to three different moments in the methodology. For all the experiments the methodology and analytical model are used to predict each cluster workload, execution time and efficiency. These predictions are done for the best and worst possible execution time. From these data it is also predicted the multi-cluster best and worst execution time and efficiency.

The first analyzed moment, called *complete resources*, occurs when the application is just ported to the multi-cluster, **not tuned** and executes using all the available computers. This execution ignores the predicted efficiency or execution time.

The evaluation for the *complete resources* execution estimates a low efficiency because the resources were not selected. The objective of this experiment is to verify that the efficiency reduction and if it happens because of the predicted causes.

The second analyzed moment is called *selected resources*. This moment takes place after the first complete evaluation of the application, when resources are selected and before the application was tuned. The predicted workload, efficiency and execution time for this experiment can be seen at Table 5-16 and Table 5-17.

This experiment targets to demonstrate that, after the methodological process and before the tuning, the minimum efficiency threshold was guaranteed.

The third experiments that finalizes the first validation part corresponds is called *tuned application*. This experiment corresponds to the final result of the methodological process where the application was tuned and all the resources are used. The prediction of this experiment can be seen at Table 5-20 and Table 5-21.

The second part of the validation shows a comparison of efficiency between prediction and execution for several experiments, with different problem sizes and clusters configurations for the three mentioned execution types. The objective is to show the prediction accuracy for the three different moments with smaller workloads, that can cause greater load imbalance, and greater workloads, for whose it is more probable greater Internet throughput variances.

Proposed Problem Experiments

Along the methodological process of the SRMSD application it was concluded that it is not possible to obtain an 80% efficient execution of the application without changes. The reason for this impossibility was the fact that the Spanish cluster was computational bounded and just could attain part of its performance.

In order to check this impossibility an experiment was designed with the unchanged application and all the computers in the three clusters multi-cluster. This experiment corresponds to what we characterized a *complete resources* experiment.

Table 5-22 shows, using the analytical model for this *complete resources* experiment, the best and worst estimated workload, execution time and efficiency for each cluster and for the multi-cluster. It can be seen in this table that the estimated efficiency for multi-cluster execution is below 50% because of the 39% expected efficiency of the Spanish cluster.

Table 5-22: Complete resources experiment estimated best and worst execution time and efficiency for each cluster and for the multi-cluster.

Cluster	Estimated Workload (tasks)		Estimated Execution Time (sec)		Estimated Efficiency	
	Best	Worst	Best	Worst	Best	Worst
Argentina	58	58	10h 09'33sec	10h 20'04sec	99.90%	97.86%
Brazil	111	108	10h 07'54sec	10h 22'38sec	98.89%	94.25%
Spain	331	334	10h 10'05sec	10h 22'48sec	39.57%	39.12%
Multi-Cluster	**500**	**500**	**10h 10'05sec**	**10h 22'48sec**	**49.67%**	**48.65%**

At Table 5-23 it can be seen for each cluster and for the multi-cluster the workload, execution time and efficiency results for the experiment. A graphical comparison of the predicted best and worst execution time with the experimentally obtained execution time can be seen in Figure 5-18.

Table 5-23: Experimentally obtained workload, execution time and efficiency for the complete resources experiment.

Cluster	Workload (tasks)	Execution Time (sec)	Efficiency
Argentina	57	10h 02'48sec	99.37%
Brazil	106	10h 04'32sec	95.30%
Spain	337	10h 08'25sec	40.40%
Multi-Cluster	**500**	**10h 08'25sec**	**49.80%**

From the comparison of the prediction with the experimentally obtained data it can be seen that the Spanish cluster execution time was 0.27% lower than the best expected value. Analyzing the experiment real Internet throughput we observed that the average outgoing throughput for the Spanish cluster was 21,558 bytes/sec. This value is 1.77% greater than the value of 21,206 bytes/sec used for estimation.

It can be concluded that the Internet performance had the influence on the increase of the Spanish cluster contribution. This increase resulted that the Spanish cluster was responsible for more tasks which lead to the execution of fewer tasks by the Argentinean and Brazilian clusters. This is the reason for fact that these clusters execution time is better than expected.

This conclusion is reinforced when analyzing the efficiency for each cluster. Just the Spanish cluster had its efficiency not between the predicted minimum and maximum values.

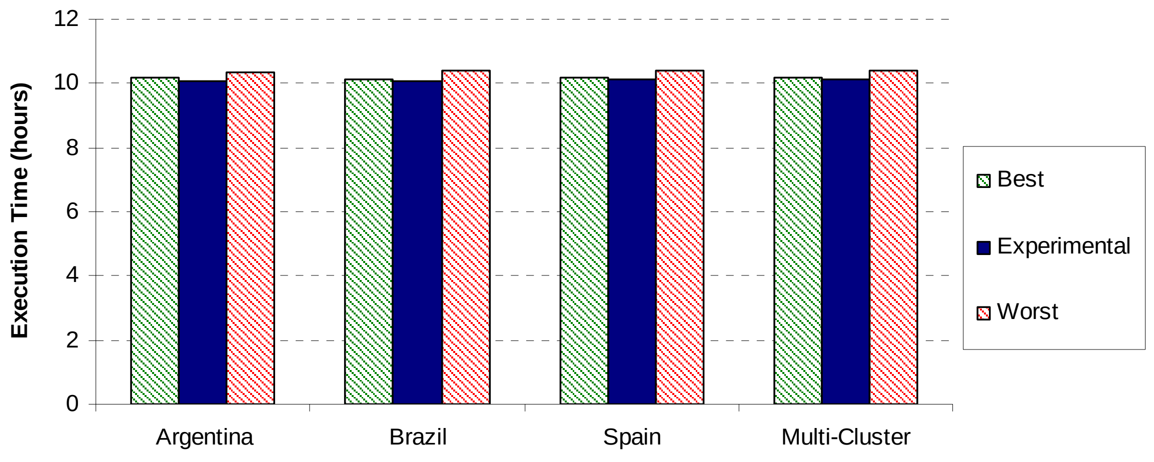


Figure 5-18: Graphical comparison, for the *complete resources* experiment, between the best and worst predicted execution time and the experimentally obtained time.

Even with the Internet variance, the prediction was 99.7% accurate when comparing the experimental execution time with the best predicted execution time. As predicted by the methodology, it was not possible to reach the totality of the selected resources, reducing the system efficiency to 49.8%.

The *selected resources* experiment is done with the application not tuned, after the multi-cluster analysis. The predicted values for this experiment are shown at Table 5-24. These values were taken from the *multi-cluster analysis* results (Table 5-16 and Table 5-17).

Table 5-24: Selected resources experiment estimated best and worst execution time and efficiency for each cluster and for the multi-cluster.

Cluster	Estimated Workload (tasks)		Estimated Execution Time (sec)		Estimated Efficiency	
	Best	Worst	Best	Worst	Best	Worst
Argentina	59	59	10h 20'04sec	10h 30'35sec	98.58%	97.04%
Brazil	114	111	10h 24'12sec	10h 38'56sec	98.52%	94.43%
Spain	327	330	10h 28'58sec	10h 38'20sec	99.42%	98.77%
Multi-Cluster	**500**	**500**	**10h 28'58sec**	**10h 38'56sec**	**99.11%**	**97.57%**

Comparing this prediction with the *complete resources* prediction it can be seen that at the *selected resources*, the estimated efficiency is greater than 97% while at the *complete resources* it lower than 50%. The expected execution time for the *selected resources* execution is just 2.6% greater than for the *complete resources* one.

At Table 5-25 it is shown for each cluster and for the multi-cluster the experimentally obtained workload, execution time and efficiency. A graphical comparison of the predicted best and worst execution time with the experimentally obtained execution time can be seen in Figure 5-19.

Table 5-25: Experimentally obtained workload, execution time and efficiency for the complete resources experiment.

Cluster	Workload (tasks)	Execution Time (sec)	Efficiency
Argentina	59	10h 29'09sec	98.55%
Brazil	111	10h 31'41sec	95.51%
Spain	330	10h 35'17sec	99.34%
Multi-Cluster	**500**	**10h 35'17sec**	**98.13%**

It can be seen that the model adjusts to the system once all the experimental execution and efficiency values are inside the predicted range.

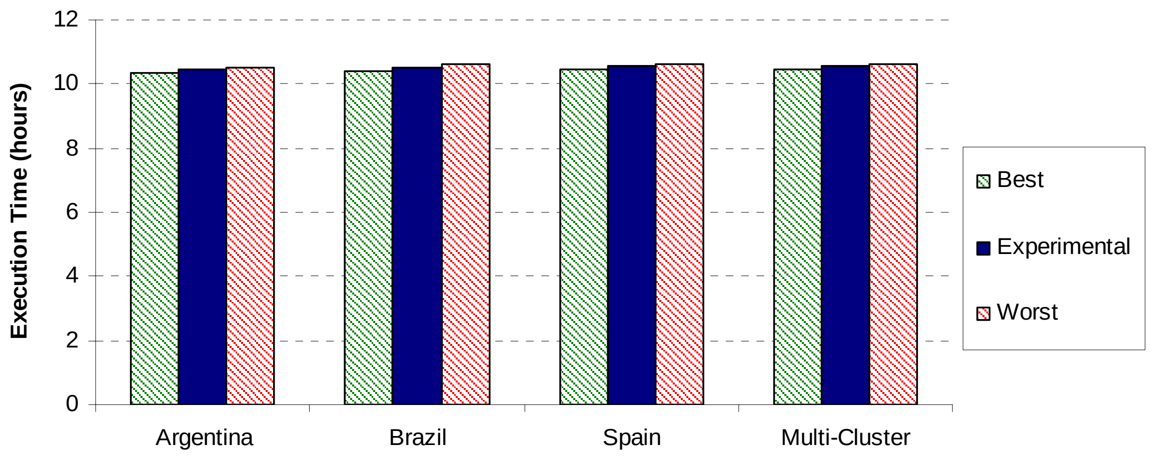


Figure 5-19: Graphical comparison, for the *selected resources* experiment, between the best and worst predicted execution time and the experimentally obtained time.

The *tuned application* experiment is done with all the resources for the application after tuning. The predicted values for this experiment are shown at Table 5-26. These values were taken from the results of the *multi-cluster analysis* for the tuned application (Table 5-20 and Table 5-21).

Table 5-26: Tuned application experiment estimated best and worst execution time and efficiency for each cluster and for the multi-cluster.

Cluster	Estimated Workload (tasks)		Estimated Execution Time (sec)		Estimated Efficiency	
	Best	Worst	Best	Worst	Best	Worst
Argentina	29	29	5h 04'48sec	5h 15'19sec	99.01%	94.44%
Brazil	54	51	4h 58'05sec	5h 12'49sec	95.36%	85.90%
Spain	417	420	5h 07'48sec	5h 22'41sec	98.82%	94.93%
Multi-Cluster	**500**	**500**	**5h 07'48sec**	**5h 22'41sec**	**98.44%**	**93.90%**

According to the prediction, for this execution the 80% efficiency threshold is guaranteed for the multi-cluster using all the resources.

At Table 5-27 it is shown for each cluster and for the multi-cluster the experimentally obtained workload, execution time and efficiency. A graphical comparison of the predicted best and worst execution time with the experimentally obtained execution time can be seen at Figure 5-20.

Table 5-27: Experimentally obtained workload, execution time and efficiency for the complete resources experiment.

Cluster	Workload (tasks)	Execution Time (sec)	Efficiency
Argentina	29	5h 09'52sec	98.35%
Brazil	52	4h 49'58sec	97.47%
Spain	419	5h 09'52sec	98.63%
Multi-Cluster	**500**	**5h 09'52sec**	**97.78%**

The model provides a valid approximation to the system since for each cluster and for the multi-cluster the experimentally obtained execution time and efficiency data are inside the predicted boundaries.

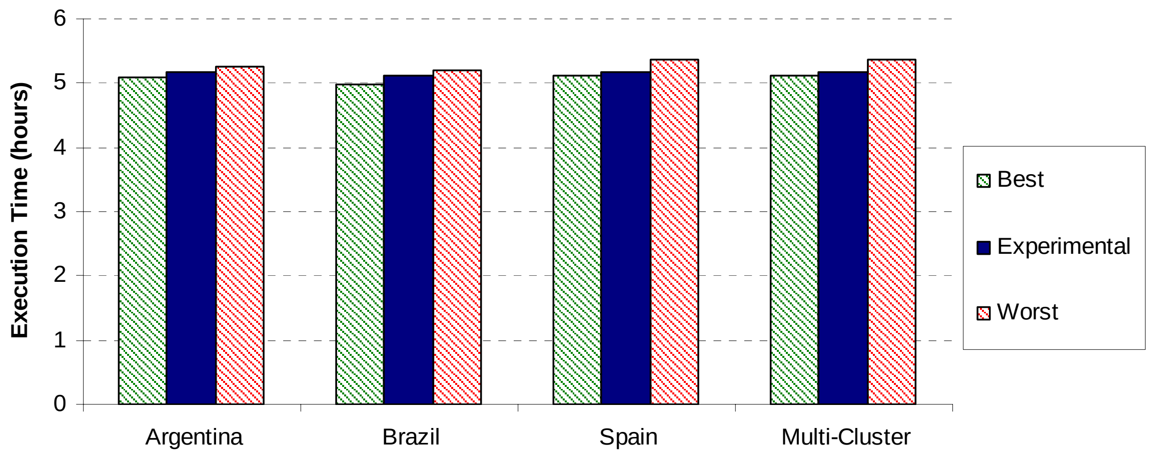


Figure 5-20: Graphical comparison, for the *tuned application* experiment, between the best and worst predicted execution time and the experimentally obtained time.

The comparison between each experiment predicted and obtained execution times in the multi-cluster can be seen in Figure 5-21. Figure 5-21 also shows the obtained multi-cluster efficiency for these experiments. It can be concluded that after the resources selection there is a tiny increase in the execution time justified by a great increment on the system efficiency. When the application is tuned the execution time is decreased in almost 50% without loss in the obtained efficiency.

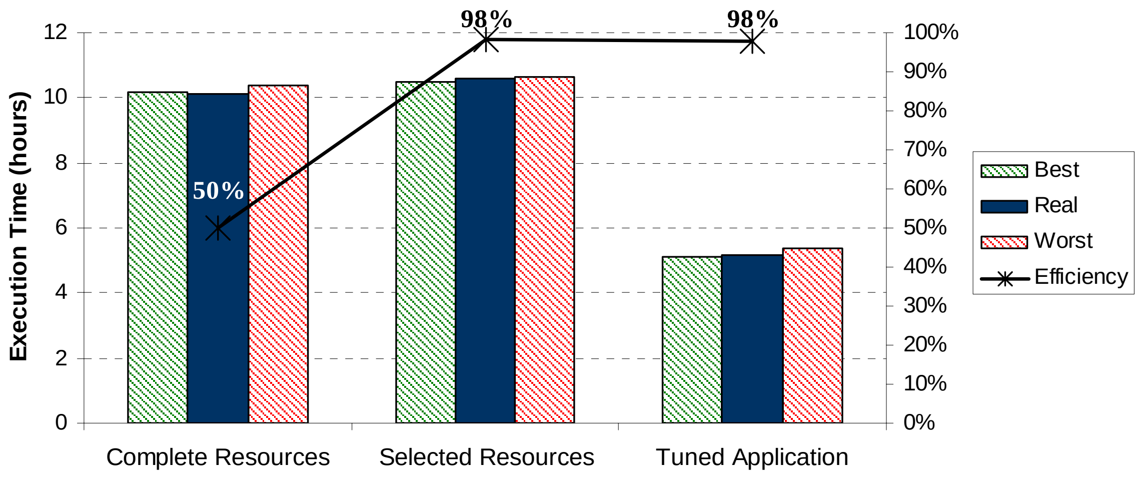


Figure 5-21: Graphical comparison, for the *selected resources* experiment, between the best and worst predicted execution time with the experimentally obtained time.

