

Tesis defendida por
Anuar Lezama Barquet
y aprobada por el siguiente comité

Dr. Andrey Chernykh
Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez
Miembro del Comité

Dr. Raúl Ramírez Velarde
Miembro del Comité

Dr. David Hilario Covarrubias Rosales
Miembro del Comité

Dr. José Antonio García Macías
Coordinador del programa de posgrado
en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales
Director de la Dirección de Estudios de
Posgrado

07 de septiembre de 2012

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR
DE ENSENADA



Programa de Posgrado en Ciencias
en Maestría en Ciencias de la Computación

Experimental Analysis of a HPC model for Infrastructure as a Service Cloud

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de
Maestro Ciencias

Presenta:

Anuar Lezama Barquet

Ensenada, Baja California, México
2012

Resumen de la tesis de Anuar Lezama Barquet, presentada como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación. Ensenada, Baja California. 2012.

Experimental Analysis of a HPC model for Infrastructure as a Service Cloud

Resumen aprobado por:

Dr. Andrey Chernykh
Director de Tesis

En esta tesis abordamos el problema de calendarización en un modelo conocido como “infraestructura como servicio” (IaaS) en nube computacional. El modelo utilizado permite la calendarización de trabajos secuenciales con diferentes niveles de servicio para el usuario. Estos niveles de servicio se distinguen en el tiempo de respuesta máximo que el trabajo admitido recibe. En este modelo, cada nivel de servicio consta de un factor de laxitud y un precio por unidad de tiempo de ejecución. En caso de que un trabajo sea aceptado, el sistema se compromete a completarlo antes de su fecha límite. La fecha límite de un trabajo es calculada mediante el tiempo de procesamiento del trabajo por el factor de laxitud del nivel de servicio. Una vez que el trabajo es recibido, el proveedor debe de manera inmediata e irrevocable aceptar o rechazar la ejecución del trabajo. Evaluamos experimentalmente mediante simulaciones el desempeño de cuatro algoritmos que combinan distintos modelos de sistemas (único y paralelo) y número de niveles de servicio ofrecidos (único y múltiple), con una carga de trabajo realística. Tomando un conjunto de métricas determinamos los mejores parámetros para los algoritmos evaluados. Nuestros experimentos y análisis mostraron que los algoritmos para sistemas paralelos poseen un mejor desempeño y que existe una discrepancia entre los mejores parámetros para el grupo de métricas centrado en el sistema y centrado en el algoritmo con respecto al grupo centrado en el usuario.

Palabras Clave: Infraestructura como Servicio, Computación en Nube, Calendarización, Niveles de Servicio

Abstract of the thesis presented by Anuar Lezama Barquet as a partial requirement to obtain the Master of Science degree in Computer Sciences. Ensenada, Baja California, México 2012.

Experimental Analysis of a HPC model for Infrastructure as a Service Cloud

Abstract approved by:

Dr. Andrey Chernykh
Thesis Director

In this thesis, we address a scheduling problem for an infrastructure as a service (IaaS) type of cloud. In this problem, a user executes sequential jobs with different service levels. These levels are distinguished by the maximum response time that is guaranteed for accepted jobs, and a price per unit of job execution. If the system accepts a job, it provides quality of service (QoS) in the form of a commitment to complete its execution before a certain deadline. This deadline is calculated using the processing time of the job and the slack factor of the service level. Once the job is submitted, the system must decide immediately and irrevocably if it accepts or rejects the job for execution. We perform an experimental evaluation through simulations of four algorithms that combine different system models (single and parallel), and number of service levels (single and multiple), with a realistic workload. Using a series of metrics we obtain the best parameters for the evaluated algorithms. Our results show that the algorithms for parallel systems have a better performance, and that there exist differences between the best parameters for the system-centric and the algorithm-centric group of metrics compared with the user-centric group.

Keywords: Infrastructure as a Service, Cloud Computing, Scheduling, Service Levels

Dedication

This thesis is dedicated to all the people that make
research in Mexico possible.

Acknowledgements

This thesis would not have been possible without the guidance of my advisor Professor Andrey Chernykh. His advice, support and encouragement in the last two years have allowed me to discover the area of parallel computing and to learn the methodology to perform this research work.

My gratitude to the members of my thesis committee: Professor Carlos Brizuela Rodriguez, Professor David H. Covarrubias Rosales and Professor Raúl Ramírez Velarde for their time, comments and interest in assisting me in this research work. I am also thankful to all the people that have helped me with corrections for this thesis.

I would like to thank to CICESE, the hosting institution of my studies and CONACYT for the scholarship with record number 242921 provided.

I am grateful to all my professors and peers at CICESE, especially to the research team of parallel computing and vision for sharing the laboratory and meal time with me. I thank to all the friends I have made here in Ensenada and the friends I have back in Mexico City, who have helped me in many ways in the last years.

Lastly, I would like to thank my family and my girlfriend Ana Angelica Hernandez for their support in my decision to pursue my master studies here at Ensenada.

Table of contents

	Page
Resumen	1
Abstract.....	2
Dedication.....	3
Acknowledgements.....	4
Table of contents	5
List of figures	7
List of tables.....	10
1. Introduction	11
1.1 Cloud computing.....	11
1.1.1 Cloud architecture	11
1.1.2 Cloud computing characteristics	12
1.1.3 Service models.....	13
1.1.4 Deployment models	14
1.1.5 Service level agreements	14
1.2 Problem statement.....	15
1.3 Justification	16
1.4 Objectives	18
1.4.1 General objective	18
1.4.2 Specific objectives.....	18
1.4.3 Research questions	19
1.4.4 Thesis organization	19
2. Background.....	21
2.1 Related work.....	21
2.2 Theoretical framework	24
3. Model Description	29
3.1 IaaS model.....	29
3.2 Formal definition	33
3.3 Scheduling algorithms.....	38
3.4 Experimental setup	41
3.4.1 Single Service Level Single Machine (SSL-SM) algorithm.....	41
3.4.2 Single Service Level – Parallel Machines (SSL-PM) algorithm	43
3.4.3 Multiple Service Level - Single Machine (MSL-SM) algorithm.....	43
3.4.4 Multiple Service Level - Parallel Machines (MSL-PM) algorithm	45
3.5 Criteria analysis	45
3.6 Workload.....	46

4. Experimental Competitive Factor Analysis.....	49
4.1 Price function	49
4.2 Optimal income bounds	52
4.3 Competitive factor	56
5. Service provider benefits	62
6. Experiments Results	65
6.1 Single Service Level – Single Machine (SSL-SM) algorithm.....	65
6.2 Single Service Level -Parallel Machines (SSL-PM) algorithm	71
6.3 Multiple Service Levels-Single Machines (MSL- SM) algorithm	79
6.4 Multiple Service Levels - Parallel Machines (MSL-PM) algorithm	85
7. Criteria Analysis	92
7.1 Group of metrics analysis	92
7.2 Criteria analysis results	93
Conclusions	99
Bibliographic references	103
Appendix.....	106
A.1 Degradations metrics plots	106

List of figures

<i>Figure</i>		Page
1	Competitive factor bound for the SSL-SM scheduling problem.	25
2	Minimum competitive factor bound for the SSL-SM and SSL-PM algorithms.	26
3	Minimum competitive factor bound for the MSL-SM and MSL-PM algorithms.	28
4	Cloud middleware diagram.	31
5	Competitive factor for the SSL-SM algorithm with a sf from 1 to 1000.	42
6	Number of jobs in the experiment interval.	47
7	Processing time of the jobs in the workload.	48
8	Cost and prices for the SSL-SM algorithm.	52
9	Upper bound of the unlimited resources optimal income for the SSL and MSL algorithms.	53
10	Upper bound of the limited resources optimal income for the SSL algorithms.	54
11	Upper bound of the limited resources optimal income for the MSL algorithms.	55
12	Saturation matrix for the competitive factor limits.	56
13	Competitive factor for the SSL algorithms.	58
14	Competitive factor for the MSL algorithms.	60
15	Service provider benefits for the SSL-PM algorithm.	62
16	Service provider benefits for the MSL-PM algorithm.	63
17	Relative service provider benefits for the MSL-PM algorithm.	64
18	Total processing time for SSL-SM algorithm.	66
19	Machines efficiency for SSL-PM algorithm.	67

List of figures (continued)

<i>Figure</i>		Page
20	Mean waiting time for SSL-SM algorithm.	68
21	Mean bounded slowdown for SSL-SM algorithm.	69
22	Machine efficiency for SSL-SM algorithm.	69
23	Mean number of interruptions per job for SSL-SM algorithm.	70
24	Total processing time for SSL-PM algorithm.	72
25	Relative processing time for SSL-PM algorithm.	72
26	Percentage of rejected jobs for SSL-PM algorithm.	74
27	Mean waiting time for SSL-PM algorithm.	75
28	Mean bounded slowdown for SSL-PM algorithm.	76
29	Machines efficiency for SSL-PM algorithm.	77
30	Mean number of interruptions per job for SSL-PM algorithm.	78
31	Total processing time for MSL-SM algorithm.	80
32	Incoming SLA contribution to TPT for MSL-SM algorithm.	81
33	Percentage of rejected jobs for MSL-SM algorithm.	82
34	Mean waiting time for MSL-SM algorithm.	82
35	Mean bounded slowdown for MSL-SM algorithm.	83
36	Machine efficiency for MSL-SM algorithm.	84
37	Mean number of interruptions per job for MSL-SM algorithm.	85
38	Total processing time for MSL-PM algorithm.	87
39	Percentage of rejected jobs for MSL-PM algorithm.	87
40	Mean waiting time for MSL-PM algorithm.	88
41	Mean bounded slowdown for MSL-PM algorithm.	89
42	Machines efficiency for MSL-PM algorithm.	89
43	Mean number of interruption per job for MSL-PM algorithm.	90

List of figures (continued)

<i>Figure</i>		Page
44	System-centric metric group for the SSL-PM algorithm.	94
45	User-centric metric group for the SSL-PM algorithm.	95
46	Algorithm-centric metric group for the SSL-PM algorithm.	95
47	System-centric metric group for the MSL-PM algorithm.	96
48	User-centric metric group for the MSL-PM algorithm.	97
49	Algorithm-centric metric group for the MSL-PM algorithm.	98
50	Mean bounded slowdown degradations for SSL-PM algorithm.	106
51	Percentage of rejected jobs degradations for SSL-PM algorithm.	107
52	Service provider benefits degradations for SSL-PM algorithm.	107
53	Competitive factor degradations for SSL-PM algorithm.	108
54	Mean waiting time degradation for SSL-PM algorithm.	108
55	Mean bounded slowdown degradations for MSL-PM algorithm.	109
56	Percentage of rejected jobs degradations for MSL-PM algorithm.	109
57	Competitive factor degradations for MSL-PM algorithm.	110
58	Service provider benefits degradations for MSL-PM algorithm.	110
59	Mean waiting time degradation for MSL-PM algorithm.	111

List of tables

Table		Page
1	Implemented metrics.	36
2	Slack factor distribution over the number of SLAs available.	44
3	Group metrics for the criteria analysis.	93

Chapter 1

Introduction

In this chapter, we present important concepts of cloud computing related to our research. Additionally, we introduce the definition of Service Levels as a tool for establishing characteristics of the provided service and quality of service received by end users.

1.1 Cloud computing

Cloud computing is a new paradigm related to how computing services are offered. It is defined by the US National Institute of Standards and Technology (Mell et al., 2011) as a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal effort of service provider interaction. The location of physical resources and devices being accessed are typically unknown to the end user. It also provides facilities for users to develop, deploy and manage their applications “on the cloud”, which involves virtualization of resources that maintains and manages themselves.

1.1.1 Cloud architecture

Cloud computing architecture is categorized into two main sections: front end and back end. Front end can be end user or client or any other application (i.e. web browser, etc.) which uses cloud services. Back end is the network of servers with any computer program and data storage system. It is usually assumed that cloud contains infinite storage capacity for any software available in market. Cloud has different applications that are hosted on their own dedicated server farms.

Cloud has a centralized server administration system. A centralized server administers the system, balances client supply, adjusts demands, monitors traffic and avoids congestion. This server follows protocols, commonly known as middleware.

Cloud architecture is based on a very important assumption, which is mostly true. This assumption is that the demand for resources is not always consistent from client to cloud. For this reason the servers of cloud are unable to run at their full capacity. To avoid this scenario, server virtualization techniques are applied. In server virtualization, all physical servers are virtualized and they run multiple servers with either same or different application. As one physical server acts as multiple physical servers it curtails the need for more physical machines.

1.1.2 Cloud computing characteristics

Cloud computing typically entails the following characteristics: (i) Agility, the cloud works in the distributed mode environment. It shares resources among users and tasks, while improving efficiency and agility (responsiveness). (ii) High availability and reliability, availability of servers is high and more reliable as the chances of infrastructure failure are minimal. (iii) Multi-sharing, with the cloud working in a distributed and shared mode, multiple users and applications can work more efficiently with cost reductions by sharing common infrastructure. (iv) Services in pay-per-use mode, service level agreements (SLAs) between the provider and the users must be defined when online offering services in pay per use mode. This may be based on the complexity of services offered. (v) Application Programming Interfaces (APIs) may be offered to the users so they can access services on the cloud by using these APIs. (vi) Rapid elasticity, capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the user, these capabilities for

provisioning appear to be unlimited and can be appropriated in any quantity at any time.

1.1.3 Service models

Cloud computing is divided in three main service models that are also referred as layers.

Software as a Service (SaaS). The service offered to the user is the use of provider's applications running on a demand infrastructure. The applications are accessible from various client devices through different interfaces such a web browser or a program interface. The user does not manage or control the underlying cloud infrastructure and he is limited to only modify the user application configuration settings.

Platform as a Service (PaaS). In this model the user has the capability to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming language, libraries, services, and tools supported by the provider. The user does not manage or control the underlying cloud infrastructure but has control over the deployed applications and possibly configuration settings for the application –hosting environment.

Infrastructure as a Service (IaaS). This model gives the capability to provision processing, storage, networks, and other fundamental computing resources, where the customer is able to deploy and run arbitrary software, which can include operating systems and applications. The user does not manage or control the underlying cloud infrastructure but has control over the software stack, storage and processing capacity and limited control over selected networking components.

1.1.4 Deployment models

Cloud computing has three deployment models which differ on the access that is available to the cloud. The first type is the public cloud. In this deployment model the provider owns and manages the cloud, users grant access to the resources through subscription. Once the subscription is established, a user initializes consumption of computing and storage resources through service catalog self-service portal. These resources are highly scalable and automatically provisioned. The resources are metered and billed on a pay per use basis.

Private cloud is the second type of development model. Here the resources of the cloud are client-dedicated and its access is defined by the client. Data governance rules and regulations are established to the user making this type of cloud more secure. A private cloud is designed to offer the same features and benefits of cloud systems, but removes a number of objections to the cloud computing model including control over data, security and regulatory compliance.

Finally, the third deployment model is a hybrid cloud. In this model enterprise computing and private clouds are extended outward to consume public compute resource for peak needs.

1.1.5 Service level agreements

SLAs are binding contracts between a service provider and the user of that service. The SLA should contain the list of services that the provider will deliver and a complete definition of each service, metrics to determine whether the provider is delivering the service as promised, an auditing mechanism to monitor the service, responsibilities of the provider and the user and remedies available to

both if the terms of the SLA are not met, and the description of how the SLA will change over time.

The responsibilities of the provider are in the form of the level of quality of service (QoS) that the service should keep. On the other hand the responsibilities of the user are in the form of payments for the service received. Economic penalties or other form of compensation are commonly used in case of non-fulfillment on the terms of the SLA, such as QoS levels.

There are two types of SLAs (IBM, 2010). The first type is off-the-shelf agreements which are offered by public clouds and often are non-negotiable which may not be acceptable for those with mission-critical apps or data. The second type are customized SLAs which terms are negotiated between the provider and the user until both agree.

Negotiable SLAs require a mechanism of negotiation and common markets. There has been research on negotiation mechanism, market places, protocols and definition of SLAs parameters that allows this negotiation to take place in an automatic and transparent way for the user.

1.2 Problem statement

Infrastructure as a service is becoming attractive in many commercial and research areas with the emergence of providers and tools surrounding on-demand deployment of virtual machines. However, for the commercial success of this computing model, the cloud providers need to provide better and strict QoS guarantees. These QoS guarantees are established with the service definition as maximum and minimum metric values in the form of SLAs.

Currently, IaaS provides mechanisms to deploy and manage virtual machines. Although there are several cost-metrics involved in calculating the running costs for a virtual machine, the cost is dominated by instance uptime and not by the CPU usage. The more resources an instance has, the higher the cost associated with keeping it up and running. A cost effective utilization of this IaaS model by the user requires ensuring that the instance is only running where is needed. Even more, the user needs to consider the costs and effort of booting the virtual machine and installing the necessary software before being able to start executing jobs. Another factor to consider is that in the cloud, the costs are charged on timing units that make it economically unacceptable to run jobs of short length and high parallelization.

Such restrictions serve as a preamble to design and propose IaaS models that help to extend the adoption of clouds for the execution of HPC jobs through the incorporation of a QoS guarantee scheme and a flexible economic model.

1.3 Justification

The use of SLAs is a fundamentally new approach for job scheduling. With this approach, schedulers are based on satisfying QoS constraints. The main idea is to provide different levels of service, each addressing different set of customers for the same services, in the same SLA, and establish bilateral agreements between a service provider and a service consumer to guarantee job delivery time depending on the service level (SL) selected.

The shifting emphasis of the grid and clouds towards a service-oriented paradigm led to the adoption of SLAs as a very important concept, but at the same time led to the problem of finding the most stringent SLAs.

There has been significant amount of research on various topics related to SLAs: admission control techniques (Wu et al., 2011); incorporation of the SLA into the Grid/Cloud architecture (Patel et al., 2009); specifications of the SLAs (Andrieux et al., 2004)(IBM, 2010); usage of SLAs for resource management; SLA-based scheduling (Wu et al., 2011), SLA profits; automatic negotiation protocols (Cosmin Silaghi et al., 2010); economic aspects associated with the usage of SLAs for service provision (Macías et al., 2010), etc. Little is known about the worst case efficiency of SLA scheduling solutions. There are only very few theoretical results on SLA scheduling, and most of them address real time scheduling with given deadlines.

A more complete study is presented in (Schwiegelshohn et al., 2012). Authors theoretically analyze the single (SM) and the parallel machine (PM) models subject to jobs with single (SSL) and multiple service levels (MSL). Their analysis is based on the competitive factor, which is measured as the ratio between the income of the infrastructure provider obtained by the scheduling algorithm and the optimal income. They provide worst case performance bounds of four greedy acceptance algorithms named SSL-SM, SSL-PM, MSL-SM, MSL-PM, and two restricted acceptance algorithms MSL-SM-R, and MSL-PM-R. All of them are based on the adaptation of the preemptive EDD (Earliest Due Date) algorithm for scheduling the jobs with deadlines.

In this thesis, we adopt the models of IaaS cloud model proposed by Schwiegelshohn et al. (2012). To show the practicability and competitiveness of the algorithms, we conduct a comprehensive study of their performance and derivatives using simulation. We take into account an important issue that is critical for practical adoption of the scheduling algorithms, the use of workloads based on real production traces of heterogeneous HPC systems.

We also propose a price function that helps us to calculate the performance of the algorithms. The price function considers costs that a system has with a full-acceptance algorithm, and where its income is only related with total processed time. The price function is related with the process of assigning SLAs to jobs. Therefore, a more complex price function requires a more complex assignment of jobs to SLAs. We define the competitive factor lower bound. This definition can be used to obtain the optimal income of SLA based algorithms according to the workload's characteristics

1.4 Objectives

1.4.1 General objective

Implementation and experimental analysis of the SSL-SM, SSL-PM, MSL-SM, and MSL-PM scheduling algorithms for IaaS type of clouds with QoS determined by SLAs.

1.4.2 Specific objectives

1. Formal definition of the cloud scheduling problem
2. Design and implementation of a simulation platform for the IaaS model
3. Design and implementation of the proposed algorithms
4. Evaluation of cloud performance using different algorithms and system parameters
5. Propose and use a price function for the model

1.4.3 Research questions

In order to achieve the proposed objectives, we intent to answer the following research questions:

1. How can scheduling algorithms guarantee performance when SLA constraints are considered?
2. How does the SLA model change the system performance?
3. Which performance measure is the most appropriate for analyzing scheduling algorithms with SLA constraints?
4. Which SLA model performs better in the studied systems?
5. How job rejection changes the system's competitiveness?
6. What kind of performance guarantees can be assured when the scheduler is based on SLA models?

1.4.4 Thesis organization

This thesis is organized in the following way. Chapter 1 shows an introduction to cloud computing, the problem statement and the objectives of this thesis. In Chapter 2, we mention a series of relevant works related with the topic of clouds and parallel systems scheduling with SLAs. In this chapter, we also present the theoretical work that supports the IaaS model used in this thesis. In Chapter 3, we make a formal description of the IaaS cloud model together with the scheduling algorithms used in our experimental study. Additionally, we also present a description of the metrics implemented, the experimental setup, and a brief description of the workload. In Chapter 4, we present the analysis of the competitive factor, the price function employed, and a comparison of the experimental results of this particular metric with the theorems available. In Chapter 5, we present the analysis of the service provider benefits. In Chapter 6, we make a description and analysis of the rest of the metrics experimentally

obtained. In Chapter 7, we present a criteria based analysis that gathers different metrics into groups to provide guidance in the selection of the best parameters for the studied algorithms. Finally, we present the conclusions of this work.

Chapter 2

Background

In this chapter, we review related works in the area of cloud, grid and parallel machines scheduling with SLAs. Here, we also present the theoretical foundation of the IaaS model used in this thesis.

2.1 Related work

Traditional IaaS models do not require job scheduling. They provide resources to a user who has to manage these resources by himself to execute jobs. The user is then responsible for the resource management to achieve his own processing necessities and QoS requirements.

There exist research models of IaaS clouds such the one proposed by Garg et al. (2011) that schedule HPC jobs co-allocating them with transactional applications such as Web services. This method allows the system to take advantage of the different requirements of processing from different typical applications executed on the cloud. The algorithm co-allocates transactional jobs with HPC jobs so the HPC jobs can use the underutilized resources from the execution of transactional jobs. Additionally, for peak utilization intervals from the transactional jobs, the HPC jobs can be delayed to provide more resources. The algorithm requires for the acceptance of HPC jobs the availability of underutilized machines. The users of HPC jobs receive a guarantee in the form of maximum response time, which could not be complied by the provider.

The idea of offering QoS in the form of maximum response time was previously presented for parallel machines by Islam et al., (2003). This model is the first one

that implements admission control and commitment in deadlines for admitted parallel jobs. This study shows the feasibility of the model and compares its trade-offs with respect to existing non-deadline based schemes. The proposed algorithm schedules uniprocessor tasks with hard real time deadlines on multiprocessor systems. They use Earliest Deadline First and Least Laxity First as heuristics to select jobs that extend the current feasible partial schedule.

In order to associate deadlines with jobs, they first use EASY back-fill algorithm to generate a valid schedule. Based on the completion time of the schedule generated by EASY back-fill and a stringency factor, they assign a deadline to every job. The stringency factor determines how tight the deadline has to be set compared with the EASY back-fill schedule. With a stringency factor of zero, the deadlines are set to completion times of the jobs according to the EASY back-fill schedule. They use the stringency factor to test their scheme under a variety of scenarios with loose and stringent deadlines.

The use of SLAs has been proposed as a tool for scheduling jobs in multi-user systems such as Grids. In the work of Sakellariou et al., (2008), the authors define a model of SLA that includes the following terms: (i) Earliest job start time, (ii) Latest job finish time, (iii) Reserved time for job execution, (iv) Number of CPU nodes required, and (v) Final price agreed. These terms are used to calculate the SLA priority which is then used to schedule the jobs. The work consists on an experimental comparison between different heuristics to prioritize the SLAs.

While the idea of using QoS guarantees in the form of deadlines and SLAs as a tool for scheduling jobs is not new, the models previously presented do not consider SLs for determining the deadlines guarantees within the SLAs. Additionally, they may accept delays in the deadlines with penalties. The scheduling models available for clouds do not consider that the infrastructure can be used for just executing HPC jobs without requiring a model with independent VMs.

Regarding theoretical results of SLA scheduling little is known about the worst case efficiency of these solutions. There are only very few theoretical results on SLA scheduling, and most of them address real time scheduling with given deadlines.

Baruah and Haritsa (1997) discuss the online scheduling of sequential independent jobs on real time systems. They presented the ROBUST (Resistance to Overload By Using Slack Time) algorithm that guarantees a minimum slack factor for every task. The slack factor f of a task is defined as a ratio of its relative deadline over its execution time requirement. It is a quantitative indicator of the tightness of the task deadline. The algorithm provides an Effective Processor Utilization (EPU) of $(f - 1)/f$ during the overload interval.

Baruah et al. proposed a new solution to the problem using multiple processors and a new on-line scheduling algorithm. He shows that given enough processors, on-line scheduling algorithms can be designed with performance guarantees arbitrarily close to that of optimal clairvoyant uniprocessor scheduling algorithms.

A more complete study is presented in (Schwiegelshohn et al., 2012). Authors theoretically analyze the single (SM) and the parallel machine (PM) models subject to jobs with single (SSL) and multiple service levels (MSL). Their analysis is based on the competitive factor, which is measured as the ratio between the income of the infrastructure provider obtained by the scheduling algorithm and the optimal income. They provide worst case performance bounds of four greedy acceptance algorithms named SSL-SM, SSL-PM, MSL-SM, MSL-PM, and two restricted acceptance algorithms MSL-SM-R, and MSL-PM-R. All of them are based on the adaptation of the preemptive-EDD (Earliest Due Date) algorithm for scheduling the jobs with deadlines.

They prove a series of theorems and corollaries that show the complexity of the problem and also the response in terms of the competitive factor of the greedy acceptance algorithm. This model is the one employed in this thesis.

2.2 Theoretical framework

In this section, we present theorems and corollaries related with the algorithms experimentally studied in this thesis. A complete demonstration of the theorems and corollaries can be found in (Schwiegelshohn *et al.* 2012).

Theorem 1: $c_V \leq 1 - \left(1 - \frac{p_{min}}{p_{max}}\right) \cdot \frac{1}{f_I}$ holds for SSL-SM with service level S_I . Conditions the minimal $p_{min} > 0$ and p_{max} is the maximal possible processing time of any submitted job. Rational slack factors are allowed but with restrictions in its granularity such that either $f_I = \lfloor f_I \rfloor$ or $f_I - \lfloor f_I \rfloor \gg \frac{p_{min}}{p_{max}}$ holds.

Theorem 1 provides a maximum competitive factor value (c_V) for the problem of scheduling jobs with a single slack factor on a single machine system (SSL-SM). The plot in Figure 1 shows the c_V bound. To build this graph, we considered that the processing times of the jobs are normalized with respect to p_{min} , thus p_{min} has a value of 1. Then, we consider p_{max} in the interval of 1 to 10 and f_I from 1 to 20. In the plot we see how as the values of $f_I \rightarrow \infty$ or the value of $p_{max} \rightarrow 0$, the $c_V \rightarrow 1$.