The Figure 5-22 compares the maximum speedup and the obtained speedup for each experiment related with the original execution just in the local cluster. The maximum speedup is obtained considering the selected resources *available performance*.

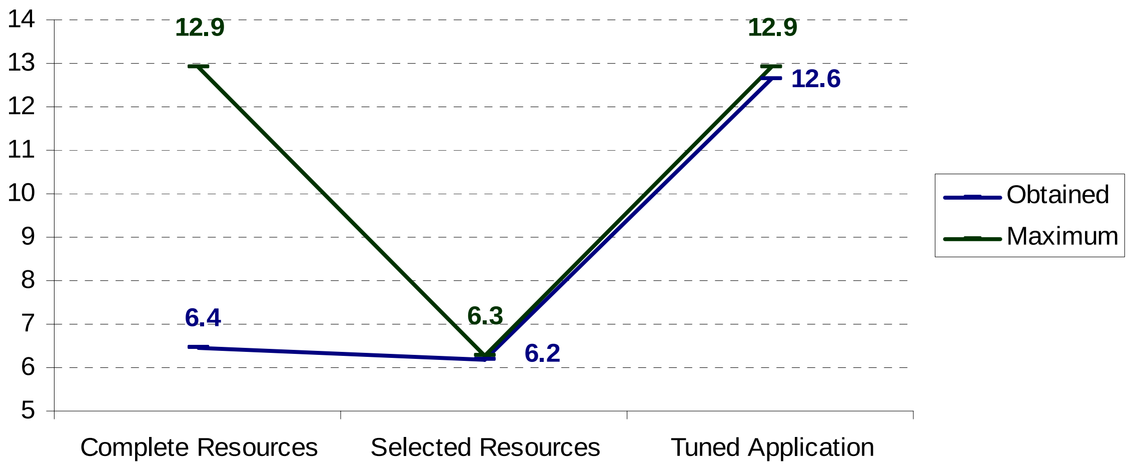


Figure 5-22: Speedup of the experiments related to the local cluster stand alone execution time.

Other Experiments

For validating the methodological approach, some experiments with different workloads and clusters configurations are presented. Two basic problem workloads were chosen: 100 and 500 tasks.

Executing in the multi-cluster 100 tasks improves the balance unload while 500 tasks executions are more likely to face Internet's throughput variations. Experiments were done for each execution type: *complete resources*, *selected resources* and *tuned application*.

The presented experiments (from A to H) categorized by experiment type and workload size can be seen in Table 5-28.

Table 5-28: Designed experiments categorized by experiment type and workload size.

	Workload Size	
Experiment Type	100 tasks	500 tasks
Complete Resources	A, B	C
Selected Resources	D, E	F
Tuned Application	G	H

For these experiments the multi-cluster was formed by the Argentinean, Brazilian and Spanish clusters. For each experiment we compare the predicted and obtained efficiency. This comparison can be seen in Figure 5-23.

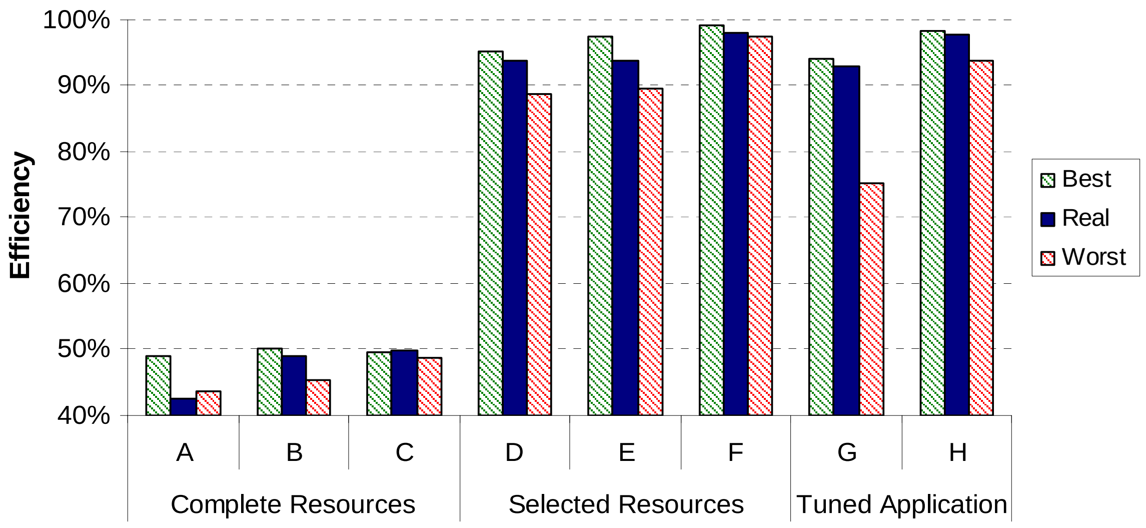


Figure 5-23: Speedup of the experiments related to the local cluster stand alone execution time.

For all the experiments except the experiment A the model is a good approximation to the experimental data. The experiment A obtained efficiency is lower than the minimum estimated value because for this experiment the obtained Internet throughput for the Spanish cluster was 16,102 bytes/sec. This value is 24% inferior to the evaluated throughput used for the estimation.

5.4. Traveling Salesman Problem

The traveling salesman problem (TSP) is a well known NP-hard combinatorial optimization problem. A salesman is required to visit once and only once each of N different cities starting from a base city, and returning to this city. The objective is to find out the path that minimizes the total distance traveled by the salesman.

It is possible to formulate as a TSP many problems in science, engineering, and bioinformatics fields, such as flexible manufacturing systems, routing problems for printed circuits manufacturing, physical mapping problems [2], genome rearrangement [69], and phylogenetic tree construction [57].

The most direct solution for the TSP problem would be to try all the permutations (ordered combinations) and see which one is cheapest. Nevertheless given that the number of permutations is the factorial of the number of cities, the execution time for this solution increases rapidly.

The algorithm factorial complexity motivated the research in two lines to attack the problem: exact algorithm or heuristics. The exact algorithms search for the optimal solution through the use of branch-and-bound technique (40 to 60 cities), linear programming (120-200) or branch-and-bound and cut based on linear programming (5,000 cities) [49].

Heuristic solutions for the TSP are approximation algorithms that in a fraction of the time of the exact algorithm reach an approximate solution, close to the optimal. Heuristic algorithms to solve the traveling salesman problem might be based on genetic and evolutionary algorithms [75][74], simulated annealing [65], Tabu search, neural networks [6], ant system, etc.

The traveling salesman problem is used in our work for analyzing different levels of predictability in the computation time of tasks. One of the proposed analytical model inputs is the performance for each computer. This performance is measured with executions of the application at the workstations.

The precision of this measurement approach is accurate for applications with a high level of determinism and where the execution time does not significantly vary for different data inputs. Both the matrix multiplication (considering no predominance of zero-valued cells) and the SRMSD applications are deterministic.

Three different versions to obtain the exact TSP solution were developed in order to analyze the influence of different levels of determinism in the robustness of the analytical model accuracy. This analysis is used to evaluate strategies to adapt the model for applications with different levels of predictability.

In this section the different TSP implementations are presented, the performance behavior for each implementation is studied and changes in the analytical model usage are proposed to guarantee the threshold efficiency. Experiments with different sets of cities are designed and the accuracy of different levels of predictability is analyzed.

5.4.1. TSP Algorithms

The three different implementations of the TSP are master-worker programs based on calculating the distance of the possible permutations. Considering C different cities, the master defines a certain level L to divide the tasks. Tasks are the possible permutations of C cities in L elements. The granularity G of a task is the number of cities that defines the task sub-tree: G = C – L.

A city is defined with two coordinates in a plane and at the execution startup the master sends the cities coordinates to every worker.

A diagram of the possible permutations for 5 cities, considering the salesman starts and ends his trip at the city 1, can be seen in Figure 5-24. The master can divide this problem into 1 task of level 0 or 4 tasks of level 1 or 12 tasks of level 2 for example. The tasks of the first level would be represented by the cities 1 and 2 for the first task, 1and 3 for the second, followed by 1and 4 and 1 and 5.

Workers are responsible for calculating the distance of the permutations left in the task and send to the master the best path and distance of these permutations.

The differences between the three analyzed algorithms are on the pruning strategy. One of the characteristics of the TSP is that once the distance for a path is superior to the already computed minimum distance it is possible to prune this path tree. The pruning process is illustrated at Figure 5-25 where each arrow has the distance between the two cities it connects.

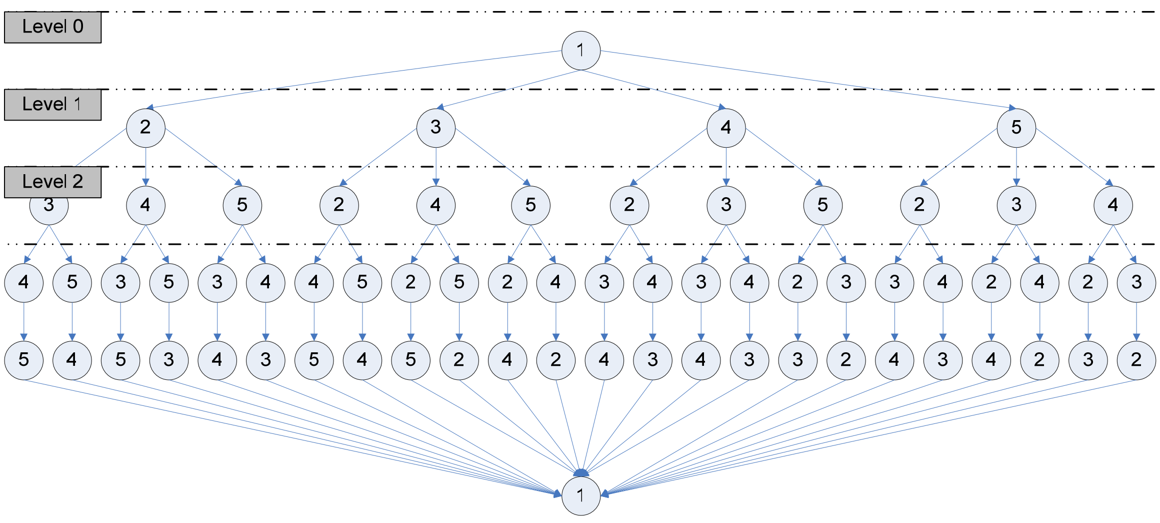


Figure 5-24: Possible paths for the salesman considering 5 cities.

The total distance for the first followed path (in the left) at Figure 5-25 is of 38 units. The distance between 1 and 2 on the second path (in the right) is already of 44 units. It is then not necessary in this path for the algorithm to keep calculating distances from the city 2 on because it is impossible to reach a better distance for this branch.

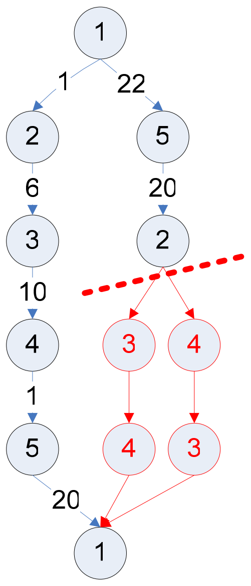


Figure 5-25: Illustration of the pruning process in the TSP.

Our three approaches for the solution of the TSP differ on the pruning strategy. At the first approach, which we call *exhaustive*, there is no pruning and the distance for the whole possible permutations is calculated. It is expected a predictable behavior for this approach because the amount of operations does not change with the input data.

For the second approach, called *task pruning*, the worker starts a task with no minimum distance and prunes with the distances found inside the task. At the third approach, *global pruning*, the master sends tasks with the current achieved minimum distance and workers prune whenever it is possible, based on the global distance value, from the beginning of the task on.

The predictability decreases from the *exhaustive* to the *global pruning* approaches as illustrated on Table 5-29.

Table 5-29: TSP algorithms used in the thesis, pruning strategy and expected predictability level.

Algorithm	Pruning Strategy	Expected Predictability
Exhaustive	No pruning. Every path is evaluated.	High
Task Pruning	Branches are pruned inside a task based on the task minimum distance.	Average
Global Pruning	Branches are pruned based on the global minimum value sent within the task.	Low

5.4.2. Algorithms Performance Behavior

Our first step is to analyze the performance behavior of the algorithm with different set of cities as data inputs. In order to perform this analysis we measured the execution time of several tasks in a single computer for different sets of cities.

Exhaustive algorithm

For the *exhaustive* algorithm we analyzed the TSP with 15 cities (C=15) and a computer executing sub-trees with 11 cities (G=11 and L=4). The graphics in Figure 5-26 and Figure 5-27 show the distribution of tasks per execution time for the *exhaustive* algorithm execution in a specific computer. Two different data inputs were used for Figure 5-26 and Figure 5-27. The red line shows the average execution time of a task.

Analyzing these graphics it can be seen that the variance of time between tasks is lower than 0.5% for the first set of data and lower than 1.5% for the second set. The average time for the first set of data was 8.533 sec while for the second set was 8.680. The difference between the averages of the two data sets is of less than 2%.

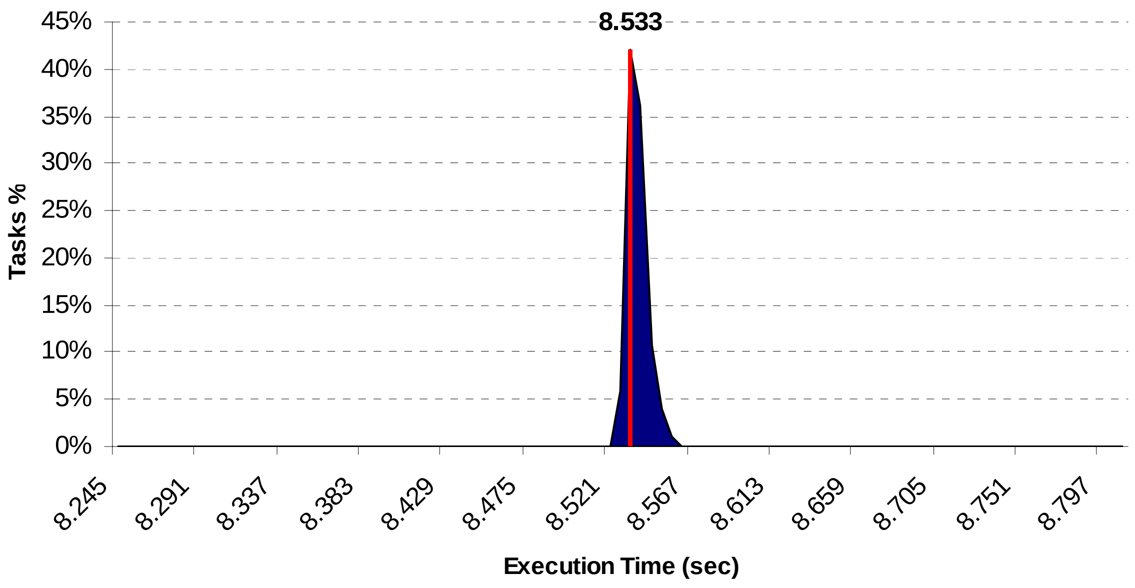


Figure 5-26 Tasks distribution for the *exhaustive* TSP algorithm execution of the first dataset with 15 cities in 11 cities branches.

It is possible to conclude that our proposed methodology would fit well in this distribution because of its level of predictability.

For proving the predictability of this algorithm, the *exhaustive* TSP program execution time was evaluated for a workload of 14 cities divided in tasks of 11 cities. Three different sets of cities were used for the cluster execution.