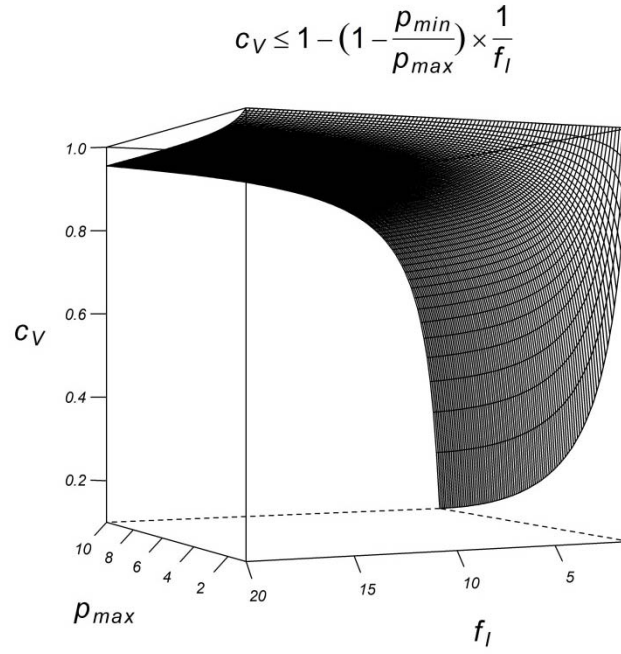


Figure 1. Competitive factor bound for the SSL-SM scheduling problem.

Theorem 2: Greedy acceptance has the competitive factor greater or equal to $1 - \frac{1}{f_I}$ for SSL-SM with service level S_I .

Theorem 2 provides the minimum value of c_V that we can expect that the greedy SSL-SM scheduling algorithm to obtain. In Figure 2, the plot of this bound is shown. We see that the c_V bound increases as the f_I increases.

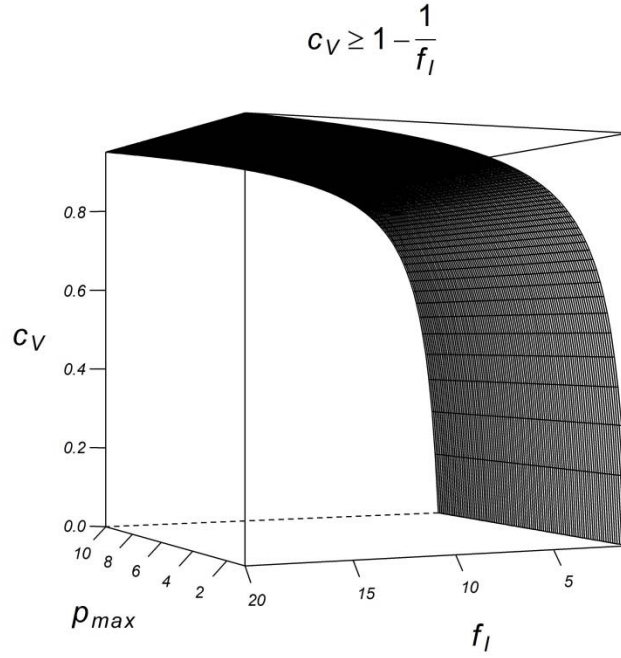


Figure 2. Minimum competitive factor bound for the SSL-SM and SSL-PM algorithms.

In the case of the SSL-PM algorithm, Theorem 3 claims that the bound obtained for the SSL-SM algorithm also holds when the infrastructure is extended with any number of identical machines as long there is still only a single service level.

Theorem 3: $c_V \leq 1 - \frac{1}{f_I}$ holds for SSL-PM with service level S_I if the ratio between the largest and the smallest processing time of a job can be arbitrary large.

If the processing times of the jobs are restricted, Corollary 4 states that a slightly upper bound for the SSL- PM algorithm can be obtained.

Corollary 4: $c_V \leq \frac{f_I}{1+f_I \cdot \left(1 - \frac{p_{min}}{p_{max}}\right)}$ holds for SSL-PM with service level S_I and its integer slack factor $f_I < \frac{p_{max}}{p_{min}}$.

Theorem 5: Greedy acceptance has a competitive factor $1 - \frac{1}{f_I}$ for SSL-PM with service level S_I .

Theorem 5 shows that the minimum value of c_V obtained for the SM holds for the case with PM.

The MSL-SM algorithm analysis only refers to two different service levels S_I and S_{II} . Requiring that $1 < f_I < f_{II} < \infty$ and $u_I > u_{II}$ for the slack factors and the price values, respectively. In general, Theorem 1 can be used to determine an upper bound for the competitive factor. Where the smallest of these upper bounds is an upper bound for the competitive factor for MSL-SM. A stronger upper bound can be determined by Corollary 6 if $f_I - \lfloor f_I \rfloor \gg \frac{p_{min}}{p_{max}}$ holds for the slack factor f_I of service level S_I and $f_{II} < 2$ for two services levels S_I and S_{II} .

Corollary 6: $c_V \leq \max \left\{ \frac{\frac{p_{min}}{p_{max}}}{(f_I - 1)}, \frac{f_I - 1 + \frac{p_{min}}{p_{max}}}{f_I - 1 + \frac{u_I}{u_{II}}} \right\}$ holds for MSL-SM.

The plot of this bound is difficult to build because it is a four dimension function and, additionally, we can always refer to the plot of Figure 4 for a bound of general values of this algorithm's slack factor.

The performance of the greedy acceptance for multiple service levels has a bound defined by Corollary 7.

Corollary 7: Greedy acceptance has the competitive factor $\frac{u_{II}}{u_I} \cdot (1 - \frac{1}{f_I})$ for MSL-SM with service level S_I and S_{II} .

Figure 3 shows the bound defined by Corollary 7. We plot the relation $u_I > u_{II}$ and give it a value in the interval $(0,1]$, We see that the competitive factor reaches a

value of one when the ratio $\frac{u_{II}}{u_I} = 1$ and $f_I = 1$. If we keep $\frac{u_{II}}{u_I} = 1$, when the value of f_I grows the competitive factor linearly decrease towards zero. If instead, we keep $f_I = 1$, we see that as the ratio $\frac{u_{II}}{u_I}$ tends to zero, the competitive factor also tends to zero.

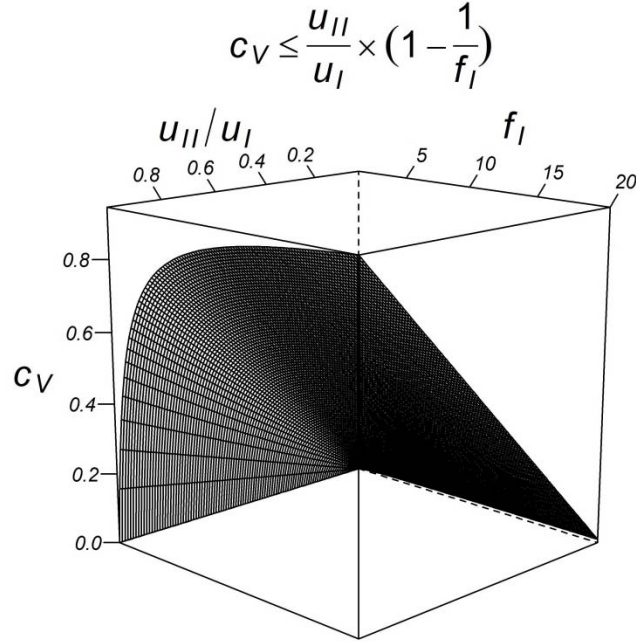


Figure 3. Minimum competitive factor bound for the *MSL-SM* and *MSL-PM* algorithms.

Corollary 11: Greedy acceptance has a competitive factor $\frac{u_{II}}{u_I} (1 - \frac{1}{f_I})$ for *MSL-PM* with service levels S_I and S_{II} .

Corollary 11 claims that the bound for the *MSL-SM* is also valid for the *MSL-PM* algorithm. This bound is shown in Figure 6.

In the next chapter we present the formal description of the model employed, the metrics implemented and the workload used.

Chapter 3

Model Description

In this chapter, we describe the general Infrastructure as a Service model, some of its characteristics, and the difference between this general model and the one we use in our research. Once that point is covered, we define mathematical notations and metrics implemented for algorithm's evaluation, followed by the experimental setup, and workload analysis.

3.1 IaaS model

Infrastructure as a Service (IaaS) is the delivery of virtual hardware (server, storage and network), and associated software (operating systems, file system, etc.) as a service. It is considered an evolution of traditional hosting data that does not require any long term commitment and allow users provision of resources on demand. It has an advantage to allow minimizing or even eliminate associated capacity cost and letting customers to add or remove capacity from their IT infrastructure to meet peak or oscillating service demands while paying only for the capacity used. IaaS benefits its customers by transferring the responsibility for housing, running and maintaining the infrastructure to the provider.

The IaaS provider supply administrative services and the platform for storing and executing applications. The provider allows scaling resources such as processing power, bandwidth, memory and storage. These resources can be purchased with a contract or on a pay-as-you-go basis.

Some characteristics and components of IaaS include: an utility computing service and billing model, the automation of administrative tasks, dynamic scaling, multi-tenancy, policy-based services, and internet connectivity.

The IaaS provides an environment for running user built virtualized systems in the form of virtual machines (VMs) in the cloud provider's data center. A user starts with building a VM in an IaaS environment, and then he uploads, configures, and finally deploys it within this environment. One mechanism for deploying VMs into the IaaS infrastructure is through the use of SLAs.

SLAs describe the service parameters in terms of requirements and QoS levels that the provider is committed to keep. These parameters can then be used by the cloud middleware to allocate the user requests into the physical infrastructure.

According with Garg et al. (2011) current cloud datacenters host a variety of applications each with different SLA requirements. The transactional applications require response time and throughput guarantees, while non-interactive batch jobs concerns performance for example in the form of completion times.

A picture of the cloud middleware is shown in Figure 4. Cloud consumers (a) need flexible infrastructure on demand. The cloud consumers can be diverse such individual users, other clouds or PaaS applications. Every set of customers requires different resources and needs that can be established through different SLAs. The cloud management provides remote and secure interfaces for creating, controlling, and monitoring virtualized resources on an IaaS cloud. This layer should take the different requirement in the form of SLAs and manage the resources to full fit the QoS levels containing in them. This layer is also responsible to monitor the resources and reallocate them in case it is necessary. Virtual infrastructure (VI) management provides primitives to schedule and manage VMs across multiple hosts. Some toolkits currently do not use VI managers and, instead manage directly VMs themselves. Finally, the VM managers (d) allow multiple operating systems termed guests to run concurrently on a server infrastructure.

This allows the VM to share the hardware resources. They provide with simple primitives (start, stop and suspend) to manage VMs on a single host.

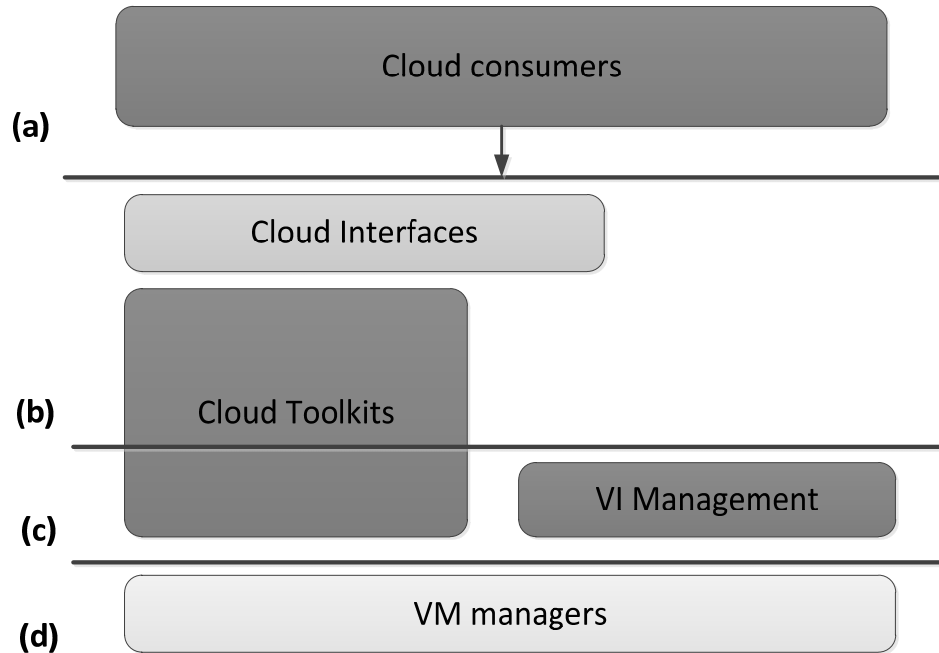


Figure 4. Cloud middleware diagram.

Using this technique VMs are loaded with all the software that will eventually run in the cloud. These include custom build software as well as licensed software. After the VM is built, it is uploaded to the IaaS vendor's hosting environment where it can be configured to use the IaaS vendor raw storage. Once configured, the VM can be deployed and started via VM manager or cloud manager which automatically finds available hardware to run the VM. Once the VM is started the IaaS vendor can ensure that the running VM continues to look healthy as a whole. The computers needed to run the application and the raw storage that is needed by the application are owned and supported by the IaaS vendor. It is responsibility of the user to monitor all the custom built software and licensed software to ensure that they are operating properly.

The cloud service customer and service provider (Bhardwaj et al., 2010) play key roles in a cloud environment. The cloud service customer needs a secure anytime anywhere access to low cost services that are flexible and easy to use. The biggest hurdle to adoption of cloud has to do with customer's discomfort in the following areas: security of services and the underlying data, service availability and reliability, service management to ensure SLAs, ensuring control over access policies, and the appropriate administration to facilitate flexible pricing structures. The service provider actually runs the service that the service customer wants and was designed and developed by the service creator. The cloud service creator needs tools and capabilities to offer differentiated services, offer incentives to ensure that customers keep coming back to use the services, and the ability to change services on-demand to stay competitive and address threats.

By using VMs in the cloud infrastructure jobs preemption can be supported, suspending the jobs of lower-priority and resuming it time after or potentially migrating it to other available nodes or even another cloud. We can do this without having to make the job inside the VM aware that it will go to be suspended, resumed or even migrated.

Virtualization is just a multi-tenancy strategy. Multi-tenant architecture allows a single instance of the hosted application be capable of servicing all customers (tenants). Not all clouds use virtualization. Examples are Google Apps Engine and Salesforce.com which have multi-tenancy models without virtualization.

Our IaaS model is intended to execute high performance computing (HPC) jobs. We propose a model that does not provide the service of VMs, but a mechanism that allows the computational job submitted by the customer to be executed on the cloud infrastructure with an application defined SLA capability. This kind of SLA should guarantee that the job will receive enough resources to achieve a certain QoS level in the form of maximum completion times.

This model differs from services such as Amazon EC2, a popular IaaS provider, which architecture allows its users to monitor and control their application as an instance but not as a service.

3.2 Formal definition

We consider a sequence of n_r jobs $\mathcal{J}_r = \{J_1, J_2, \dots, J_i, \dots, J_{n_r}\}$. Each job is submitted to a system with m machines with a SLA S_i from a set of SLAs \mathcal{S} offered by the provider. Each SLA represents a QoS SL guarantee.

Each service level (SL) S_i is characterized by a slack factor $sf_i \geq 1$ and a price per unit of execution u_i . The SLAs are ordered in \mathcal{S} by non-decreasing slack factor, i.e. $sf_i \leq sf_{i+1}$. We also assumed that $sf_i \sim \frac{1}{u_i}$ therefore $u_i \geq u_{i+1}$ is also met. A job j can be represented as a tuple $J_j = (r_j, p_j, S_j)$, where r_j and p_j are the released time and the processing time of the job j , respectively, and S_j is the SL chosen by the user for the execution of this job. Additional parameters of the job can be calculated by the system using these initial parameters. The deadline (d_j) is the latest time that the system has to complete the job J_j if it is accepted. This value is calculated by the equation (1).

$$d_j = r_j + sf_j \cdot p_j \quad (1)$$

This means that the slack factor ($sf_i \geq 1$) is the parameter that determines the deadline of a job. The profit that the system will obtain executing job J_j is calculated by the equation (2).

$$g_j = u_j \cdot p_j \quad (2)$$

The form of QoS that the system assures is that, once a job is accepted, it will complete its execution before its deadline is reached. The set of n jobs accepted by the system is $\mathcal{J} = \{J_1, J_2, \dots, J_i, \dots, J_n\}$ a subset of \mathcal{J}_r .

In order to evaluate the system performance we came with a series of metrics useful for systems scheduled through SLs, where traditional measures such as the maximum completion time (C_{\max}) become irrelevant.

For these systems the metrics must allow the provider to measure the performance of the system in terms of parameters that helps him to establish utility margins as well as user satisfaction for the service.

Due to the definition of the system model, the QoS guarantee in the form of response time of the accepted jobs will always been granted. However, other parameters for assessing user satisfaction can be considered, for example, the user reliability on the system. We can define this reliability as the capacity of the system to accept jobs for execution. Considering both the performance and the user satisfaction we employ the following series of metrics.

The first metric we use is the competitive factor (c_v) that measures the ratio in which the income generated by our algorithm gets closer to the value obtained by an optimal income V^* . The competitive factor (c_v) is defined by equation (3).

$$c_v = \frac{\sum_{j=1}^n u_i \cdot p_j}{V^*} \leq 1 \quad (3)$$

c_v requires the optimal income, which in most cases is not possible to calculate.

We consider two definitions of optimality in order to obtain a bound for V^* . The first one is when the system has an unlimited number of resources, therefore, it is always possible to execute all jobs released without restrictions of their deadline.

That means that the system is always able to provide a new machine for each incoming job, so it is capable to completely execute the workload. Multiplying this processing time with the maximum value in S (u_{\max}) we obtain the maximum income that the system can obtain. We define this bound as *unlimited* and its value is given by equation (4).

$$V^* \leq \hat{V}_u^* = u_{\max} \cdot \sum_{j=1}^{n_r} p_j \quad (4)$$

The second definition of the optimal solution is when the system has limited resources so there is a restriction given by the deadline on the number of jobs it can accept. We define this bound as *limited* and its value is given by equation (5).

$$V^* \leq \hat{V}_l^* = u_{\max} \cdot \min \left(\sum_{j=1}^{n_r} p_j, d_{\max} \cdot m \right) \quad (5)$$

The limited resource bound is obtained by computing the minimum value between the sum of the maximum deadline per the number of machines in the system and the processing jobs in the released jobs. Considering limited resources, no processing can be done above the minimum of these values; by multiplying this value by u_{\max} we obtain the maximum income value that the system is capable to obtain under these conditions.

With \hat{V}_u^* and \hat{V}_l^* we obtain upper bounds of the actual optimal income V^* and lower bound values of the competitive factors $c_V^u \leq c_V$ and $c_V^l \leq c_V$. Finally, we define two experimental bounds of the c_V one for the unlimited bound equation (6) and the other one for the limited bound equation (7).

$$c_V \geq c_V^u = \frac{\sum_{j=1}^n u_i \cdot p_j}{u_{\max} \cdot \sum_{j=1}^{n_r} p_j} \quad (6)$$

$$c_V \geq c_V^l = \frac{\sum_{j=1}^n u_i \cdot p_j}{u_{\max} \cdot \min \left(\sum_{j=1}^{n_r} p_j, d_{\max} \cdot m \right)} \quad (7)$$

Therefore the value of competitive factor obtained through our analysis is a minimum bound where the actual competitive factor could be any value over the bound presented.

If we consider that there is only one SL, we have that $u_1 = u_2 = \dots = u_i = u_{\max}$ then the equations (6) and (7) can be simplified.

Besides the competitive factor, we consider as well other metrics. These metrics are presented in Table 1.

Table 1. Implemented metrics.

	Metric	Description
1	Percentage of rejected jobs	$PRJ = \frac{n_r - n}{n_r} \cdot 100\% \leq 100\%$
2	Mean bounded slowdown	$MBSD = \frac{1}{n} \sum_{j=1}^n \frac{c_j - r_j}{\max(10, p_j)} \leq \max(f_i)$
3	Mean waiting time	$MWT = \frac{1}{n} \sum_{j=1}^n (c_j - p_j - r_j)$
4	Total processing time	$TPT = \sum_{j=1}^n p_j$
5	Mean number of interruption per job	$MNI = \sum_{j=1}^n \text{int}(J_j)$
6	Machine efficiency	$ME = \frac{\sum_{j=1}^n p_j}{m \cdot (c_n - r_1)} \leq 1$
7	Service provider benefits	$G = \sum_{i=1}^n (u_i \cdot p_i) - \alpha \sum_{j=1}^{n_r} p_j$

The percentage of rejected jobs (PRJ) defined by us measures our definition of reliability which is the ratio between jobs admitted by the system and jobs released. The interpretation of this metric is simple, when the PRJ is closer to zero the user has higher certainty that when he sends a job for execution it will be accepted. However, this definition of reliability cannot represent situations where the system is not able to accept the jobs because a disproportionate demand over limited resources. Therefore, considering the same workload over two systems with different capacities, the system with more resources always will have more reliability than one with fewer resources.

The mean bounded slowdown ($MBSD$) (D. Feitelson et al., 1997) represents the mean of the actual slack factor that jobs executed on the system had at the moment of being completed. As our admission test guarantees that $c_j \leq d_j$, the value of $MBSD$ will always be smaller or at most equal to the maximum slack factor of the set \mathcal{S} .

While the $MBSD$ measures the time that a job spends in the system as a ratio of the processing time, the mean waiting time (MWT) (D. Feitelson et al., 1997) measures just the time that the job spends in the system waiting for execution. The total processing time (TPT) is defined as the sum of the processing time of all accepted jobs by the system.

The mean number of interruptions (MNI) allows us to determine the overhead that our scheduling algorithm will produce on the system. Here $\text{int}(J_j)$ stands for the number of interruptions that job J_j had until its completion. Our scheduling algorithms assure that this value is always less than one, although we will always prefer values closer to zero.

The machine efficiency (ME) is defined by us as the ratio between the interval from the release time of the first job (r_1) to the completion of the last (c_n) where useful processing time was performed. A value of $ME = 1$ tell us that in the whole interval from the release of the first job to the completion of the last, jobs were executed without idle intervals between them.

We also use the service provider benefits (G) as an economic metric to measure the return that the provider will have to supply the service of executing computational jobs on his infrastructure. This metric depends on the sum of processing time of the set of accepted jobs \mathcal{J} , and the SLA of each job, as these parameters determine the price of the unit of execution u_i . Service levels with tighter slack factor are more expensive. The service provider benefits metric also considers that the income has to be used to operate the infrastructure. These expenditures can be diverse such as wages, electric bills, etc. and can be represented as a cost per unit of operation (α) multiplied by a fixed interval of time where the provider needs to keep the system running. Since this interval must be large enough to execute all jobs, we consider that its length is equal to the sum of processing times of the set of jobs released to the system (\mathcal{J}_r).

3.3 Scheduling algorithms

In this thesis, we analyze the performance of the following algorithms:

1. Single Service Level - Single Machine (SSL-SM)
2. Single Service Level – Parallel Machine (SSL- PM)
3. Multiple Service Level - Single Machine (MSL-SM)
4. Multiple Service Level – Parallel Machine (MSL- PM)

They contemplate different characteristics of the system, different SLs available to the user, and let us compare the efficiency of different IaaS cloud models.

These algorithms are based on the online scheduler algorithm proposed by Gupta et al., (2001). This algorithm is in turn based on the preemptive-EDF (Earliest Deadline First) scheduling algorithm. We first reproduce the details about the scheduling algorithm of Gupta and then explain the modifications for algorithms SSL-SM, SSL-PM, MSL-SM and MSL-PM.

Consider a scheduling algorithm A_m over m machines. When a job J arrives, A_m first runs an admission test on the machines M_1, M_2, \dots, M_m , in any predefined order, to check if all previously admitted jobs that have not yet completed plus job J , can be completed by their respective deadlines on some machine M_j . If so A_m admits J on machine M_j , otherwise it rejects J . Admitted jobs in each machine are executing by A_m in non-decreasing order of their deadline. Thus, preemptions may occur: a currently executing job will be preempted in favor of a newly admitted job with earliest deadlines.

The scheduler A_m maintains a queue Q_j that contains three pieces of information: (1) its job number, (2) its deadline, and (3) its remaining execution time (i.e. its execution time minus the processor time that it has consumed so far). The jobs in the queue are ordered by non-decreasing order of deadlines. Thus, if a machine M_j is busy, then it is executing the job at the head of Q_j (which has the earliest deadline) and then the remaining execution time of this job decreases as it continues to execute. The job is deleted from Q_j when its remaining execution time becomes zero, and the job (if any) that becomes the new head of the queue Q_j is executed next on M_j .

Consider the case of the scheduling algorithm A_m for the machine M_j . When a new job J arrives, if the job has not already been scheduled in one of the machines M_1, M_2, \dots, M_{j-1} (for $j > 1$) then A_m inserts J in its proper position in Q_j (such that the deadlines of jobs in Q_j are ordered in non-decreasing deadlines) with the same deadline as J then J is inserted after these jobs. Then A_m performs the following admission test to determine whether to admit or reject J on machine M_j .

```

reject = FALSE;
J' = J;
repeat
s = sum of the remaining execution times of all jobs in the queue
up to and including job J';
if s > d(J') then reject = TRUE
else J' = next job in the queue after J';
until (J' = NULL or reject = TRUE);

```

That is, *reject* is set to *TRUE* if admission of J will cause J or some job with a deadline later than J to miss its deadline (note that jobs with deadline earlier than J will not be delayed by J and hence need not to be checked in the admission test). If *reject* = *TRUE*, then J is deleted from Q_j and is rejected. Otherwise, J is admitted and retains its position in the queue Q_j . If the position of the queue is not the head of the queue, then A_m continues execution of the current job at the head of the queue Q_j on machine M_j . However, if J is positioned at the head of the queue Q_j , then the current job is preempted and J begins execution on M_j .

For defining our algorithms, we extend Gupta algorithm and consider a set \mathcal{S} of the SLs offered. Thus, before the job J arrives it is assigned with a SL from the set \mathcal{S} .

The SL represents a slack factor that is used to calculate the deadline of the job J . Once the assignment and calculus of the deadline is done, the algorithm proceeds like the Gupta algorithm. The differences between each algorithm comes from the number of machines m and the cardinality of the set \mathcal{S} . If $m = 1$ then the algorithm is denoted as Single Machine (SM) otherwise if $m > 1$ is denoted as Parallel Machine (PM). In case the cardinality of the set $|\mathcal{S}|=1$ then the algorithm is denoted as Single Service Level (SSL) otherwise if $|\mathcal{S}| \geq 1$ then the algorithm is denoted as Multiple Service Level (MSL).

3.4 Experimental setup

We propose a series of experiments to evaluate the response of the four algorithms. The experiments were performed using the grid scheduling simulator tGSF (teikoku Grid Scheduling Framework). tGSF is a standard trace based simulator that is used to study grid resources management problems. We modified and extended tGSF to include the capabilities necessary to simulate the IaaS cloud model, these include the greed admission control, the scheduling algorithm preemptive-EDD and the implementation of metrics. Design details of the simulator are described in (Hirales Carbajal et al., 2010). In the following sections we describe these experiments and the parameters we changed to evaluate the algorithms.