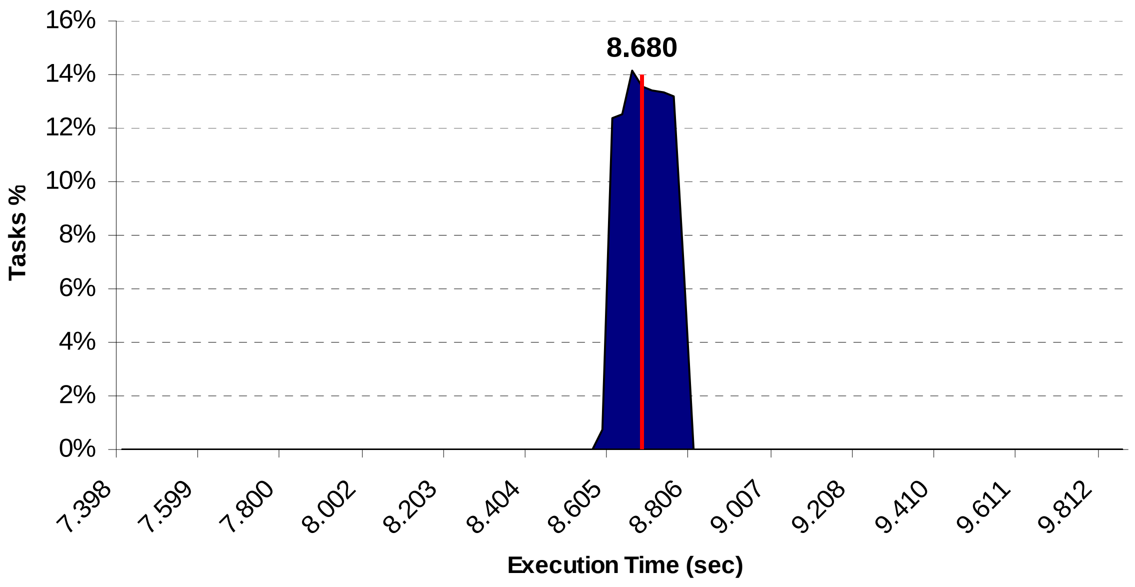


Figure 5-27 Tasks distribution for the *exhaustive* TSP algorithm execution of the second dataset with 15 cities in 11 cities branches.

Table 5-30 shows the execution time prediction and the experimental execution time for the *exhaustive* algorithm in the cluster with the different data input sets. It can be seen in this table that the prediction was accurate once the experimental execution time was inside the estimated range.

Table 5-30: Estimated and execution time for the TSP exhaustive execution with different sets of 14 cities with tasks of 11 cities.

City Set	Estimated Time (sec)		Execution Time (sec)
	Min	Max	
A	291.65	302.89	295.68
B	291.65	302.89	294.31
C	291.65	302.89	294.35

Figure 5-28 shows the execution performance in tasks/sec along time for the three different sets of cities. It can be seen in this graphic that there are no variations on the obtained performance.

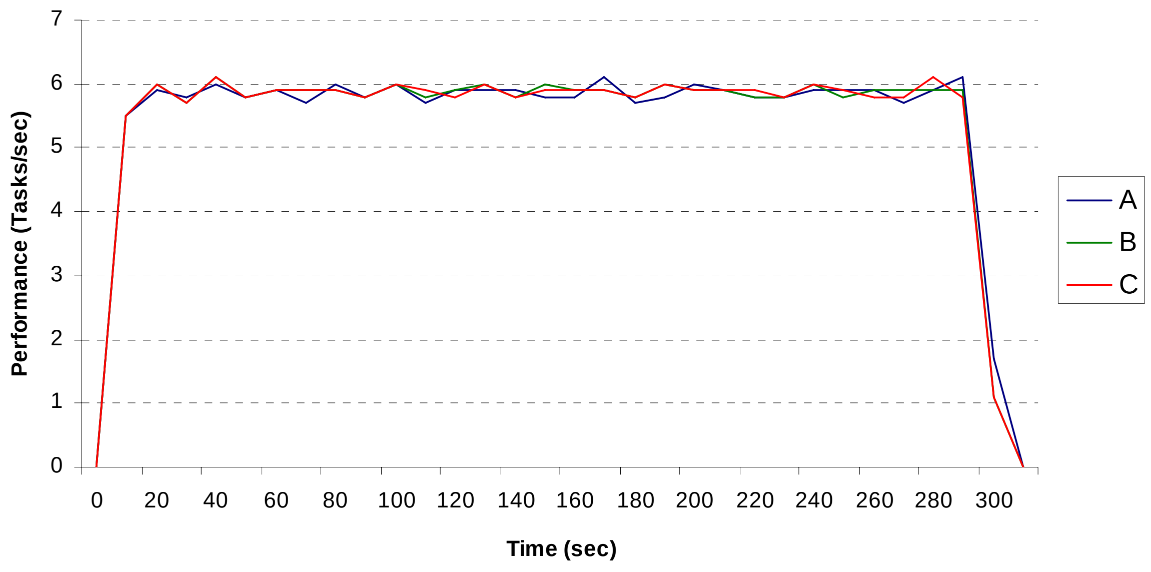


Figure 5-28 Execution performance along time for the exhaustive algorithm with different sets of cities in a cluster.

Task pruning algorithm

The same datasets were used for the *task pruning* algorithm. The task distribution for each dataset can be seen in Figure 5-29 and Figure 5-30. This distribution variance is far greater than the one in the exhaustive algorithm varying from 41% to 206% of the average value for the first dataset and from 51% to 170% for the second dataset.

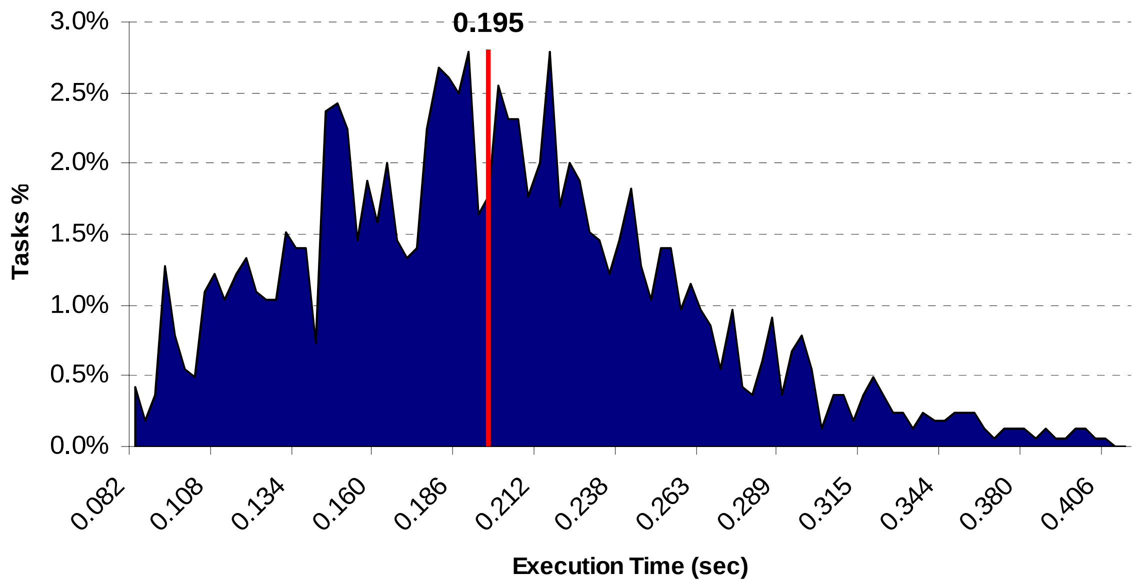


Figure 5-29 Tasks distribution for the *task pruning* TSP algorithm execution of the first dataset with 15 cities in 11 cities branches.

The average task execution time for the datasets also differs in 16%; from 0.195 to 0.230 sec meaning that different datasets might result in different average execution times. It is not possible than to use our proposed methodology based simply in static performance data obtained before the execution.

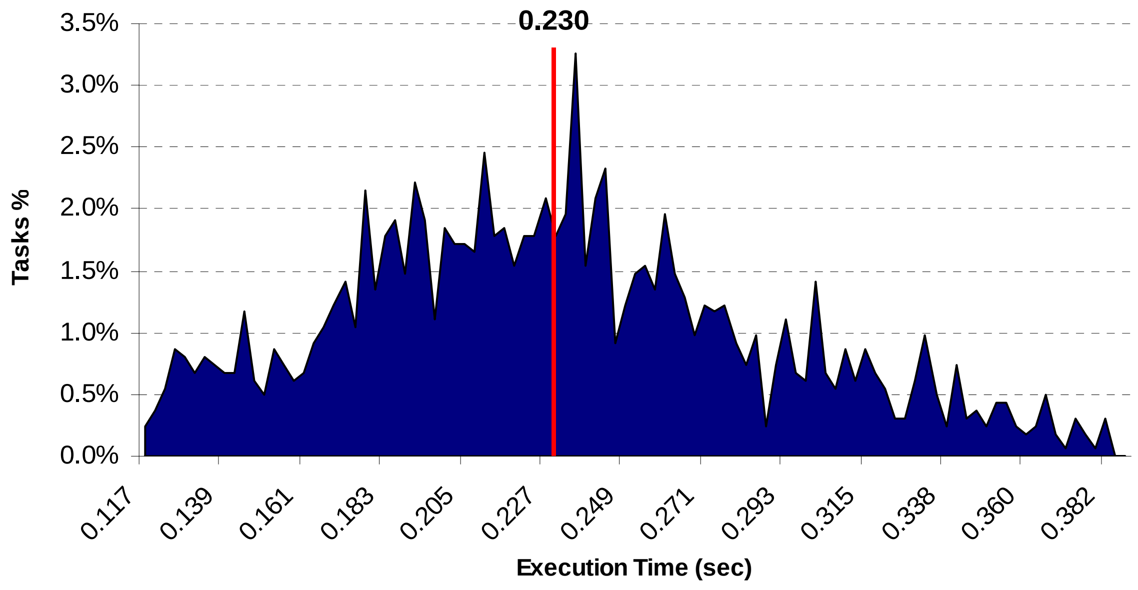


Figure 5-30 Tasks distribution for the *task pruning* TSP algorithm execution of the second dataset with 15 cities in 11 cities branches.

Although the level of predictability of this algorithm is moderate and our proposed model might be used dynamically. The best alternative would be to incorporate the model into the application or into the middleware. It would then be possible to dynamically determine the average performances and to use the analytical model to predict, along the execution, the possible execution time range and its probability. This prediction can be used for dynamic decision taking of, for example, change the distribution granularity targeting to balance computation and communication.

Figure 5-31 shows the performance execution along time for the *task pruning* algorithm with three different sets of 16 cities with tasks of 12 cities. It can be seen in this graphic that there is a certain variance (from 3 up to 7 tasks per second) in the obtained performance along time. It can also be seen that for different workloads, different execution times was obtained, varying from 550 to 800 seconds.

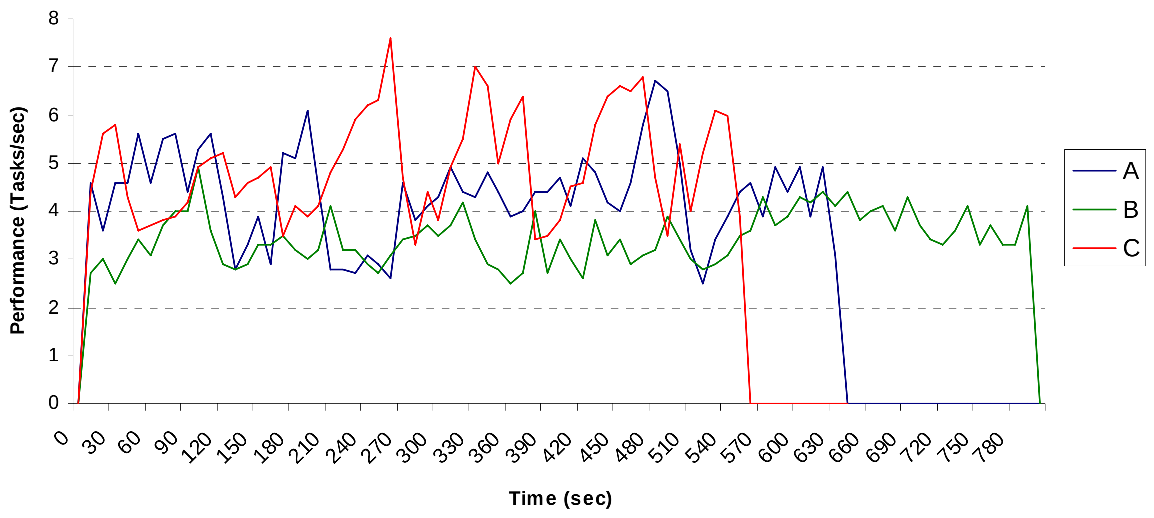


Figure 5-31 Execution performance along time for the task pruning algorithm with different sets of cities in a cluster.

In Table 5-31 it can be seen the dynamic prediction of the execution time using the performance average after 10% of the execution. This table also shows the prediction error compared to the experimental time. It can be seen that the prediction error was under 10%.

Table 5-31: Estimated, execution time and prediction error for the TSP task pruning execution with different sets of 16 cities with tasks of 12 cities.

Cities Set	Estimated Time (sec)		Experimental Time (sec)	Prediction Error
	Min	Max		
A	593.56	609.76	631.74	4%
B	859.97	876.16	798.24	7%
C	597.90	614.09	550.12	8%

Global pruning algorithm

The task distribution for the *global pruning* algorithm with different datasets can be seen in Figure 5-32 and Figure 5-33. For this algorithm the maximum and minimum task execution time can vary up to 200 or 300 times.

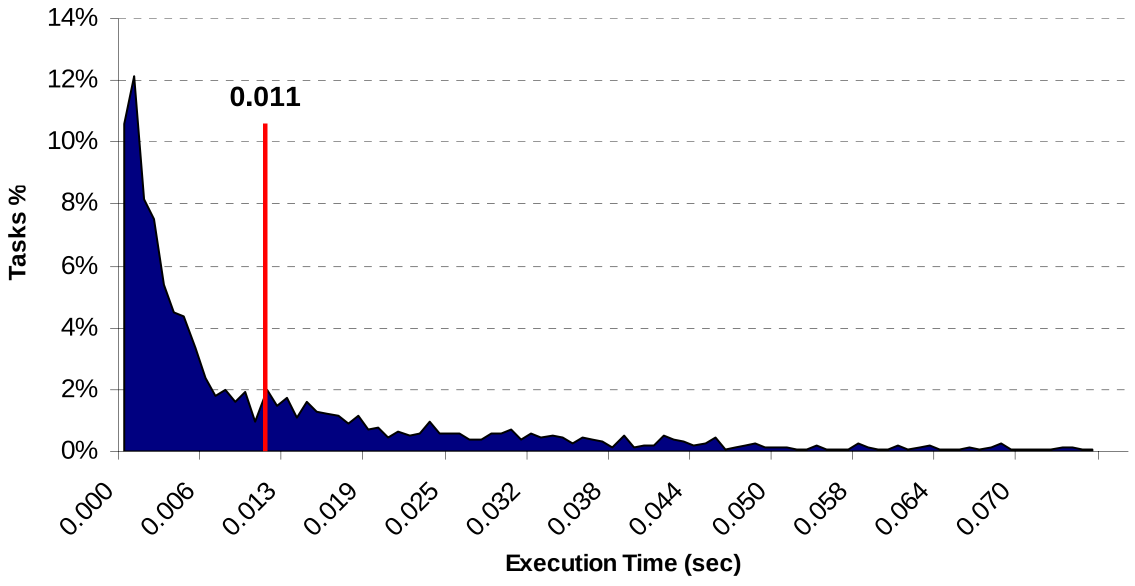


Figure 5-32 Tasks distribution for the *global pruning* TSP algorithm execution of the first dataset with 15 cities in 11 cities branches.

Once the pruning is based in the global minimum there is a high probability of short time execution tasks. The predictability of this algorithm is very low and our proposed model can not be applied for the execution prediction although it could be used for dynamic granularity selection. The average performance value for an execution is unpredictable.