3.4.1 Single Service Level Single Machine (SSL-SM) algorithm

In this set of experiments we suppose that the system have only one processing unit or machine ($m = 1$) that receives all the jobs in the workload. These jobs are

assigned with the same SLA because $|\mathcal{S}|=1$ which means a single value of slack factor.

For evaluating the SSL-SM algorithm we consider the slack factor of the SLA as the parameter of adjustment to evaluate its performance. We first define an extended interval and then by evaluating the c_V we obtain a tighter region of study. For the extended interval we took discrete values of slack factor of 1, 2, 5, 10, 20, 50, 100, 200, 500, and 1000. Figure 5 shows the plot of the c_V^l varying the sf . The plot shows that there are two regions where the competitive factor has a greater value; the first one is where slack factor is relatively low and the second one where is very large tending to infinity.

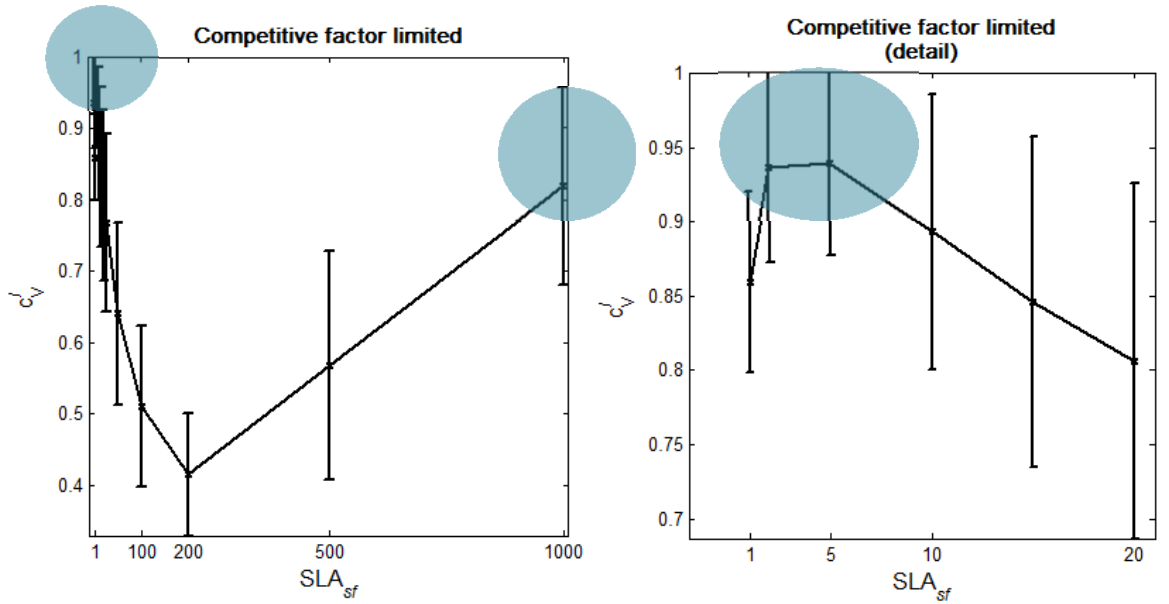


Figure 5. Competitive factor for the SSL-SM algorithm with a sf from 1 to 1000.

Taking into account that a $c_V = 1000$ means that in the worst case the user would need to wait one thousand times the processing time of his job, we consider that these larger slack factors offer an unacceptable QoS. Therefore, we decided to

limit our study to lower slack factors. Thus, for our set of experiments we define an interval of slack factor from 1 to 20 with steps of one.

3.4.2 Single Service Level – Parallel Machines (SSL-PM) algorithm

For the design of the SSL-PM algorithm experiments, we need to consider that now system model instead of having just one processing unit can have any number $m > 1$. The number of machines is another parameter that can have any integer value greater than one. Here we considered that the systems are designed with a number of machines power of two, thus, we proposed systems with 2, 4, 8, 16, 32 and 64 machines to design our experiments. The slack factors of the single SLA are taken from the same interval considering for the single machine.

3.4.3 Multiple Service Level - Single Machine (MSL-SM) algorithm

For the MSL-SM algorithm, we consider a system model with a single machine $m = 1$ and that jobs submitted can have more than one SLA therefore, the value of slack factor of the jobs within a workload can be different. In order to design the experiments for this algorithm we need to consider that there are two independent parameters, the number of SLAs and the values of slack factor of each SLA. Considering both parameters we can have a large number of combinations. In order to avoid this and to have a manageable number of experiments, we established the number of SLAs as our main variable and restricted its value to the interval from one to twenty. Then, for assigning a value of slack factor to each SLA we consider the interval used in the SSL experiments from one to twenty. The idea is that, if the provider has only one SLA to offer, he

would seek to prioritize the QoS and then the slack factor that he will assign to that unique SLA would be a value of one (the best SL). In case the provider manages two SLAs, he will maintain offering one SLA with a $sf = 1$ and the other with a $sf = 2$. And this logic will be kept as many SLAs the provider has to offer. Table 2 shows this distribution of slack factors over the number of SLAs available.

Table 2. Slack factor distribution over the number of SLAs available.

Number of SLAs	SLAs' slack factor
1	$SLA_{sf}^1 = 1$
2	$SLA_{sf}^1 = 1, SLA_{sf}^2 = 2$
3	$SLA_{sf}^1 = 1, SLA_{sf}^2 = 2, SLA_{sf}^3 = 3$
4	$SLA_{sf}^1 = 1, SLA_{sf}^2 = 2, SLA_{sf}^3 = 3, SLA_{sf}^4 = 4$
5	$SLA_{sf}^1 = 1, SLA_{sf}^2 = 2, SLA_{sf}^3 = 3, SLA_{sf}^4 = 4, SLA_{sf}^5 = 5$
6	$SLA_{sf}^1 = 1, SLA_{sf}^2 = 2, SLA_{sf}^3 = 3, SLA_{sf}^4 = 4, SLA_{sf}^5 = 5, SLA_{sf}^6 = 6$
7	$SLA_{sf}^1 = 1, SLA_{sf}^2 = 2, SLA_{sf}^3 = 3, SLA_{sf}^4 = 4, \dots, SLA_{sf}^7 = 7$
...	...
20	$SLA_{sf}^1 = 1, SLA_{sf}^2 = 2, SLA_{sf}^3 = 3, \dots, SLA_{sf}^{19} = 19, SLA_{sf}^{20} = 20$

Once we established how the SLAs are composed, we propose that the mechanism that assigns the SLAs to the jobs in the charge would be random with a uniform distribution. The uniform distribution represents that the user is equally likeable to select any of the SLAs offered by the provider to submit his jobs. We think that this is possible if the price scheme offered to the user gives an equal cost-benefit to all the SLAs offered. Thus, the price function used by the provider must be such that for a unit of increment in the QoS offered it would demand in the same proportion of economic return from the user. In case that the price function does not comply with this relation then the user might be inclined to select a SLA that offers a better cost-benefit. In this case, we could not consider that there is a uniform distribution for selecting SLAs to submitted jobs.

In conclusion, we think that there is a relation between the price of each SLA and the distribution that the different SLAs will have among the jobs received, therefore we need to reflect that relation when we consider the function that assigns SLAs to the different jobs and the price function.

3.4.4 Multiple Service Level - Parallel Machines (MSL-PM) algorithm

These experiments use a combination of the multiple machine parameters discussed for the Single SL - Parallel Machines and the Multiple SL parameters from the Multiple SL - Single Machine algorithms experiments. Therefore we have seven times the number of experiments that in the case for the MSL-SM which corresponds to evaluate the system's performance in the different multiple machine models. These systems are the closest models to real systems where there exists different machines and the provider offers different SLAs in order to satisfy the diversity of processing needs and budgets from a variety of users.

3.5 Criteria analysis

In order to provide an effective guide in the selection of the best algorithm and its parameters, we perform a joint analysis of several metrics according with the methodology proposed in (Tsafrir et al., 2007), and applied to Grid scheduling problems by Ramírez Alcaraz et al., (2011). They introduce a method assuming equal importance for each metric. The objective is to find a robust strategy and with good performance under all the testing cases expecting that it would keep its performance under other conditions, i.e. other number of SLAs, values of slack factor or a different workload.

The analysis is performed as follows. First, we evaluate the degradation (relative error) of each metric under each parameter using the following formula $\left(\frac{\text{metric value}}{\text{metric best value}} - 1\right) \times 100$. In this way, for each metric, each algorithm parameter is characterized by a tuple of values, reflecting its relative degradation under all the metrics. Afterwards, we average these values (assuming equal importance for each metric), and sort them. The best algorithm which has the lowest average degradation has a rank of one.

3.6 Workload

In order to provide a realistic case of study we employ for our simulations a Grid workload based on real production traces. These traces are logs from real parallel computer systems and give us a good insight of how our proposed schemes will perform with real users. The predominance of low parallel jobs in real logs is well known. However, some jobs require multiple processors. In a later case, we consider that machines in our model have enough capacity to process them, so we can abstract their parallelism.

IaaS clouds are a promising alternative to computational centers; therefore, we assume that workload submitted to the cloud will have similar characteristics to the workloads submitted to actual parallel and grid systems.

In our log, we considered nine traces from: DAS2—University of Amsterdam, DAS2—Delft University of Technology, DAS2—Utrecht University, DAS2—Leiden University, KTH, DAS2—Vrije University Amsterdam, HPC2N, CTC, and LANL. Details of the log characteristics can be found in the PWA (Feitelson D., 2008) and GWA (Iosup et al., 2008).

To obtain valid statistical values, 30 experiments of one week interval were simulated for each experimental scenario. We calculate job deadlines based on the real processing time of the jobs. Therefore we do not use user-run time estimates.

Figure 6 shows a histogram of the number of jobs in the simulation period of 30 weeks.

Figure 7 shows a histogram of the processing time of the jobs in the workload. For clarity, the view of this histogram has been limited in the vertical axis to 10,000 jobs. The first bin for jobs with processing time in the range of 1 to 1000 seconds is the only one limited; this interval has around 470,000 jobs.

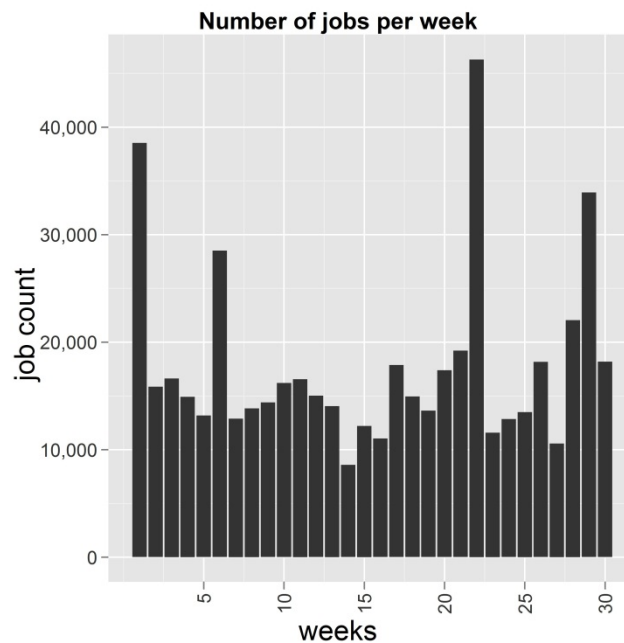


Figure 6. Number of jobs in the experiment interval.

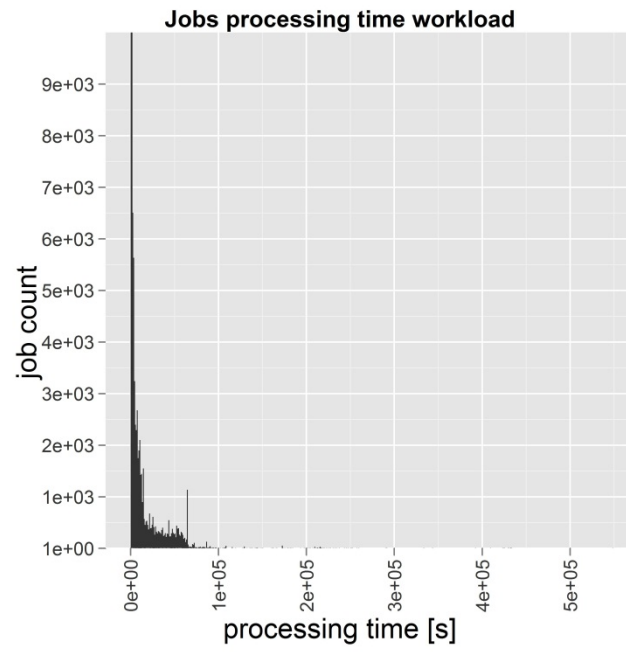


Figure 7. Processing time of the jobs in the workload.

Chapter 4

Experimental Competitive Factor Analysis

In this chapter we analyze the experimental results that we obtained. We focus our analysis in the competitive factor and in the following chapters we present the system benefit and the other secondary metrics.

4.1 Price function

IaaS just like any other service is an interchange of some work performed for a user and a return in some form of economic benefit to the provider. In order to establish an economical return rate the provider uses some price strategy, which involves considering key factors, including the market offer of the service, the costs associated with providing the service, and an understanding of the relationship between QoS and price.

There exist many works that deal with the problem of pricing the execution of the jobs in different processing models such grids and clouds considering different market models (Buyya et al., 2008). In a real world market, there exist various economic models for setting the price for services based on supply-and-demand and their value to the user. Implementing those models would require implementing an adaptive pricing model.

Since the implementation of these models is beyond the objectives of this thesis, we stuck our analysis considering a monopolistic/oligopolistic market (Buyya et al., 2002), where the providers dominates the market and sets the prices of its service

without considering market conditions or the possibility of negotiating the prices with the users.