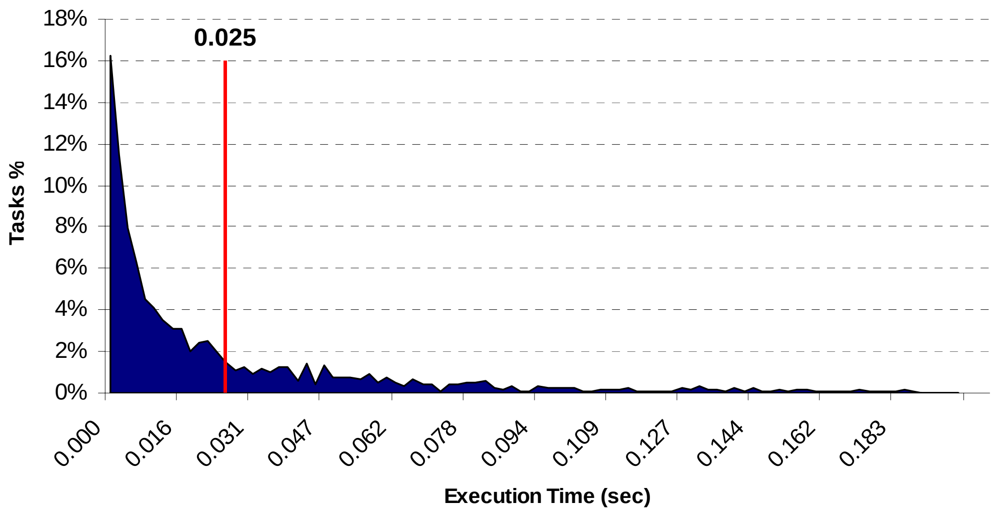


Figure 5-33 Tasks distribution for the *global pruning* TSP algorithm execution of the second dataset with 15 cities in 11 cities branches.

Figure 5-34 shows the performance execution along time for the *global pruning* algorithm with three different sets of 19 cities with tasks of 13 cities. It can be seen in this graphic that there is great variance (from 3 up to 1600 tasks per second) in the obtained performance along time. It can also be seen that for different workloads, quite different execution times was obtained, varying from 1,535 up to 4,644 seconds.

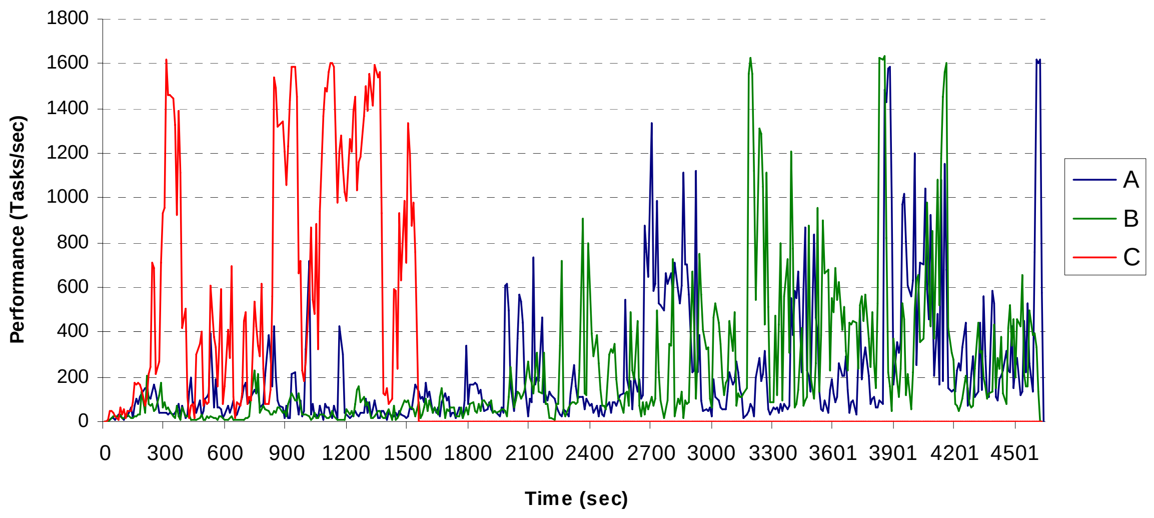


Figure 5-34 Execution performance along time for the task pruning algorithm with different sets of cities in a cluster.

In Table 5-32 it can be seen the dynamic prediction of the execution time using the performance average after 10% of the execution. This table also shows the unpredictable behavior of the algorithm once the prediction error was over 72%.

Table 5-32: Estimated, execution time and prediction error for the TSP task pruning execution with different sets of 16 cities with tasks of 12 cities.

	Estimated Time (sec)		Experimental Time (sec)	Prediction Error
Cities Set	Min	Max		
A	16.717,58	16.733,78	4624,2165	72%
B	24.010,13	24.026,33	4605,7743	81%
C	18.457,48	18.473,67	1535,646	92%

5.5. Conclusions

At this chapter three different applications were used to evaluate the proposed methodology and analytical model: the matrix multiplication, Stochastic Resonant Memory Storage Device and traveling salesman problem.

At the matrix multiplication the possibilities of selecting local cluster and inter-cluster granularities was shown and the model proved to predict with more than 90% precision using over 90% of the available resources performance.

For the SRMSD application the whole methodological steps were detailed and the methodology was used for prediction, for selecting resources in order to improve efficiency and for tuning the application in order to reduce the execution time keeping the efficiency threshold.

The executions results showed the model accuracy for the three scenarios once the experimental execution times were inside the minimum and maximum predicted values.

The traveling salesman problem application was used to prove the robustness of the model with different levels of computation predictability. It was shown that depending on the level of predictability the performance model can be used dynamically with a high level of accuracy.

Chapter 6

Conclusions and Future Work

6.1. Conclusions

Clusters of workstations became a common solution to attain the goals of parallel computing. Along time clusters often become heterogeneous, having computers with different configurations. Heterogeneous clusters of workstations can be found spread over the world in many university departments or companies, directly or indirectly connected to Internet.

Those clusters are often used to solve complexity-increasing applications that demand more power than the one available with a single cluster. A possible way to surpass single-cluster performance limitations is to use Internet to interconnect these clusters into a cooperative multi-cluster.

An important issue for the execution of applications in multi-clusters environments is the efficiency. An efficient execution results in the increase of clusters throughput and in the decrease of the execution direct and indirect associated costs.

Many are the challenges to make possible an efficient execution of applications in multi-clusters:

- Heterogeneity management: Not just in-clusters machines are heterogeneous. There are also throughput, latency and reliability differences between intra-cluster local area network and inter-clusters Internet. Each cluster itself is also different as a whole from others, characterizing another level of heterogeneity.

- Application migration to a multi-cluster: It is an important aspect that the migration of existent applications to a multi-cluster is transparent and scalable. Problems of inter-cluster communication reliability and throughput should be surpassed with, if none, a minimum of changes in applications.
- Speedup achievement: Speedup is a key aspect since a multi-cluster execution can only be justified by a gain on the overall performance and reduction of the execution time.
- Efficiency threshold obtainment: Efficiency on a multi-cluster execution might just be achieved through selection of some of the available resources or maybe just through modifications to tune the application. These alternatives should be evaluated.

The work presented in this thesis addressed to provide tools and strategies to overcome these problems and reduce the execution time of applications through the use of multiple Internet-connected clusters. It is aimed that applications speedup is reached guaranteeing a certain level of efficiency.

This work reached the objectives through the proposal of a system architecture, an analytical performance model and a performance prediction and system tuning methodology.

The proposed system architecture is based on a hierarchical master-worker with the inclusion of communication managers for improving Internet usability and adding reliability to the inter-cluster communication. The system architecture provides to master-worker applications scalability, robustness, efficiency, and transparency on the adaptation to multi-clusters.

The developed analytical model is based in computation-communication characteristics of an application and a multi-cluster system. The analytical performance model evaluates the execution time and efficiency that can be achieved for an application in a multi-cluster system. The analytical model is the base for the proposed performance prediction and system tuning methodology.

The system architecture and analytical model were presented in the paper:

E. Argollo, D. Rexachs, F. Tinetti, and E. Luque, "**Efficient execution of scientific computation on geographically distributed clusters,"** PARA'04 State-of-the-Art in Scientific Computing, Lecture Notes in Computer Science, vol. 3732, pp. 691-698, Feb. 2006.

In order to overcome Internet's problems on latency, bandwidth, and reliability a communication library was built. This library adds reliability to long-distance communication and improves the throughput by a multi-threaded multi-connection strategy. All this process is done transparently through a simple message passing application program interface.

Experiments show that the connection turns to be fault tolerant and the average throughput could be increased in almost six times. The whole system, working with the library was described in the following publication:

E. Argollo, J. d. Souza, D. Rexachs, and E. Luque, **"Efficient execution on long-distance geographically distributed dedicated clusters,"** 11th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 3241, pp. 311-318, 2004

The proposed performance prediction and system tuning methodology targets to guide the analysis and tuning of an application to execute in a multi-cluster guaranteeing an efficiency threshold. The methodology answers if for a specific application it is possible to reduce the execution time through external clusters performance.

The proposed methodology also guides through the selection of clusters and computers in each cluster for which the execution will take place with an efficiency threshold. In the case a better execution time is still required the methodology gives the support to the application tuning analysis.

The proposed methodology was applied to the Stochastic Resonant Memory Storage Device simulation and experiments were done in different phases of the methodology. The paper:

E. Argollo, A. Gaudiani, D. Rexachs, E. Luque. **"Tuning Application in a Multi-cluster Environment"**. European Conference of Parallel Computing (EUROPAR'06). To be published, 2006.

proved the usability of the methodology, reducing the application execution time, without application changes, to 12% of the original single-cluster time with an efficiency over 90% because of the resources selection. When the application was tuned the execution time was reduced to 6% its original time using 94% of the available resources over 90% efficiency.

In order to validate the thesis contributions three applications were selected: the matrix multiplication, the Stochastic Resonant Memory Storage Device and the traveling salesman problem.

The testbed multi-cluster is formed by three geographically distributed clusters located in Argentina, Brazil and Spain. Experiments were done with different multi-cluster configurations to validate the accuracy of the proposed model and the applicability of the proposed methodology and architecture.

The performance model demonstrated over 95% precision and proved to be usable in order to support the application tuning. It was possible to achieve, after the proper tuning, 94% of the available performance in the multi-cluster with the matrix multiplication and 98% with the SRMSD application.

The traveling salesman problem was used to prove the robustness of the model through three different optimal algorithm variations: exhaustive, local pruning and global pruning. The algorithms have different levels of predictability. For the exhaustive algorithm the execution time of tasks is deterministic and independent of the task data, and the performance model achieves a high level of predictability precision.

The local pruning algorithm has an unpredictable average execution time of tasks with data-driven behavior but the variances follow a statistical distribution. For this case the analytical model can be used dynamically, receiving the input parameters values along the execution to improve the prediction precision.

The global pruning algorithm presents a high level of unpredictability. In this case it would be necessary to study the relation between the input data and the application task execution time in order to improve the prediction accuracy. This study is beyond the work of this thesis.

6.2. Future work

This work leaves opened lines for future research. The methodology could be used dynamically at the application startup to dimension, based on historical parameters values, the resources for an efficient execution of the application.

A multi-cluster execution is a dynamic environment and changes in communication (Internet or available LAN throughputs) or computation (process co-allocation at the clusters, computers or clusters failures) might happen along the execution. The methodology can be used to answer to these changes in the system parameters by dynamically adding or removing clusters/computers, guaranteeing the maintenance on the system efficiency.

Depending on the application and on its programming strategy it is also possible to use the system methodology to dynamically tune the granularity inside a cluster or between clusters.

It is possible to study the applicability of the developed proposal on the system architecture, analytical model and performance prediction and tuning methodology for different programming paradigm. The system architecture could, for example, be used as a master-worker based run-time for different paradigms. In this case the computation/communication could be evaluated dynamically allowing the usage of the analytical model and system methodology.

The performance model and the proposed methodology could be part of an execution framework or of an MPI extension, with maybe some specific primitives for giving hints to the model in order to allow the dynamic evaluation or tuning.

The performance prediction and tuning methodology might not just be used to dimension the resources in order to guarantee a threshold of efficiency, as demonstrated in this thesis, but it should be also possible to analyze the possibility of the methodology to reach an efficient execution of an application, with the minimum resources in a specific execution time.

References

Referenced papers and books

[1] Aida, K.; Natsume, W. and Futakata, Y. “Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm,” Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03), pp. 156-163, 2003.

[2] Alizadeh, F.; Karp, R. M.; Newberg, L. A. and Weisser, D. K. "Physical mapping of chromosomes: A combinatorial problem in molecular biology," in Symposium on Discrete Algorithms, pp. 371-381, ACM Press, 1993.

[3] Allcock, B.; Bester, J.; Bresnahan, J.; Chervenak, A. L.; Kesselman, C.; Meder, S.; Nefedova, V.; Quesnel, D.; Tuecke, S. and Foster, I. "Secure, efficient data transport and replica management for high-performance data-intensive computing," in Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies (MSS '01), (Washington, DC, USA), p. 13, IEEE Computer Society, 2001.

[4] Almeida, F.; Gonzalez, D.; Moreno, L.; Rodriguez, C. and Toledo, J. "On the prediction of master-slave algorithms over heterogeneous clusters," in Proceedings of 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing, (Los Alamitos, CA, USA), pp. 433-437, IEEE Computer Society, 2003.

[5] Almeida, F.; Gonzalez, D. and Moreno, L. "The master-slave paradigm on heterogeneous systems: a dynamic programming approach for the optimal mapping," in Proceedings of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04), (Los Alamitos, CA, USA), pp. 266-272, IEEE Computer Society, 2004.

[6] Aras, N.; Altinel, I. and Oommen, J. "A kohonen-like decomposition method for the euclidean traveling salesman problem-KNIES/spl I.bar/DECOMPOSE," IEEE Transactions on Neural Networks, vol. 14, no. 4, pp. 869−890, 2003.

[7] Argollo, E.; Souza, J. d.; Rexachs, D. and Luque, E. "Efficient execution on long-distance geographically distributed dedicated clusters," 11th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 3241, pp. 311-318, 2004.

[8] Argollo, E.; Gaudiani, A.; Rexachs, D. and Luque, E. “Predicción del computo paralelo de una aplicación sobre una colección de clusters geográficamente distribuidos.” (“Parallel Computation Prediction of an Application in a Geographically Distributed Collection of Clusters”). Congreso Argentino de Ciencia de la Computación (CACIC’05). Concordia, Entre Rios, Argentina, 2005.

[9] Argollo, E.; Rexachs, D.; Tinetti, F. and Luque, E. "Efficient execution of scientific computation on geographically distributed clusters," PARA'04 State-of-the-Art in Scientific Computing, Lecture Notes in Computer Science, vol. 3732, pp. 691-698, Feb. 2006.

[10] Argollo, E.; Gaudiani, A.; Rexachs, D. and Luque, E. “Tuning Application in a Multi-cluster Environment”. European Conference of Parallel Computing (EuroPAR’06). To be published, 2006.

[11] Aumage, O. "Heterogeneous multi-cluster networking with the Madeleine III communication library," in Proceedings International Parallel and Distributed Processing Symposium (IPDPS'02), pp. 85-96, IEEE Computer Society, 2002.

[12] Azougagh, D.; Yu, J.-L. and Maeng, S. R. "Resource co-allocation: A complementary technique that enhances performance in grid computing environment," in Proceedings

of 11th International Conference on Parallel and Distributed Systems (ICPADS'2005), vol. 1, pp. 36-42, IEEE Computer Society, 2005.

[13] Bal, H.; Plaat, A.; Bakker, M.; Dozy, P. and Hofman, R. "Optimizing parallel applications for wide-area clusters," Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP'98), pp. 784-790, 1998.

[14] Banikazemi, M.; Sampathkumar, J.; Prabhu, S.; Panda, D. K. and Sadayappan, P. "Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations," in Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99), (Washington, DC, USA), p. 125, IEEE Computer Society, 1999.

[15] Banino, C.; Beaumont, O.; Carter, L.; Ferrante, J. ; Legrand, A. and Robert, Y. "Scheduling strategies for master-slave tasking on heterogeneous processor platforms," IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 4, pp. 319-330, 2004.

[16] Banino, C. "Optimizing locationing of multiple masters for master-worker grid applications," PARA'04 State-of-the-Art in Scientific Computing, Lecture Notes in Computer Science, vol. 3732, pp. 1041-1050, Feb. 2006.

[17] Barbosa, J.; Tavares, J. and Padilha, A. "Linear algebra algorithms in a heterogeneous cluster of personal computers," in Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), pp. 147-159, 2000.

[18] Basney, J.; Raman, R. and Livny, M. "High throughput Monte Carlo," in Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing (PPSC'99), 1999.

[19] Beaumont, O.; Boudet, V.; Rastello, F. and Robert, Y. "Matrix multiplication on heterogeneous platforms," IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, pp. 1033-1051, 2001.

[20] Beaumont, O.; Boudet, V.; Petitet, A.; Rastello, F. and Robert, Y. "A proposal for a heterogeneous cluster ScaLAPAcK (dense linear solvers)," IEEE Transactions on Computers, vol. 50, no. 10, pp. 1052-1070, 2001.

[21] Beaumont, O.; Legrand, A.; and Robert, Y. "The master-slave paradigm with heterogeneous processors," IEEE Transactions on Parallel and Distributed Systems, vol. 14, no. 9, pp. 897-908, 2003.

[22] Beaumont, O.; Legrand, A. and Robert, Y. "Scheduling divisible workloads on heterogeneous platforms," Parallel Computing, vol. 29, pp. 1121-1152, 2003.

[23] Beaumont, O.; Legrand, A. and Robert, Y. “Scheduling Strategies for Mixed Data and Task Parallelism on Heterogeneous Clusters and Grids,” in Proceedings of 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euro-PDP'03), pages 209-216, 2003.

[24] Bucur, A. and Epema, D. "The performance of processor co-allocation in multicluster systems," 3rd International Symposium on Cluster Computing and the Grid (CCGRID'03), p. 302, 2003.

[25] Buyya, R. “High Performance Cluster Computing: Architecture and Systems”. Prentice Hall PTR, chapter 1, vol. 1, 1999.

[26] Buyya, R. “High Performance Cluster Computing: Architecture and Systems”. Prentice Hall PTR, vol. 2, 1999.

[27] Cantu-Paz, E. "Designing efficient master-slave parallel genetic algorithms," in Proceedings of the 3rd Annual Conference Genetic Programming (J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, eds.), (University of Wisconsin, Madison, Wisconsin, USA), p. 455, Morgan Kaufmann, 22-25 1998.

[28] Carusela, M. F.; Perazzo, R.P.J. and Romanelli, L.. “Stochastic resonant memory storage device”. Physical Review, 64(3 pt 1):031101, 2001.

[29] Caymes-Scutari, P.; Morajko, A.; César, E.; Costa, G.; Mesa, J. G.; Margalef, T.; Sorribes, J. and Luque, E. "Development and tuning framework of master/worker

applications," in Automatic Performance Analysis (H. M. Gerndt, A. Malony, B. P. Miller, and W. Nagel, eds.), no. 05501 in Dagstuhl Seminar Proceedings, Internationales Begegnungsund Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2006. .
Available at http://drops.dagstuhl.de/opus/volltexte/2006/505

[30] Choi, J.; Dongarra, J. J.; Pozo, R. and Walker, D. W. “ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers”. Fourth Symposium on the Frontiers of Massively Parallel Computation, pp 120 -127, 1992.

[31] Cierniak, M.; Zaki, M. J. and Li, W. “Compile-time scheduling algorithms for a heterogeneous network of workstations,” The Computer Journal, vol. 40, no. 6, pp. 356-372, 1997.

[32] Diaz-Cuellar, G. and Garza-Salazar, D. “A parallelization technique that improves performance and cluster utilization efficiency for heterogeneous clusters of Workstations,” Proceedings of IEEE International Conference on Cluster Computing (ICCC’02), pp. 275-283, 2002.

[33] Dickens, P.; Gropp, W. and Woodward, P. “High performance wide area data transfers over high performance networks,” in Proceedings International Parallel and Distributed Processing Symposium (IPDPS’02), pp. 254-262, 2002.

[34] Dongarra, J. J.; Croz, J. D.; Hammarling, S. and Duff, I. S. “A set of level 3 basic linear algebra subprograms,” ACM Transactions on Mathematical Software (TOMS), vol. 16, no. 1, pp. 1-17, 1990.

[35] Dongarra, J. J. and Walker, D. W. “Software libraries for linear algebra computations on high performance computers,” SIAM Review, vol. 37, no. 2, pp. 151-180, 1995.

[36] Dongarra, J.; Huss-Lederman, S.; Otto, S.; Snir, M. and Walker, D. “MPI : The Complete Reference”. The MIT Press, Vol. 1, 2nd Edition 1996.

[37] Feng, W. and Tinnakornsrisuphap, P. “The failure of TCP in high-performance computational grids,” in Proceedings of the ACM/IEEE conference on Supercomputing, (Washington, DC, USA), p. 37, IEEE Computer Society, 2000.

[38] Foster, I. “The grid: A new infrastructure for 2lst century science,” Physics Today, vol. 55, no. 2, p. 4247, 2002.

[39] Foster, I. “The Grid 2: Blueprint for a New Computing Infrastructure - Application Tuning and Adaptation”. San Francisco, CA: Morgan Kaufman, second ed., 2003.

[40] Frey, J.; Tannenbaum, T.; Livny, M.; Foster, I. and Tuecke, S. “Condor-G: a computation management agent for multi-institutional,” in Proceedings l0th IEEE International Symposium on High Performance Distributed Computing, pp. 55-63, 2001.

[41] Furtado, A.; Rebouças, A.; Souza, J. R. d.; Rexachs, D. and Luque, E. “Architectures for an efficient application execution in a collection of HNoWs,” Proceedings of the 9th European PVM MPI Users’ Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 2474, pp. 450-460, 2002.

[42] Furtado, A.; Rebouças, A.; Souza, J.; Rexachs, D.; Luque, E. and Argollo, E. “Application execution over a CoHNoWs,” in Proceedings of the International Conference on Computer Science, Software Engineering, Information Technology, e-business, and Application (CSITeA’03), 2003.

[43] Furtado, A.; Rebouças, A.; Souza, J. R. de; Rexachs, D.; Argollo, E. and Luque, E. “Improving performance of long-distance geographically distributed clusters,” in XXXI Seminário Integrado de Software e Hardware (SEMISH’04), 2004.

[44] Gabriel, E.; Resch, M.; Beisel, T. and Keller, R. “Distributed computing in a heterogeneous computing environment.” Proceedings of the 5th European PVM/MPI Users’ Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 1497, pp. 180-187, 1998.

[45] Goux, J.-P.; Linderoth, J. and Yoder, M. “Metacomputing and the master-worker paradigm”. Preprint ANL/MCS-P792-0200, MCS Division, Argonne National Laboratories, Chicago, IL, 2000.
Available at: http://citeseer.ist.psu.edu/goux00metacomputing.html

[46] Goux, J.-P.; Kulkarni, S.; Linderoth, J. T. and Yoder, M. E. "Master-worker: An enabling framework for applications on the computational grid," Cluster Computing, vol. 4, pp. 63-70, 2001.

[47] Grimshaw, A. and Natrajan, A. "Legion: lessons learned building a grid operating system," Proceedings of the IEEE, vol. 93, no. 3, pp. 589-603, 2005.

[48] Gropp, W.; Lusk, E.; Doss, N.; and Skjellum, A. "A high-performance, portable implementation of the MPI message passing interface standard", Parallel Computing, vol. 22, no. 6, pp. 789-828, 1996.

[49] Gutin, G. and Punnen, A. P. "Traveling Salesman Problem and Its Variations". Kluwer Academic Publishers, ISBN: 1402006640, 2002.

[50] Hsu, C.-H.; Lo, T.-T. and Yu, K.-M. "Localized communications of data parallel programs Qn multi-cluster grid systems," European Grid Conference (EGC'05), Lecture Notes in Computer Science, vol. 3470, pp. 900-910, June 2005.

[51] Imamura, T.; Tsujita, Y.; Koide, H. and Takemiya, H. "An architecture of Stampi: MPI library on a cluster of parallel computers," Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, vol. 1908, pp. 200-207, 2000.

[52] Javadi, B.; Akbari, M. and Abawajy, J. "Performance analysis of heterogeneous multi-cluster systems," in Proceedings of the International Conference on Parallel Processing (ICPP'05), pp. 493-500, 2005.

[53] Karonis, N. T.; Supinski, B. R. de; Foster, L T.; Gropp, W. and Lusk, E. L. "A multilevel approach to topology-aware collective op erations in computational grids," Mathematics and Computer Science Division, Argonne National Laboratory. Technical Report, vol. cs.DC/0206038, 2002.

[54] Karonis, N. T.; Toonen, B. and Foster, I. "MPICH-G2: A grid-enabled implementation of the message passing interface." Journal of Parallel and Distributed Computing (JPDC'03), vol. 63, pp. 551-563, May 2003.

[55] Kielmann, T.; Bal, H. E.; Maassen, J.; van Nieuwpoort, R.; Eyraud, L.; Hofman, R. and Verstoep, K. “Programming environments for high-performance grid computing: the albatross project,” Future Generation Computer Systems, vol. 18, pp. 1113-1125, Oct. 2002.

[56] Kishimoto, Y. and Ichikawa, S. “An execution-time estimation model for heterogeneous clusters,” in Proceedings of the l8th International Parallel and Distributed Processing Symposium (IPDPS’04), p. 105, 2004.

[57] Korostensky and Gonnet, “Using traveling salesman problem algorithms for evolutionary tree construction,” BIOINF: Bioinformatics, vol. 16, no. 7, pp. 619-627, 2000.

[58] Lee, C.; DeMatteis, C.; Stepanek, J. and Wang, J. “Cluster performance and the implications for distributed, heterogeneous grid performance,” in Proceedings of the 9th Heterogeneous Computing Workshop (HCW’OO), pp. 253-261, 2000.

[59] Lee, C.; Coe, E.; Michel, B.; Stepanek, J.; Solis, I.; Clark, J. and Davis, B. “Using topology-aware communication services in grid environments,” in Proceedings of the 3rd IEEE ACM International Symposium on Cluster Computing and the Grid (CCGrid’03), pp. 534-541, 2003.

[60] McNamara, B.; and Wiesenfeld, K. “Theory of stochastic resonance,” Physical Review A, vol. 39, no. 9, p. 4854-4869, 1989.

[61] Moscicki, J. “DIANE - distributed analysis environment for grid-enabled simulation and analysis of physics data,” Nuclear Science Symposium Conference Record, vol. 3, pp. 1617-1620, 2003.

[62] van Nieuwpoort, R. V.; Kielmann, T. and Bal, H. E. “Efficient load balancing for wide-area divide-and-conquer applications,” in Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPoPP’01), (New York, NY, USA), pp. 34-43, ACM Press, 2001.

[63] Pant, A. and Jafri, H. “Communicating efficiently on cluster based grids with MPICH-VMI,” in Proceedings of the 2004 IEEE International Conference on Cluster Computing, pp. 23-33, IEEE Computer Society, 2004.

[64] Pastor, L. and Bosque, J. L. “An efficiency and scalability model for heterogeneous clusters.,” in Proceedings of the 2001 IEEE International Conference on Cluster Computing, (Washington, DC, USA), p. 427, IEEE Computer Society, 2001.

[65] Pepper, J.; Golden, B. and Wasil, E. “Solving the traveling salesman problem with annealing-based heuristics: a computational study,” IEEE Transactions on Man and Cybernetics Systems, Part A, vol. 32, no. 1, pp. 72-77, 2002.

[66] Plaat, A.; Bal, H. E.; Hofman, R. F. H. and Kielmann, T. “Sensitivity of parallel applications to large differences in bandwidth and latency in two-layer interconnects,” Journal on Futures Generations of Computer Systems, vol. 17, no. 6, pp. 769-782, 2001.

[67] Postel, J. “Transmission Control Protocol”, RFC793, 1981.

[68] Pruyne, J. and Livny, M. “Interfacing Condor and PVM to harness the cycles of workstation clusters”, Journal on Futures Generations of Computer Systems, vol. 12, pp. 67-85, 1996.

[69] Sankoff, D. and Blanchette, M. “The median problem for breakpoints in comparative genomics,” in Proceedings of the 3rd Annual International Conference on Computing and Combinatorics (COCOON’97), (London, UK), pp. 251-264, Springer-Verlag, 1997.

[70] Shao, G.; Berman, F. and Wolski, R. “Master slave computing on the grid,” in Proceedings of Heterogeneous Computing Workshop, pp. 3-16, 2000.

[71] Sivakumar, H.; Bailey, S. and Grossman, R. L. “PSockets: the case for application-level network striping for data intensive applications using high speed wide area networks,” in Proceedings of the 2000 ACM IEEE Conference on Supercomputing, (Washington, DC, USA), p. 37, IEEE Computer Society, 2000.

[72] Souza, J.; Argollo, E.; Duarte, A.; Rexachs, D. and Luque, E. “Fault tolerant master-worker over a multi-cluster architecture,” in Proceedings of Parallel Computing (ParCo’05), 2005.

[73] Sterling, Th.; Salmon, J.; Becker, D. and Savarese, D. "How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters". The MIT Press, 1999.

[74] Tsai, H.-K.; Yang, J.-M. and Kao, C.-Y. "Solving traveling salesman problems by combining global and local search mechanisms," in Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02), vol. 2, pp. 1290-1295, 2002.

[75] Wang, L.; Maciejewski, A.; Siegel, H. and Roychowdhury, V. "A comparative study of five parallel genetic algorithms using the traveling salesman problem," in Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS SPDP'98), pp. 345-349, 1998.

[76] Williams, K. and Williams, S. "Implementation of an Efficient and Powerful Parallel Pseudo-random Number Generator".
Available from http://citeseer.ist.psu.edu/37730.html .

[77] Wilkinson, B. and Allen, M. "Parallel Programming - Techniques and Applications Using Networked Workstations and Parallel Computers", Prentice Hall, New Jersey, USA, 1999.

Referenced web links

[L1] Automatic Tuning Linear Algebra Software (ATLAS).
http://math-atlas.sourceforge.net/
Current: May/2006.

[L2] Beowulf.org.
http://www.beowulf.org/
Current: May 2006

[L3] LAM/MPI Parallel Computing.
http://www.lam-mpi.org/
Current: May, 2006.

[L4] Message Passing Interface (MPI) Forum.
http://www.mpi-forum.org/
Current: May 2006.

[L5] MPICH – A Portable Implementation of MPI.
http://www-unix.mcs.anl.gov/mpi/mpich/
Current: May 2006.

[L6] MPICH-G2 – A grid enabled MPI implementation.
http://www3.niu.edu/mpi/
Current: May 2006.

[L7] Virtual Machine Interface 2.1
http://vmi.ncsa.uiuc.edu/
Current: May 2006.

[L8] Wind: The Production Flow Solver of the NPARC Alliance, AIAA 98-0935
http://www.grc.nasa.gov/www/winddocs/aiaa98/aiaa-98-0935.html
Current: May, 2006.

Appendix A
LDS Library

The use of Internet might result in some problems concerned to the communication. The most commons problems are high communication latency, unpredictable throughput and low bandwidth when compared to a local-area network. The use of long-distance standard Internet not just intensifies those problems but also adds a new and important one: the lack of reliability.