With the previous assumption, we propose a price function that maps the slack factor of a SLA to a price per unit of execution, taking into consideration that:

1. The price function should be monotonically decreasing with regard to the slack factor
2. The price function should be designed so that it covers the execution costs of the system

The first condition guarantees that the user will receive a proportional QoS guaranteed to the price he pays. The second condition guarantees that the provider will never have losses for providing the service. The costs that a provider could have are diverse and include wages, maintenance, electric bills, etc.

We consider that if the system does not implement a SLA admission control, it will be able to execute all the jobs available in the workloads. Thus, the provider will receive a payment for the execution of all these jobs and the cost associated with the operation of the system will be charged to each user in proportion to the time that his jobs were executed.

When rejection through SLAs is implemented the amount of jobs accepted limits the processing time that the system performs, however, the system would maintain the same operational costs. Thus, the provider needs to transfer the same costs to a lower number of jobs raising the price per execution unit of each one. Equation (8) captures the previous idea.

$$cost_{\lambda} = \alpha \frac{\sum_{i=1}^{n_r} p_i}{\sum_{j=1}^n p_j} \quad (7)$$

In this equation, α is a constant that represents the cost that the provider needs to cover to keep the system in operation. The numerator represents the total processing time of the jobs releasing to the system, which is the time that the system could execute if no admission control were implemented. And finally, the denominator represents the sum of processing times that the system executes under a λ configuration. Thus, the proportion of processing time of the total processing time available in the workload and the value reached by the system determines the cost per execution unit of each job. Please note that this equation keeps valid without regarding the number of machines in the system, because the processing time in the workloads remains unchangeable.

The design of the price function was based on the experimental results of the total processing time (*TPT*) metric for the SSL-SM algorithm with $\alpha = 1$. The graph in Figure 8 shows the cost that each slack factor has and the proposed linear price function. With this price function we meet the two conditions that this function must have. The price decrements as the QoS decrements and the prices are always greater than the costs. For each sf , the provider will obtain the difference between the costs and the prices as profit. The form of the function maximizes the profit for low and middle values of slack factor. The values of $sf = 1$ and $sf = 20$ have a profit of zero.

Finally, this price function has another characteristic. Since there is a linear relation between price and sf , we can justify the use of a uniform distribution to assign jobs to SLAs. The reason is that a potential user would need to spend equal

amount of money for an equal increase on the QoS received making all the SLAs equally probable of being selected.

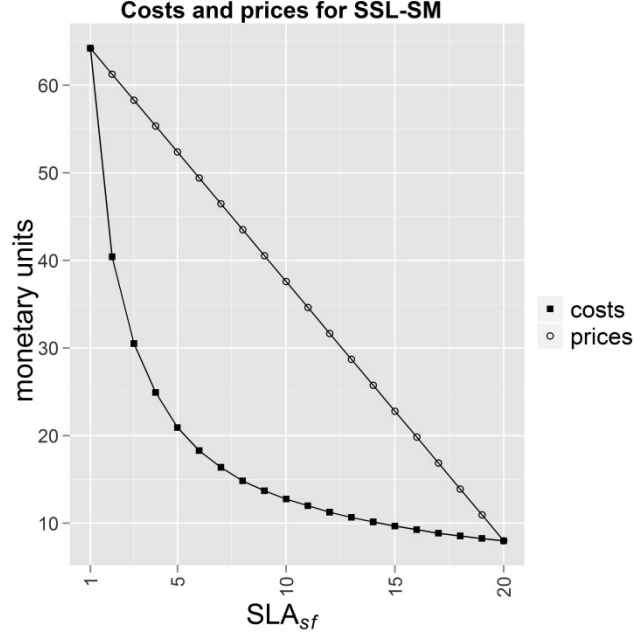


Figure 8. Cost and prices for the SSL-SM algorithm.

In the following sections we present the upper bound of the optimal income with unlimited \hat{V}_u^* and limited resources \hat{V}_l^* for the studied algorithms.

4.2 Optimal income bounds

With the price function defined in previous chapter, we obtain the optimal income bounds used to calculate the competitive factor for the different algorithms. These are bounds of the maximum income achievable by the algorithms given the workloads used and considering both definitions of optimal algorithm with a system with limited resources and unlimited resources. Values in the plot are normalized

by the maximum benefit obtainable i.e. the maximum price (u_{max}) per the mean sum of processing time ($MSPT$) of the workloads.

According with our definition, \hat{V}_u^* only depends on the number of machines in the system. The plots in Figure 9 show the bounds for the SSL and MSL algorithms with unlimited resources. In the case of the SSL its value depends on the values of u_i for the different SL therefore depends on the price function. The MSL considers that all the jobs released can be executed by the maximum price u_{max} i.e. the price of the SL with tightest sf . Thus, this bound is equal to the $MSPT$ per u_{max} for all number of SLAs.

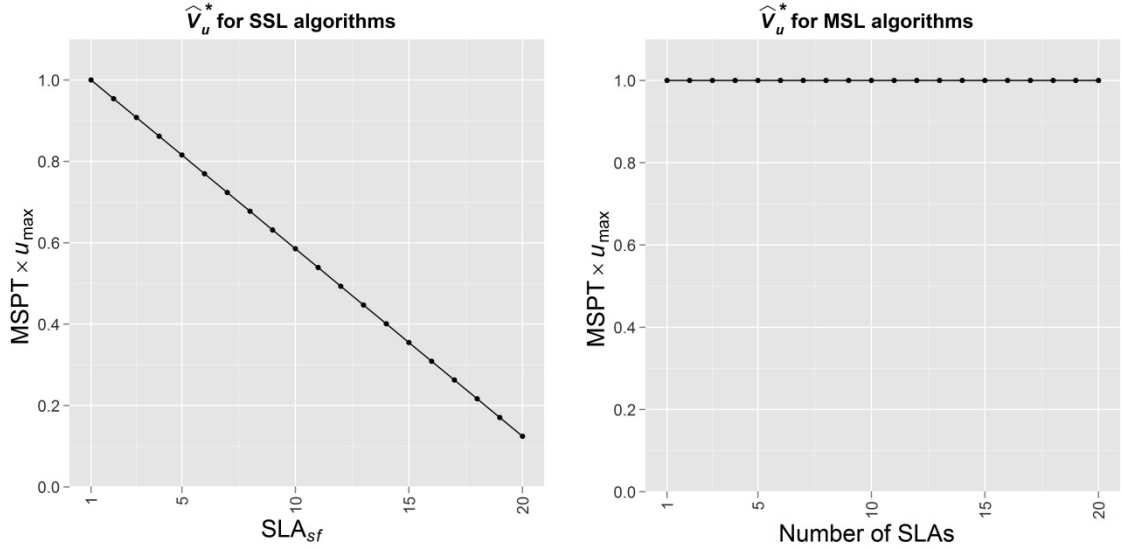


Figure 9. Upper bound of the unlimited resources optimal income for the SSL and MSL algorithms.

In case of the limited resources bound, it has two components, the sum of processing times of the workload ($\sum_{j=1}^{n_r} p_j$) and the maximum deadline per the number of machines in the system ($d_{max} \cdot m$), where the minimum of these values

define the optimum's bound of processing area. The sum of processing times is a constant, therefore $\hat{V}_l^* \leq \hat{V}_u^*$.

Figure 10 shows the income bound for the SSL algorithms. We see that the 64 machines system obtains the maximum gain bound for each sf except $sf = 1$. The same happens for some values of sf for the 32, 16 and 8 machines systems.

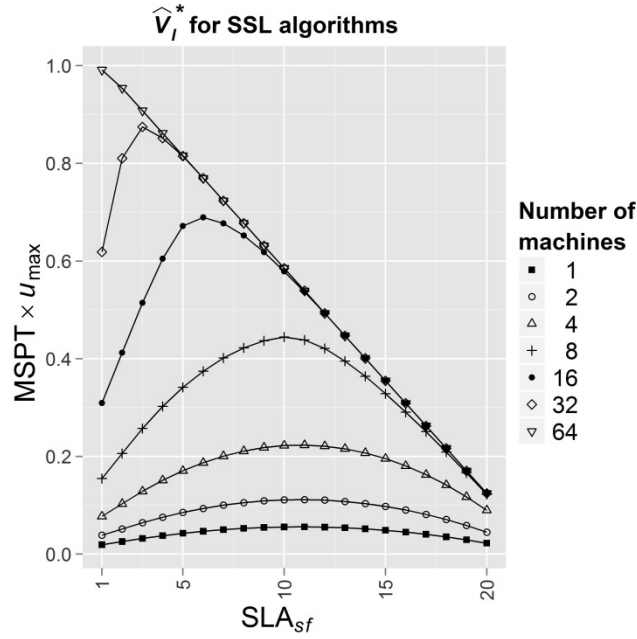


Figure 10. Upper bound of the limited resources optimal income for the SSL algorithms.

This equality is reached when the maximum deadline per number of machines becomes larger than the sum of processing times in the workload. When this

happens $\min\left(\sum_{j=1}^{n_r} p_j, d_{\max} \cdot m\right)$ is equal to $\sum_{j=1}^{n_r} p_j$, thus both bounds have the same

value and it no longer depends on the sf . Whether this equality occurs or does not depends on three parameters: the specific workload, the value of sf (which defines d_{\max}) and the number of machines in the system.

Figure 11 shows the benefit bound for the MSL algorithms. We can appreciate that the difference on the bounds for both types of algorithms is substantial. For this algorithm in case that the $d_{\max} \cdot m > \sum_{j=1}^{n_r} p_j$, the value of \hat{V}_l^* reaches the maximum value of benefit ($MSPT \times u_{\max}$) this happens for some number of SLAs in the 64, 32, and 16 machines systems

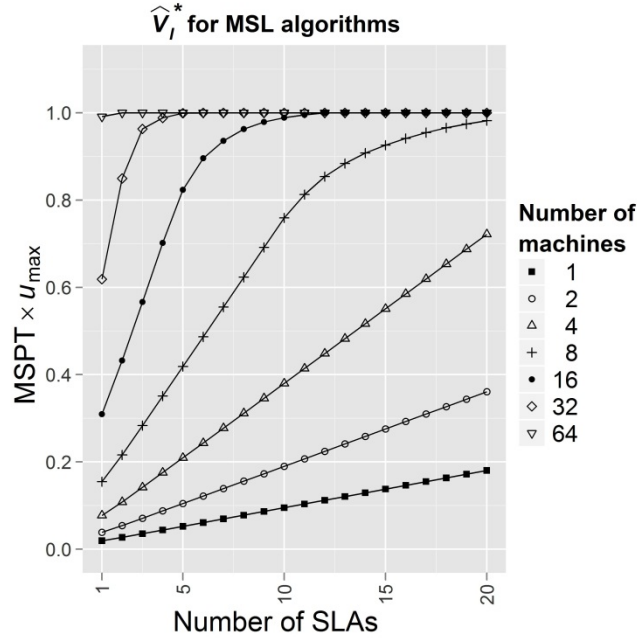


Figure 11. Upper bound of the limited resources optimal income for the MSL algorithms.

Finally, in Figure 12 we present a matrix that summarizes for the different weeks in the workload, and number of machines, the value of sf where $d_{\max} \cdot m > \sum_{j=1}^{n_r} p_j$. The vertical axis represents the weeks in the workload, the horizontal axis the number of machines in the system, and the thermometer colors the value of sf where the maximum deadline per number of machines reaches the value of the sum of processing time in the specific week. In case that there is not color presented it means that for that particular workload the equality was never met. We see that only the one and two machine systems do not present this effect. In the four machine system the effect only happens in one workload and for $sf = 20$

so it cannot be appreciated in the graphs of Figure 10 and Figure 11. Instead, for the next systems the value of sf where the equality occurs is even lower. The extreme case is the system with 64 machines where $d_{\max} \cdot m > \sum_{j=1}^{n_r} p_j$ is reached for $sf = 1$ on 29 of the 30 weeks. For these systems we consider that the workload is not suitable for the system configuration.