Figure A-1 shows the route (traceroute program execution) for the long-distance Internet communication between the Computer Science departments at University Autonoma of Barcelona, in Spain, and Catholic University of Salvador, in Brazil. It can be seen each host through which a message needs to pass to go from Spain cluster to Brazil, and its influence in the latency. The amount of 15 hosts from the origin to the destiny is considerable and denotes some of the difficulties of long-distance Internet.

```
 1  158.109.64.1 (158.109.64.1)  0.829 ms  1.113 ms  0.798 ms
 2  anella-uab.cesca.es (193.147.232.9)  2.218 ms  2.810 ms  2.149 ms
 3  GE1-0-0.EB-Barcelona0.red.rediris.es (130.206.202.1)  2.261 ms  2.247 ms  2.249 ms
 4  CAT.SO2-1-0.EB-IRIS2.red.rediris.es (130.206.240.9)  17.861 ms  17.593 ms  17.576 ms
 5  SO0-0-0.EB-IRIS4.red.rediris.es (130.206.240.2)  17.435 ms  17.951 ms  17.678 ms
 6  rediris.es1.es.geant.net (62.40.103.61)  18.132 ms  18.002 ms  17.667 ms
 7  es.it1.it.geant.net (62.40.96.186)  40.257 ms  39.824 ms  40.193 ms
 8  it.de2.de.geant.net (62.40.96.61)  49.580 ms  49.201 ms  49.448 ms
 9  abilene-gw.de2.de.geant.net (62.40.103.254)  143.361 ms  143.901 ms  143.538 ms
10  atlang-washng.abilene.ucaid.edu (198.32.8.65)  159.092 ms  159.140 ms  159.066 ms
11  abilene-oc3.ampath.net (198.32.252.253)  177.465 ms  177.943 ms  177.475 ms
12  rnp.ampath.net (198.32.252.238)  285.899 ms  281.244 ms  280.907 ms
13  ba-serial4-1-0.bb3.rnp.br (200.143.253.90)  312.897 ms  312.338 ms  312.963 ms
14  200.128.6.171 (200.128.6.171)  312.447 ms  312.420 ms  313.337 ms
15  * * *
```

Figure A-1: Spain to Brazil traceroute result.

The communication is not guaranteed at long-distance Internet even when the reliable transport-layer protocol (TCP) is used. At short time executions (hours), this problem is not representative since it is not frequent, but it becomes essential on a long duration execution (days to weeks). It is important to remark that a break on the communication can cause the whole collaboration from one cluster to the other to be lost.

This reliability problem was discovered when intercommunication tests were made with PACX MPI library. The system proved not to maintain the communication for long periods of time: Internet's transport-layer protocol (TCP) communication sockets presented unexpected disconnections. After those failures, to prove that the problem was on the network itself, direct TCP communication was tested. The same results were gotten: sudden disconnections. It was impossible to determine for how long the communication would stay alive and active.

Since all others MPI implementations are based on the TCP protocol, it became necessary to build an MPI level library that, with a high-level interface to the user, could maintain the communication for long-duration, tolerating possible Internet failures transparently. For the user, a one-hour disconnection should represent just one hour with no communication, since the connection is maintained and all exchanged data is guaranteed to be correctly communicated.

It is important to remark that this reliability at inter-clusters communication level was necessary not just because of Internet's transport protocol failures. This feature is also relevant because Internet, and specially standard long-distance one, is not a controlled environment like a LAN. When targeting long-duration executions other factors like a router maintenance or physical network malfunction can also cause communication failures.

The efforts to solve all the presented problems evolved into the development of a library called Long Distance Service (LDS). LDS not just adds transport layer reliability, but also exploits the best of long-distance communication throughput and latency in order to obtain the best use of Internet link. The reason to be done as a library is that, through this, all features are transparent to the user. The library interface is similar and as easy to use as popular MPI standard.

This appendix introduces the LDS library. Section A.1 describes the library development goals. A layered model and each layer details are presented at Section A.2. Then, at Section A.3, we present experiments made with different configurations and their results. Finally, conclusions and future work are shown at Section A.4.

A.1 Goals

To communicate long-distance geographically distributed clusters through Internet is not an easy task and presents some problems. Among the difficulties the most important are high latency, low throughput, variable bandwidth and, specially for long-duration communications, lack of reliability.

LDS library was built to be an easy to use communication tool that could, transparently to the user, implement strategies to overcome those problems. LDS main goals are:

- Add reliability to the communication

  To the end user the communication between clusters must never fail and the program execution must continue even if data can not be effectively sent or received. The main reason for this is that once the execution can be as long as weeks, the fact that the Internet link is not available for hours or even days does not mean the collaboration could not be achieved. During the disconnection, both clusters can be executing and generating result data that can be sent after the communication problem is solved.

- Implement strategies to exploit in a maximum the throughput and its peaks.

  Since Internet's throughput varies along time, it presents peaks of performance. A buffering strategy could be used to have enough data to exploit properly those peaks increasing the available throughput usage. One other thing that can be done to improve the effective throughput, in case of long-distance communications, is to use at once more than on simultaneous connections.

- Avoid the interference of low Internet bandwidth at the high-speed LAN.

  It is important that one cluster element does not get stuck in feeding the other cluster with data. There should be buffering strategies, asynchronous sending, and ways to the user to know that a long-distance communication can be done without interfering on its performance and local communication response time.

- Be easy to use and transparent to the end user.

All those functionalities must be provided by a simple and standardized interface and all complexity details should be hidden to the user, although the user must have the possibility to configure the library for specific needs.

Each of those goals can be solved at different levels on which the upper level does not need to be concerned about the problems solved at the lower level. To do this LDS is built over a layered model. The upper layer, closest to the end user, is responsible for the easy usage and transparency and the lower layer (closest to the network) solves the lack of reliability. The middle layers try to exploit the best performance from the throughput by using a buffering strategy and multiples connections through multiples threads. At the next section we introduce the layered model in more details.

A.2 Layered Model

LDS is programmed in C++ (where each layer can be mapped to one class) and the library users can accede to its functionalities through standard C functions, representing the API.

The Long Distance Service (LDS) is a library that can be used to attain efficient and reliable long-distance point-to-point communication over Internet. It works on the same MPI principles, providing a simple application program interface (API) to allow workstations data exchange through message passing mechanism. A example of LDS message flow use can be seen at Figure A-2.

LDS implements strategies to obtain the best of long-distance Internet characteristics, adding fault tolerance on this communication, taking advantage of its peaks of performance and isolating its traffic and low latency from LAN communication.

The functionalities and strategies are implemented in layers that goes from the high-level user API to the low-level network exchange of data. The layered approach allows the possibility of evolution since a layer can be easily replaced.

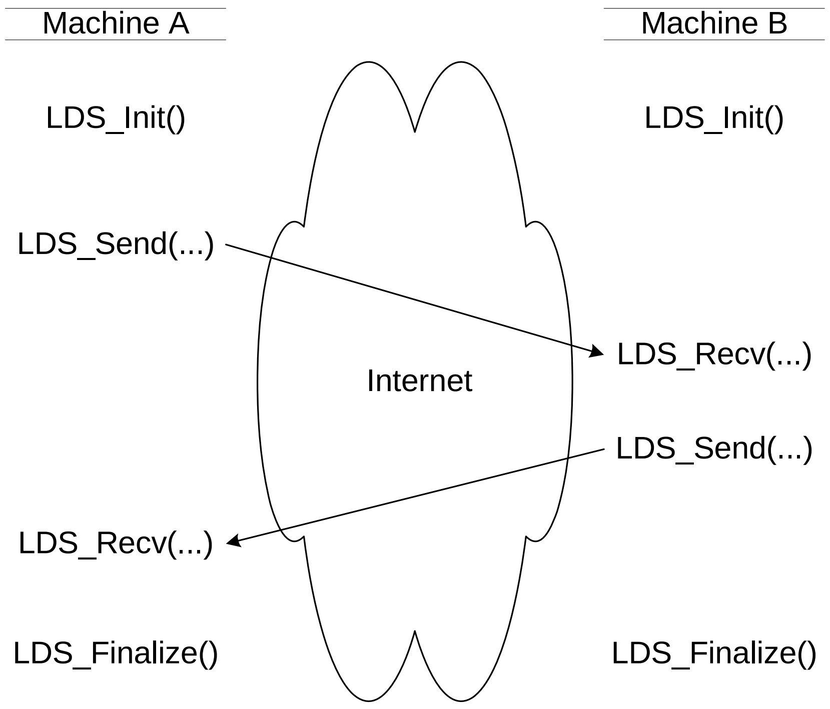


Figure A-2: LDS message flow use example.

LDS layers from the user to the hardware are: Message passing and management API layer, Buffering administration layer, Multi-pipe layer, and Connection Controlled layer. The Connection controlled layer uses standard TCP/IP protocol suite functionalities (Figure A-3).

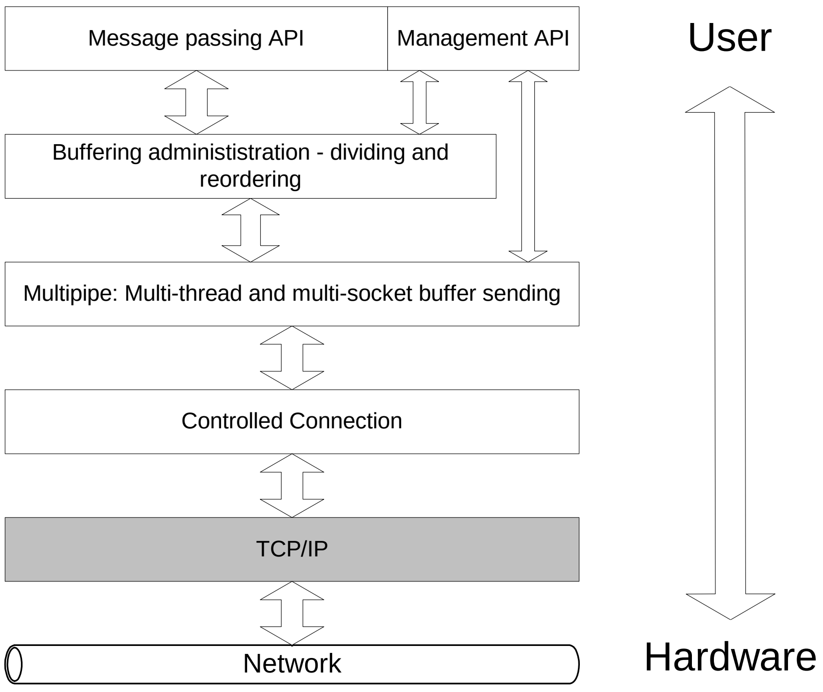

Message passing API
Management API
User
Buffering administistration - dividing and reordering
Multipipe: Multi-thread and multi-socket buffer sending
Controlled Connection
TCP/IP
Network
Hardware

Figure A-3: LDS layered model.

At the bottom of LDS layered model there is the Controlled Connection layer. The Controlled Connection layer adds failure detection and timeout functionality to standard TCP sockets so that failures can be communicated to and resolved by the Multi-pipe layer

The Multi-pipe layer is responsible for the instantiation of multiple peer-to-peer Controlled Connections, each one with one new thread. The Multi-pipe layer adds reliability for those connections and uses them to simultaneously try to send or receive the obtained buffers (at the Buffering Administration layer) through the long-distance link. The use of multiple connections allows the best use of the communication link.

The Buffering Administration layer is responsible for the division or reordering of a message to the send buffers or from the receive buffers. It is also responsibility of this layer the distribution and concurrence control of those buffers for the multiple threads of the Multi-pipe layer.

The Message Passing and Management API is the interface with the user. It provides to the user a series of functions and a configuration file. All user actions over the service are only possible through this layer. The process of a message communication through LDS layers can be seen at Figure A-4.

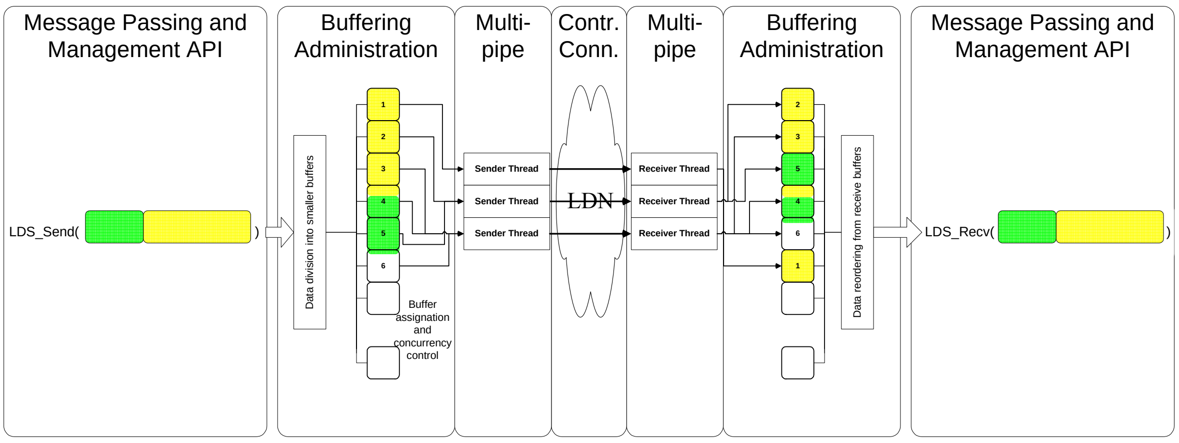


Figure A-4: Message communication process through the layered model.

The next section explains each layer in more details.

A.2.1 Message passing and management API layer

At LDS top level there are the functions that work as interface to the library user. Those functions can be divided into two categories: the message passing functionality and the whole communication service management. The idea is to provide a message passing service that starts with the initialization function (LDS_Init) and ends with a finalization function (LDS_Finalize). Between those functions send and receive message operations can be done (LDS_Send and LDS_Recv) as seen in Figure A-2. The initialization function is responsible for classes instantiation, memory allocation, threads start, and connections establishments, following the configuration file. Send and receive functions use the Buffer Administration layer services, and the finalization function disconnects and free all allocated resources.

All functions start with LDS prefix and LDSM for management functions. The management function to change the configuration file is for example LDSM_SetConfigFile. No message passing function should be called before LDS_Init or after LDS_Finalize.

At the current version there is only one management function. This function is responsible to set the configuration file (in case of not using the default) on which all service's parameters are set and it should be called before the service initialization, if needed.

On future versions several functions are going to be responsible for the library configuration changes and they can be divided in two types: static and dynamic management functions. The static ones (LDSM_st) will just be available for call before the service starts and will mainly be responsible for overall section configuration. Dynamic ones (LDSM_dy) will be responsible for either on-the-fly configuration changes or for returning system conditions or statistics, like overall bandwidth or last minute throughput.

The available functions at current version API are:

int LDSM_SetConfigFile(char *filename)

This function sets the configuration file in case the default file is not used. It must be called before the service initialization and it returns an error code in case the file passed as parameter does not exist or 0 in case of success.

The configuration file is a text file responsible for setting the system configuration. Its parameters and its properties are shown in Table A-1.

Table A-1: Configuration file parameters.

Parameter	Type	Status	Default	Data	Explanation
Server	Integer	Obligatory	-	1 for server and 0 for client	Indicates if the service to be initialized is client or a server.
ServerHostName	String	Obligatory if Server=1	-	Name of server host.	Indicates the name of the workstation that will be the communication server.
BlockSize	Integer	Optional	10000	Number representing bytes.	Represents the amount of bytes of the Buffering Administration layer block.
SendMemory	Integer	Optional	1000000	Number representing bytes.	Represents the total amount of bytes that will be available for Buffer Administration layer send messages buffer. A multiple of BlockSize is desirable.
RecvMemory	Integer	Optional	1000000	Number representing bytes.	Represents the total amount of bytes that will be available for Buffer Administration layer receive messages buffer. A multiple of BlockSize is desirable.
SenderPort	Integer	Optional	60000	Number	Represents the first socket port used by sender threads.
SenderPort	Integer	Optional	61000	Number	Represents the first socket port used by receiver threads.

The default configuration file is called LDN_Service.ini. An example of a client configuration file is shown at Figure A-5.

Config file

```
Server = 0
ServerHostName  =  infoquir2
BlockSize =   40000
SendMemory  =  1200000
RecvMemory  =  1200000
SenderPort  =  60000
ReceiverPort  =  60100
```

Figure A-5: Configuration file example.

int LDS_Init(int aitCommInstances)

This must be the first message passing function call in the program. Just after the initialization it is possible to exchange messages with data. This function initializes objects, allocates all necessary memory and establish the communication system, data to the target configuration file.

This function receives as a parameter the amount of communication instances (aitCommInstances) that will be generated for communication. This number represents the numbers of communication pipes that will be opened at Multi-pipe layer. This function call returns 0 case no errors were found, or an error code in case something fails.

int LDS_Rank(void)

This function can be used to know if this application is a client or a server. In case it is a client the function call returns 0, and it returns 1 in case of being a server.

int LDS_Send(void *buf, int count)

This function is responsible for sending a message with data from one point to the other. The message is at *buf* address and has *count* bytes. This function returns an error code in case of failure, or 0 in case of success.

This function does not block the program execution until there are free space within the send message buffers. In case the buffers are full then the system execution is blocked.

int LDS_ Recv(void *buf, int count)

This function is responsible for receiving a message with data. The message is filled *buf* address and will have *count* bytes. This function returns an error code in case of failure, or 0 in case of success.

The system keep receiving data asynchronously through the use of threads. This means that there are possibilities that the expected data is already at the buffers. In this case the function call will not block, returning instantaneously the required data. In case all the requested data is not at the receive buffers, the function will block.

int LDS_Finalize(void)

This function finalizes the service, waiting for all messages that are on its way, closing all connections, freeing all buffers and ending the communication system. After this function call, no more communication is possible.

A.2.2 Buffering Administration layer

The Buffering Administration layer is responsible for managing the buffers for sending or receiving data. Send and receive operations can be done simultaneously through multiple Multi-pipe layer threads.

To manage the buffer for a send operation means receiving from the API layer the data, dividing it into the buffers, managing the send threads concurrence to this data, managing the Multi-pipe layer confirmation of reception of the buffers, and freeing buffers confirmed. It is important to remark that, since latency is considerable, the system does not wait for a received confirmation to assign another buffer to be sent. Confirmations are asynchronous.

At the other side, buffers will be received and the reception confirmation will be sent. Buffers are received independently of a existence, at the API layer, of a receive function call and this layer controls the buffer assignation to the threads. Whenever the receive function is called, buffers will be reordered and joined at the receive data area until it reaches the amount of required data. The buffers that were used are then freed.

Figure A-6 shows the Buffering layer workflow for a send operation and its interactions with the Multi-pipe layer. The data reception works in a similar way.

It is important to remark that the size of LDS communication packet (the buffer packet size) have influence on the recovery time, on the buffer memory allocation, and on the communication overhead.

For the recovery time, the increase of the packet size means that, when a failure occurs, more time was lost at the transmission that was being made, and that there will be more data to be communicated to recover this failure. Concerning just the failure recovery, a buffer as small as possible would be best idea.

The packet size is important for the buffer memory allocation since each buffer has to have this size. As the system can keep the communication when a single failure occurs, it is desirable the existence of a great amount of buffers. The more the buffers, the bigger the probability of keep communicating until a failure is corrected. A small packet size would also be the best option.

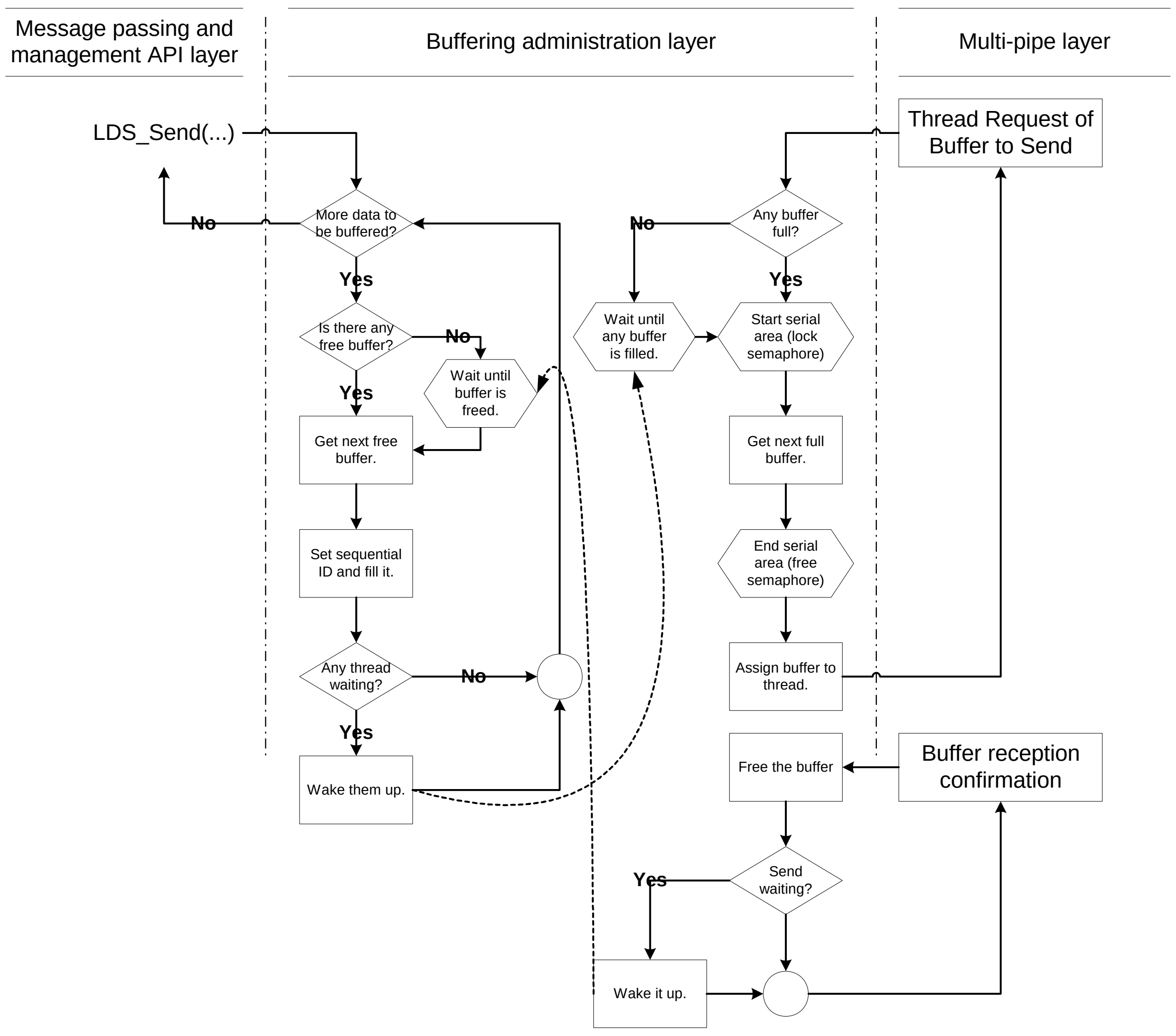

Message passing and management API layer
Buffering administration layer
Multi-pipe layer
LDS_Send(...)
More data to be buffered?
No
Yes
Is there any free buffer?
No
Wait until buffer is freed.
Yes
Get next free buffer.
Set sequential ID and fill it.
Any thread waiting?
No
Yes
Wake them up.
Thread Request of Buffer to Send
Any buffer full?
No
Yes
Wait until any buffer is filled.
Start serial area (lock semaphore)
Get next full buffer.
End serial area (free semaphore)
Assign buffer to thread.
Free the buffer
Buffer reception confirmation
Send waiting?
Yes
Wake it up.

Figure A-6: Buffering layer workflow and interfaces for a send operation.

On the other hand, regarding the communication overhead, the smaller the packet the more representative will be the protocol headers and trailers. This means that small packets communication will waste part of the available throughput with overhead.

The conclusion is that the communication packet should be as small as possible so that recovery can be efficient and as big as possible so that there is not much waste of throughput.

A.2.3 Multi-pipe layer

The Multi-pipe layer is responsible for the creation of a reliable virtual communication link between two workstations. To increase throughput multiple threads for communication can be involved. The aim is to achieve simultaneous send and receive of data, and also be prepared to tolerate and solve possible communication failures through reconnections and resending of lost data.

The use of simultaneous communication proved to be an efficient alternative, reaching even 6 times the single connection throughput. Some authors also use the idea of using multiples simultaneous connections, striping the data over several sockets [3][71].

When LDS service is started up, the user passes to the initialization function (LDS_Init) the number of desired simultaneous connections. For each connection, at each workstation, a pair of threads (one for data send and the other for data reception) and two Controlled Connection layer sockets (outcoming and incoming) are created. Figure A-7 is an example of the virtual connection threads when three was the simultaneous startup connections value.

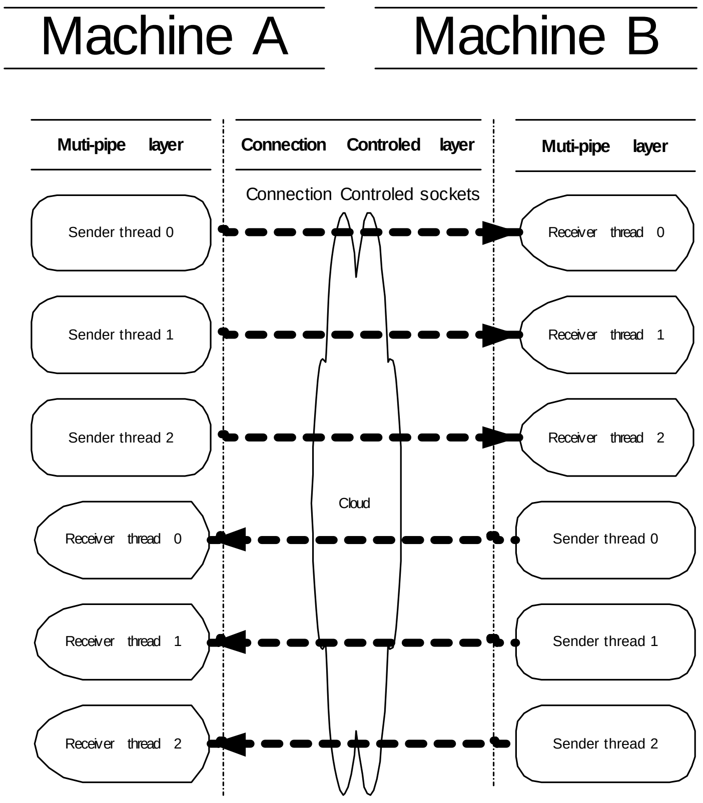


Figure A-7: Multi-thread layer communication threads for three connections.

Each Multi-pipe layer sender thread has one Controlled Connection socket to a receiver thread. This thread is continuously sending buffers and asking the Buffering Administration layer for new buffers to be sent. The Multi-pipe layer is the active system component responsible for checking the status of the send and receive operations. It is at this layer that the system fault tolerance is provided.

At the Buffering Administration layer, communication buffers are sequentially numbered with an ID. This ID indicates the order so that the Buffering Administration layer at the receiver can proceed with the reordering service. This ID is used at the Multi-pipe layer to reach the fault tolerance.

Each sender communication thread contains a list of sent buffers which reception is still not confirmed, the non-confirmed list (NCL). This list exists because long-distance communication through Internet presents high latency and a great amount of hosts buffering packets from both endpoints. It would then not be efficient to wait for one buffer confirmation of reception to start sending another one.

The sender thread keeps continuously sending buffers and putting all the non-confirmed at NCL. Each send operation gets all available confirmations and returns then so that the list can be reduced. The NCL is used in case of communication failure: the buffers at this list are resent once the communication is reestablished. The sender thread workflow can be seen at Figure A-8.

The reconnection and resending action brings reliability to the system. The Buffering Administration layer dividing and reordering brings the possibility of the existence of multiple simultaneous threads.

The receiver thread workflow is quite simpler since all the receiver does is ask for free buffers, receive data, and send data reception confirmation. This workflow is illustrated at Figure A-9.

A.2.4 Controlled Connection layer

At the base of the LDS service there is the layer responsible for the data exchange. This layer uses the standard TCP/IP Internet protocol, adding to it a special failure detection reporting to the Multi-pipe layer disconnections or long periods inactivity.

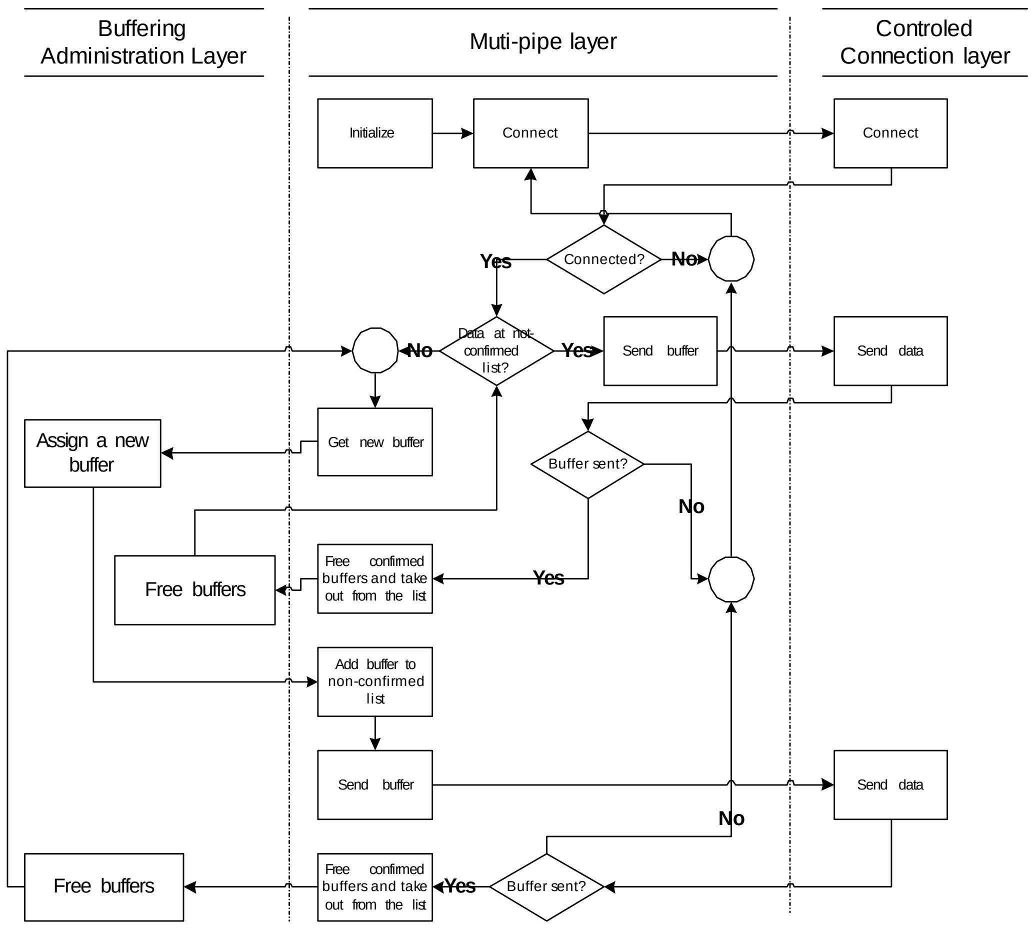

Buffering Administration Layer
Muti-pipe layer
Controled Connection layer
Initialize
Connect
Connect
Connected?
Yes
No
Data at not-confirmed list?
No
Yes
Send buffer
Send data
Assign a new buffer
Get new buffer
Buffer sent?
No
Free buffers
Free confirmed buffers and take out from the list
Yes
Add buffer to non-confirmed list
Send buffer
Send data
No
Free buffers
Free confirmed buffers and take out from the list
Yes
Buffer sent?

Figure A-8: Sender thread workflow.