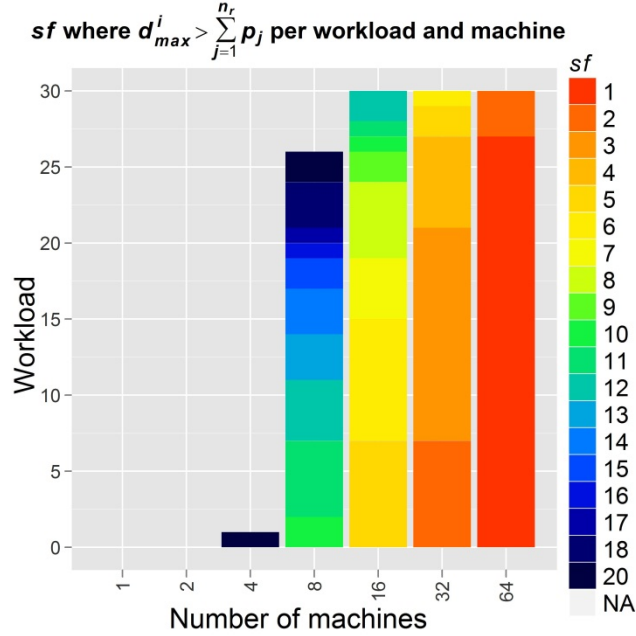
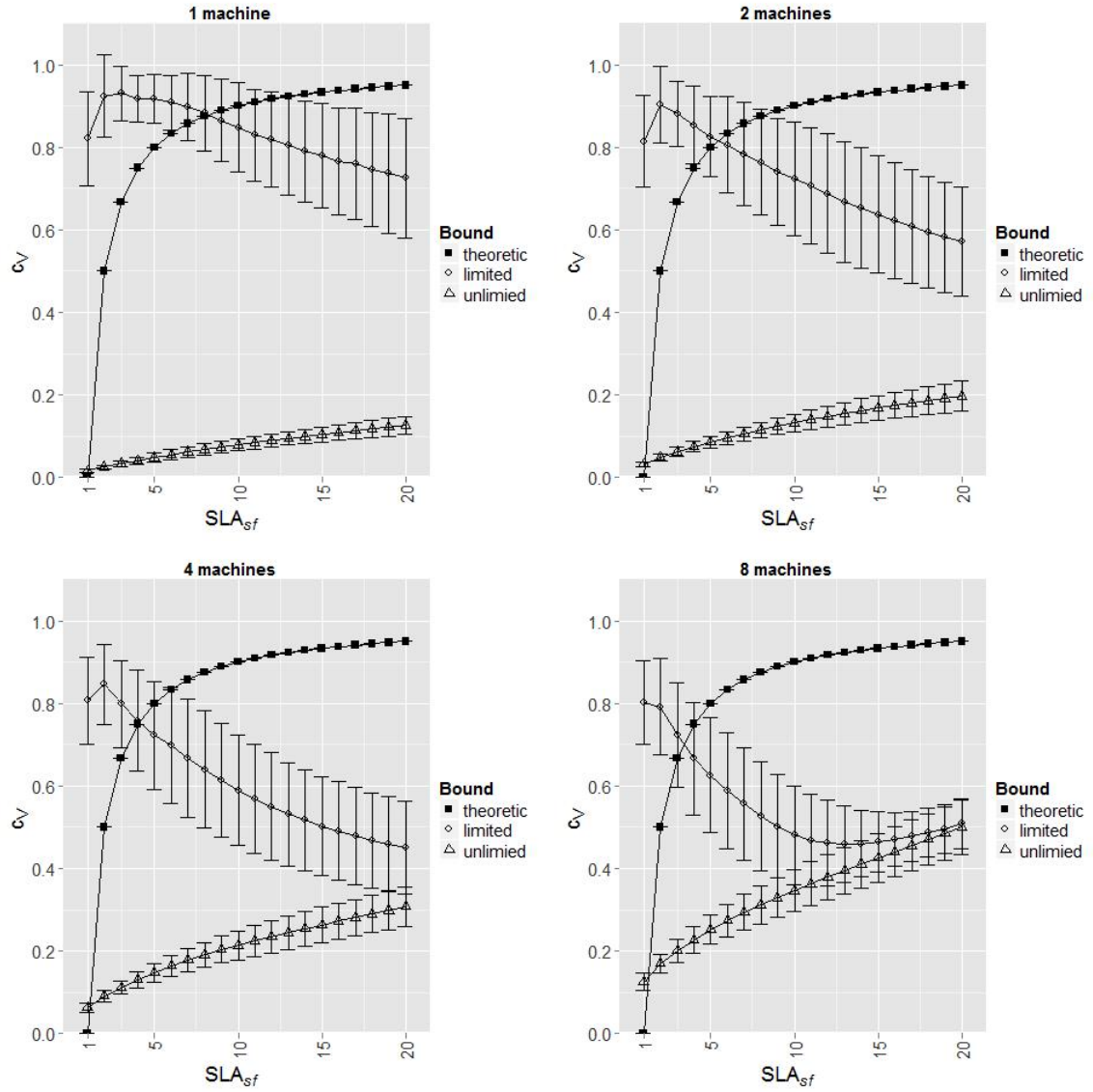


Figure 12. Saturation matrix for the competitive factor limits.

4.3 Competitive factor

In this section we present the values of the c_V for the SSL and MSL algorithms. We show the results for each machine and for the three \hat{V}^* bounds: the theoretic, the one considering unlimited resources and the one considering limited resources. The plots show the mean of the c_V obtained in the 30 experiments and an error bar

representing one standard deviation up and one standard deviation down of the mean.



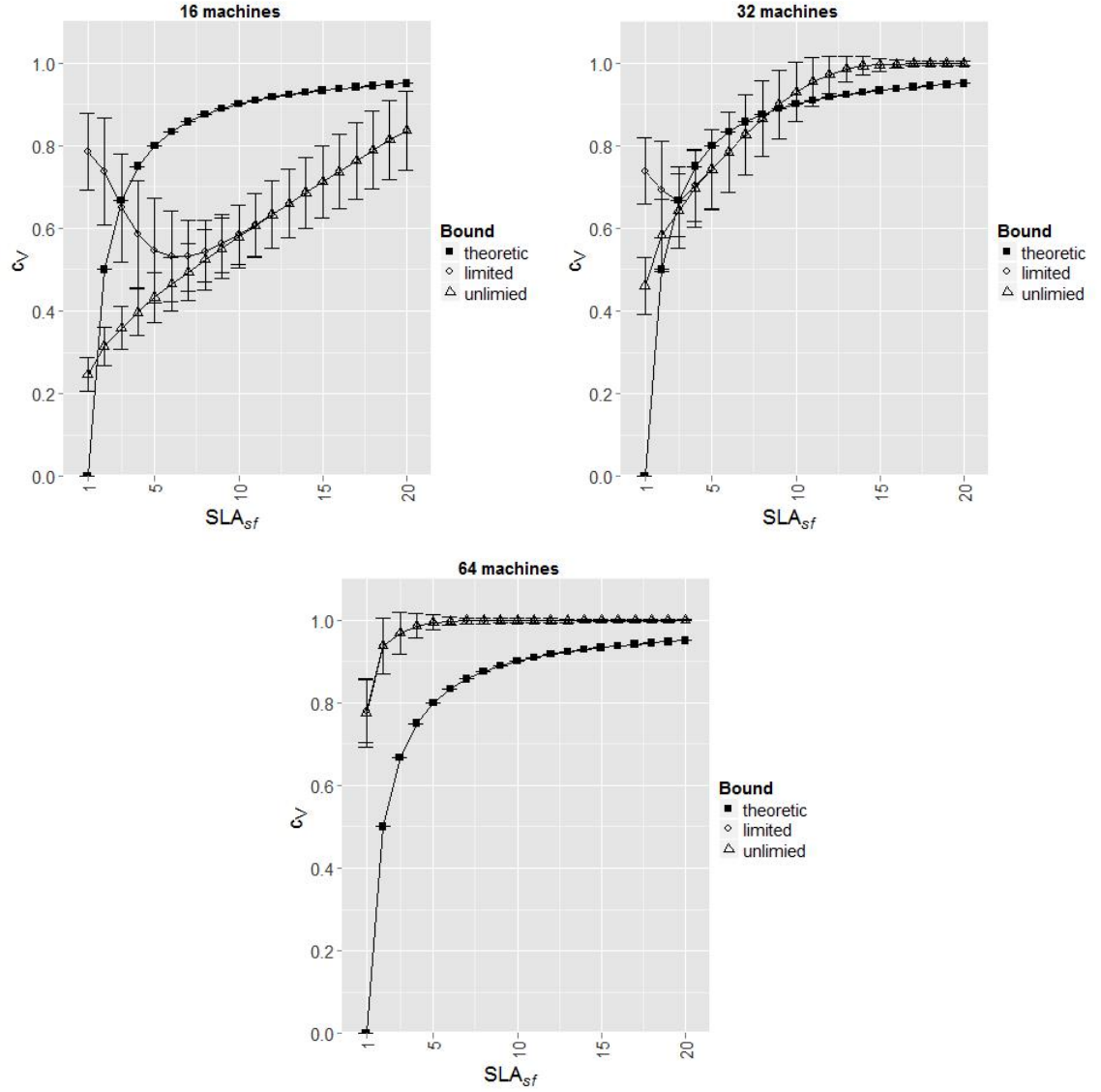
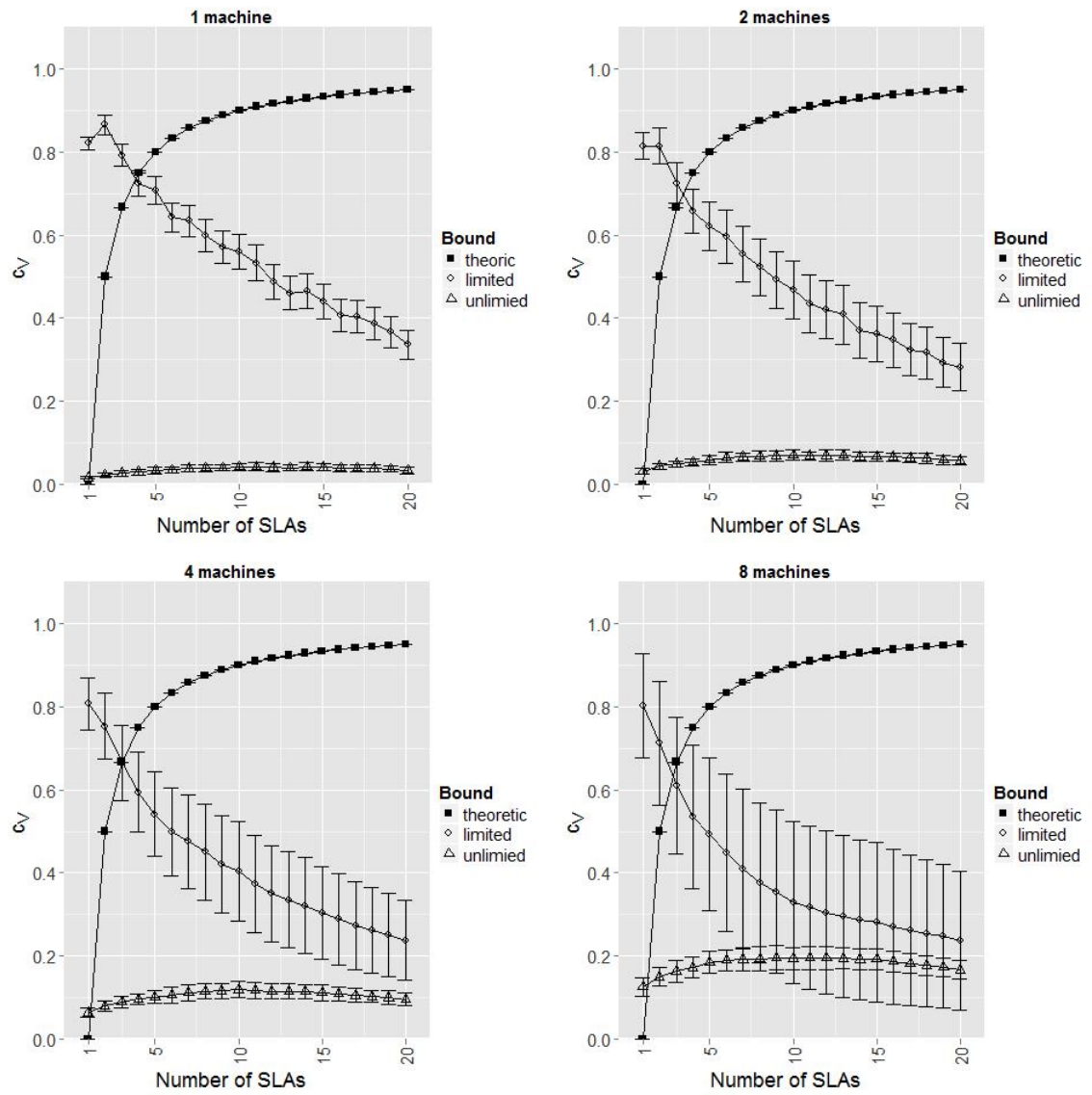


Figure 13. Competitive factor for the SSL algorithms.

We see in the plots of Figure 13 that the value of c_V of the SSL algorithms for the limited and the unlimited bounds gets closer as the number of machines increases. We also observe that for low sf the limited bound of c_V is tighter than the theoretic bound, which means that for the workload used experimentally the algorithm performs better than what theory predicts. When the number of machines in the system is very large, the c_V with experimental bounds is larger than the theoretic, this is caused by the workload used.



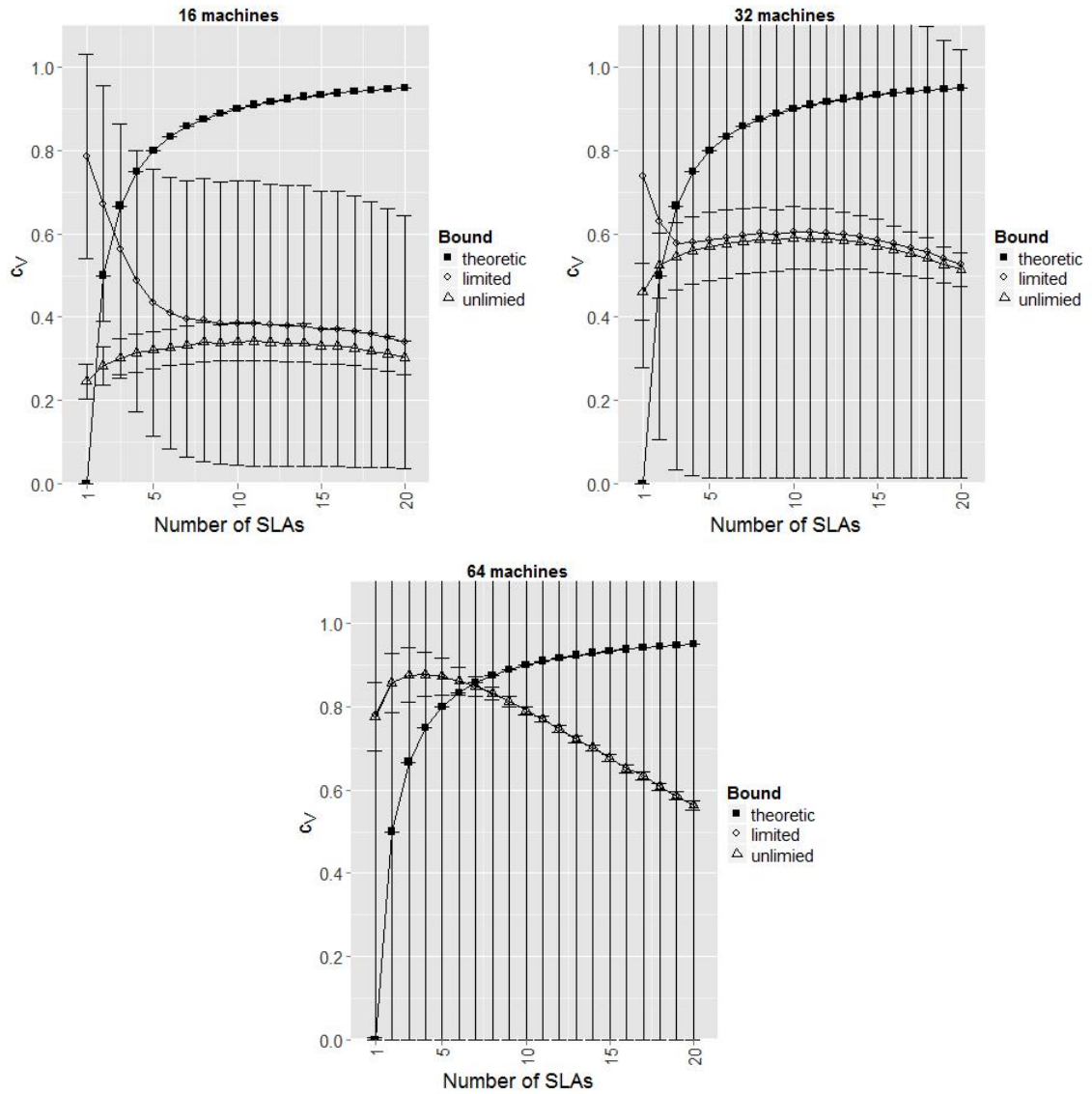


Figure 14. Competitive factor for the MSL algorithms.