It is important to remark that once the standard reliable Internet transport protocol (TCP) proved not to work properly in the described conditions, presenting failures through broken pipes, the natural way to add the desirable reliability would be to implement the communication over the unreliable Internet suite protocol (UDP).

Despite that, our decision was to keep using the standard transport protocol, since it is a mature solution with several important features implemented like data-flow control. One other reason for using TCP as the base is that although disconnections occur, they do not happen frequently.

The Controlled Communication layer is then a thin layer over TCP responsible to detect communication failures and allow the reconnection of the broken TCP communication without causing the whole system to halt.

The Controlled Connection layer not just returns the common TCP disconnection but also adds an inactivity timeout message that will be interpreted as a disconnection. The reason for this is that the disconnection can sometimes be received after 15 to 30 minutes of data transfer inactivity. This happens because TCP is still trying to keep communication alive.

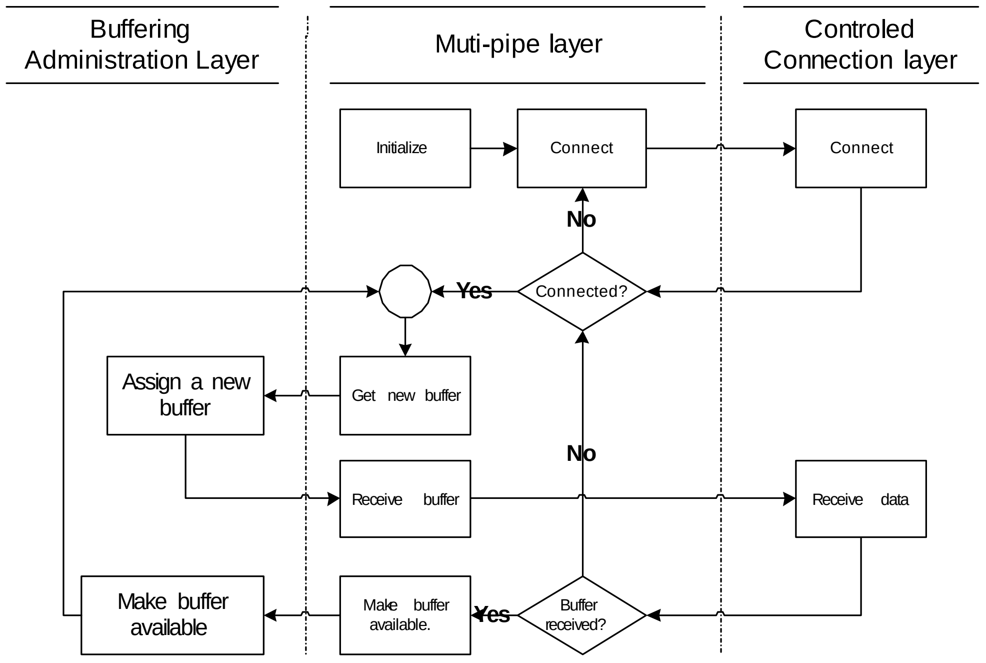


Figure A-9: Receive thread workflow.

To wait such a long time without communicating is not efficient and to avoid this behavior, the Controlled Connection layer was provided with a timeout option. Whenever a send or receive operation is performed, if a specific time is passed without the exchange of even one byte of data, the layer assumes a disconnection will occur and returns to the Multi-pipe layer this information.

This simple idea proved to be effective on reducing the time to overpass connection failures. It is implemented through the combination of the use of non-blocking sockets with a system call to check for a period of time if data was communicated. This send and receive process can be seen at Figure A-10.

A.3 Experiments and results

To prove the effectiveness and efficiency of LDS library a benchmark application was built. This application tests the communication and meters the throughput between two hosts connected through long-distance Internet. One of the hosts, called sender, continuously sends messages of a determined packet size and the other host, called receiver, receives those messages and records the communication throughput statistics.

One thread is launched to periodically record the period average throughput. The benchmark program workflow can be seen at Figure A-11.

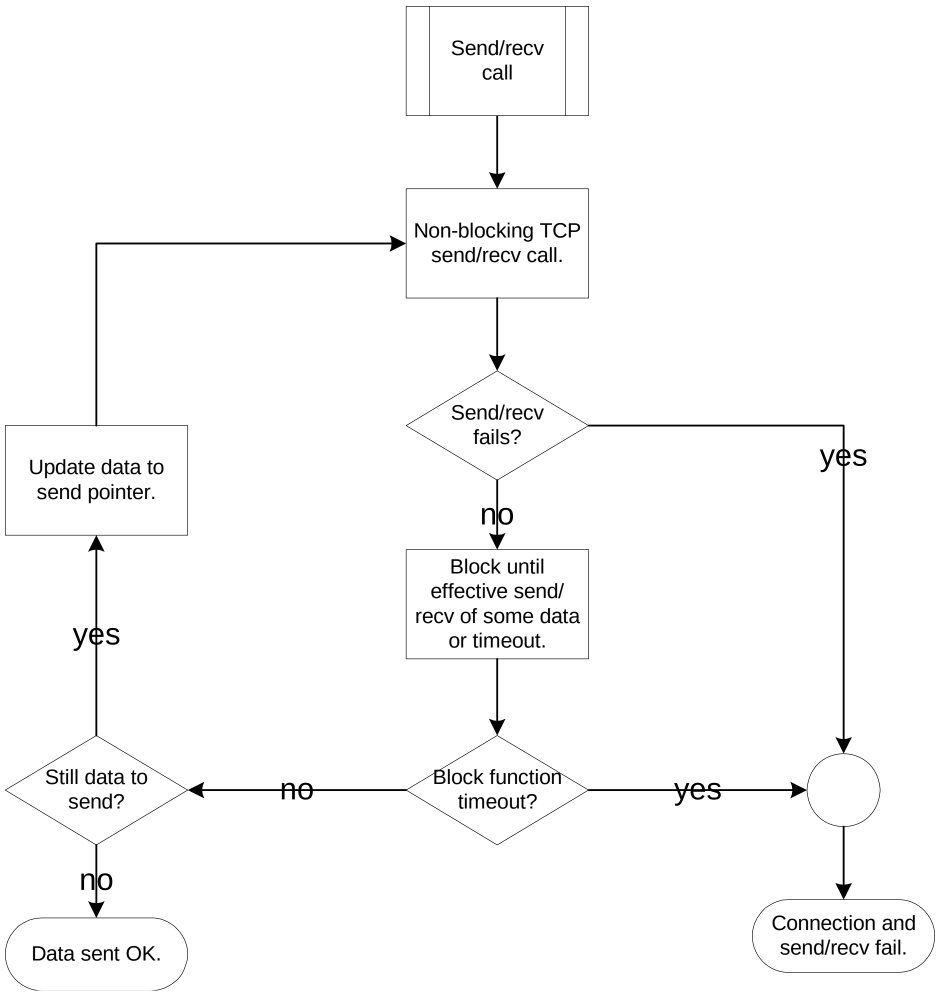

Send/recv call
Non-blocking TCP send/recv call.
Send/recv fails?
yes
no
Update data to send pointer.
Block until effective send/ recv of some data or timeout.
yes
Still data to send?
no
Block function timeout?
yes
no
Data sent OK.
Connection and send/recv fail.

Figure A-10: Send and receive with timeout and connection failure detection.

The execution parameters for this application are the number of simultaneous threads (and consequently connections). The amount of threads has influence on the throughput: the tendency is an increment on the throughput when one more thread is added until a point on which the network is saturated.

The application output is a file with the throughput average in each period of a defined amount of time. For the presented results 1 minute is considered to be the measuring period. One other output is also obtained by the LDS system: failure log file. The failure log file present the moment a disconnection occurred, the moment the system was able to reconnect and the moment all lost data was correctly retransmitted. This data is only available for the LDS system since to the end user the network presents to failures.

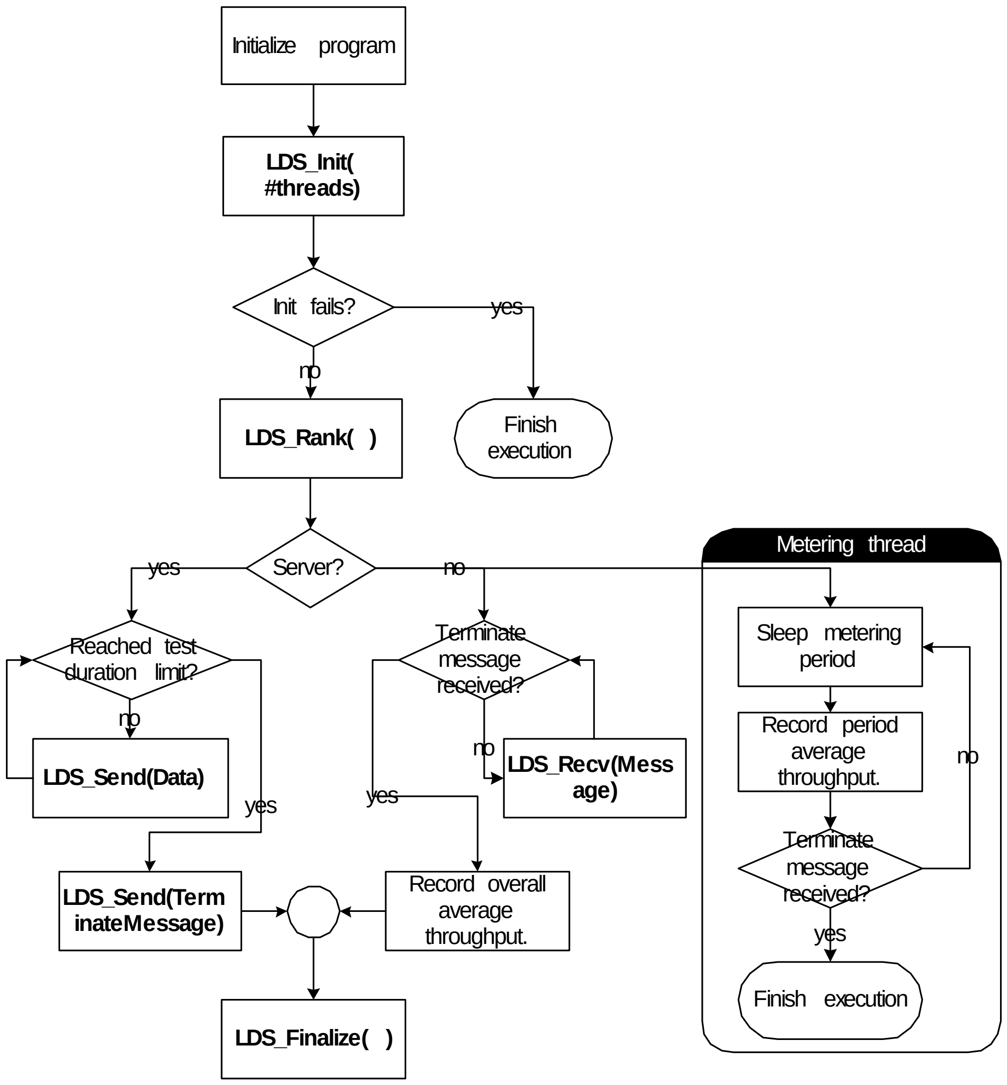


Figure A-11: Benchmark application workflow.

Three are ours key goals at the experiments: prove that the library is reliable to TCP disconnections, show the increase of performance with the multithreads use, and experimentally determine the maximum possible bandwidth, i.e. the saturation point and the amount of threads to reach this point.

For the presented experiments, LDS configuration file was set so that 100 buffers of 10 kilobytes are used. At the application level messages of 100kbytes are sent. Experiments were executed between Spain and Brazil for 1, 2, 4, 6, 8, 10, and 16. Throughput behavior along time is shown in Figure A-12.

The average execution throughput, the number of disconnections and a relative gain representing the ratio between this execution and the single thread one for Spain-Brazil execution can be seen in Table A-2.

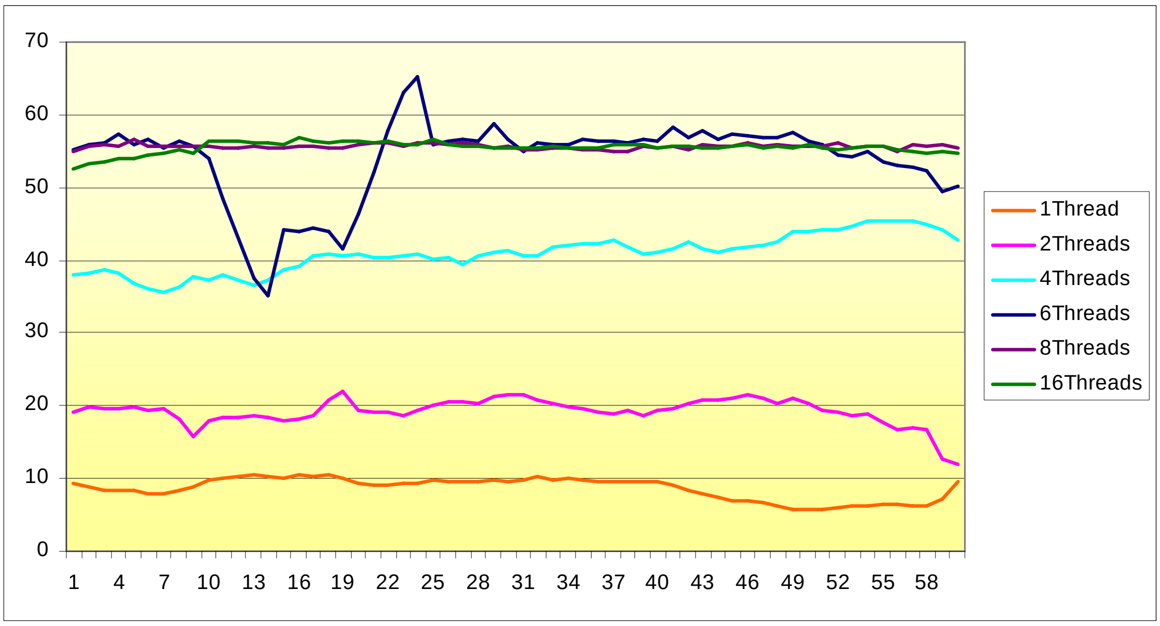


Figure A-12: Spain – Brazil throughput behavior with multiple threads comparison.

It can be seen that the system throughput speeds up almost linearly from one thread to 6 threads reaching the saturation point of 54 Kbytes/sec. For an amount of threads bigger then six there is no real gain in throughput. It is also important to notice that the number of disconnections appears to have a random behavior.

Table A-2: Average throughput, disconnections and relative throughput for LDS communication.

	Brazil-Spain		
	OveralThroughput (Kbytes/sec)	Disconnections	Relative Throughput
1Thread	8,91	2	1,00
2Threads	19,51	4	2,19
4Threads	40,49	4	4,55
6Threads	53,95	8	6,06
8Threads	55,52	3	6,23
16Threads	55,01	3	6,18
32Threads	50,11	5	5,62

A.4 Conclusions

This chapter presented the inter-cluster communication problems: unpredictable and unstable latency and throughput and the unreliable transport layer connection.

Although there are MPI solutions that work over wide-area networks, like Internet, all of them are based on the transport layer protocol TCP that proved not to be reliable when concerning long-distance and long-duration communication.

These problems, the fact that multiple connections improves throughput when long-distance standard Internet is used, and the need of a simple and transparent interface for communication, lead to the development of a library called Long Distance Service (LDS).

The LDS layered model and internal structure was presented in details and experiments were made to prove its efficiency. Experiments show that LDS not just allows, through multithreading, the improvement on the throughput that reached 6 times the single socket ones, but also shows that the reliability is accomplished.

It is important to remark that the actual stage of the library just allows the connection of two clusters. Future works would be to generalize the approach to more than two clusters and to prepare the library as an extension of MPI so that communication inter-clusters can be even more transparent through standard MPI calls. Other future work would be to put the library to dynamically increase and decrease the amount of threads to keep the network saturated along the communication without the waste of system resources.