In the series of graphs of Figure 14 we see both bounds for the MSL algorithms. For MSL algorithms with low number of SLAs, the experimental bound with limited resources is a tighter bound than the theoretic. As in the case with the SSL algorithms, we see that the unlimited and limited bounds tend to a same value when the number of machine increases. From the figure, we also observe that when the number of machines increases the standard deviation of c_V for the limited

bound increases, thus we obtain, for some experiments, competitive values very close to the optimal and in others very far apart from it.

Chapter 5

Service provider benefits

The graph in Figure 15 shows the service provider benefits for the SSL-PM algorithm. We see that benefit is lower in the extremes sf values and greater in the middle values. The reasons of this trend are both, the price function and the TPT value that each sf obtains. The system with 64 and 32 machines suffers from the lack of jobs in the workload and instead of following the same trend that the other systems they rapidly fall. This fall is so abrupt because the cost of keeping a system with such resources requires a constant increasing in execution time. We observe that for the system provider the benefit always has a positive value so there is not region with operational loss.

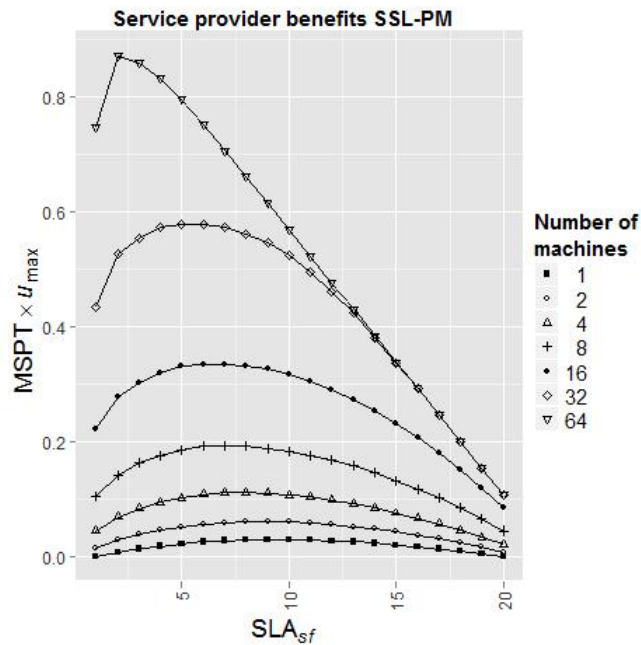


Figure 15. Service provider benefits for the SSL-PM algorithm.

Figure 16 shows the service provider benefits for the MSL-PM algorithm. We appreciate that the benefit is quite constant through all the SLAs and mainly depends on the number of machines in the system. We can conclude that for this algorithm the use of more machines will be always beneficial for the system provider. The 64 machines system suffers from a fall in the benefit caused by the unavailability of jobs in the workload, therefore in order to guarantee a good performance is necessary that the system never runs out of jobs to be processed. In general this can be accomplished by dynamically adjusting the number of machines depending on the availability of jobs.

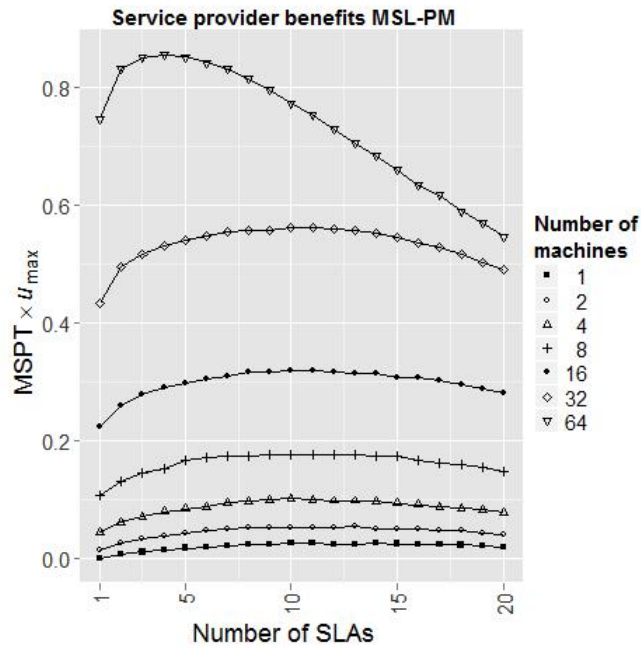


Figure 16. Service provider benefits for the MSL-PM algorithm.

In Figure 17 we see the plot of the service provider benefits for the MSL-PM algorithm. The values are relative to the ones obtained by the MSL-SM so we can appreciate the effect of the increment in the number of machines. The plot contains horizontal lines marking the 2, 4, 8, 16, 32 and 64 values. These marks make clearer the presence of values in the number of SLAs were the increment in

the service provider benefits is larger than the increment in the number of machines from the perspective of a single machine. The number of SLAs where the increment in the service provider benefits is superior to the number of machines added decreases as more machines are added to the system. The system that outperforms faster and does it in a greater rate is the 64 machines system follow by the 32 machines system.

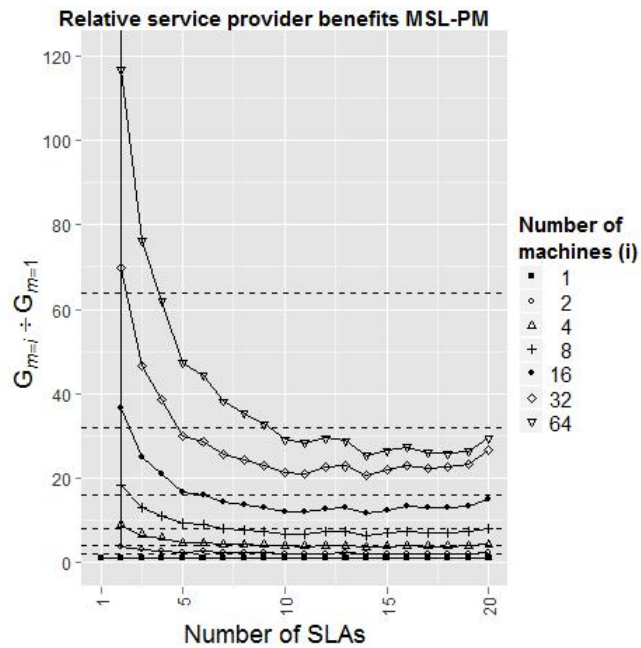


Figure 17. Relative service provider benefits for the MSL-PM algorithm.

In this graph we can conclude that the multiple machine system allows a better service provider benefits. Thus, considering only this metric from the provider perspective having greater number of machines is always beneficial.

Chapter 6

Experiments Results

In this chapter we present results obtained for the simulation of the SSL-SM, SSL-PM, MSL-SM, and MSL-PM algorithms. Each algorithm has different objectives, and was designed in an incremental manner to evaluate a new parameter in each step considering the results previously obtained. The SSL-SM evaluate the change in the sf on the different set of metrics. In the next step, with the SSL-PM algorithm, we evaluate the change in the number of machines in the system. With the MSL-SM algorithm we compare the results of the MSL algorithm with the previously obtained SSL in a system with a single machine. Finally, with a MSL-PM algorithm we have a complete system where we can evaluate the change in the number of machines when multiple SLs are used.

In this chapter, each point in the graphs represents the mean value of the 30 experiments, one experiment for each week in the workload. In case, an error bar is shown, this has a length of two standard deviation of the mean.

6.1 Single Service Level – Single Machine (SSL-SM) algorithm

In this section we present the results of the metrics for the SSL-SM algorithm. In this algorithm all the jobs submitted to a single machine system have the same SLA and therefore the same slack factor.

Figure 18 shows the total processing time (TPT) as a factor of the mean sum of processing times ($MSPT$) of the jobs in the workloads. The $MSPT$ is obtained as the average of the sum of processing time in each of the 30 workload. Since the

results we present comes from the average value obtained in each of the 30 experiments, the only way that a given algorithm can reach this $MSPT$ is by executing all the jobs in each of the workloads. By presenting the total processing time in this form, we can give a general perspective of the results, as well as an easy way to later compare the different algorithms.

In the graph, we can observe that the system is able to execute a tiny fraction of the processing time available in the interval of slack factor from 1 to 20. The general trend shows that the system is able to increment the processing time as the slack factor of the SLA increases. We could expect that if the system receives a workload with a SLA with greater slack factors the processing time would continue increasing at a lower rate. Thus, in order to be able to execute all the processing time available we will require very large values of slack factor.

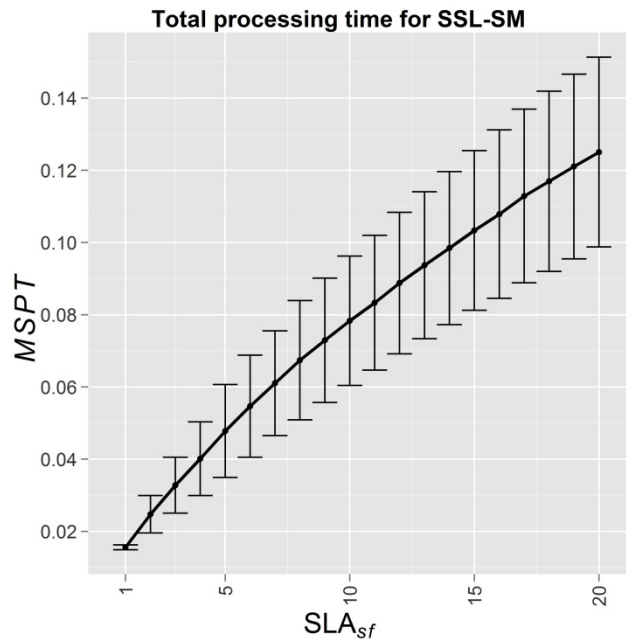


Figure 18. Total processing time for SSL-SM algorithm.

The graph in Figure 19 shows the percentage of jobs rejected by the system (PRJ). We see that for this algorithm the amount of jobs rejected in the interval is very close to 100% and its variation is small as the slack factor is increased. This metric has a mean value of approximately 98.7% with a peak value at 99% that corresponds to a $sf = 2$. The general trend indicates that the PRJ will decrease as the value of sf increases, but similar to the TPT metric, we would expect very large values of sf before this rejection percentage could significantly decrease.

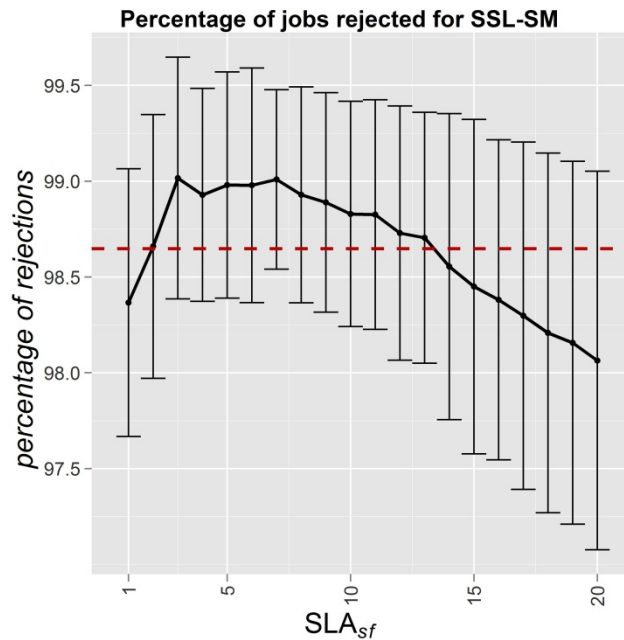


Figure 19. Percentage of jobs rejected for SSL-SM algorithm.

The graph in Figure 20 shows the mean waiting time (MWT) metric for the SSL-SM algorithm. We see, as expected, that this metric increases when the sf of the SLA increases. It reflects the flexibility of the system to delay the execution of jobs, so they spend more time waiting before get completed. It is difficult to drive conclusions about the graph, because in each experiment the set of accepted jobs is different, but we see that the increment is not constant in the interval.

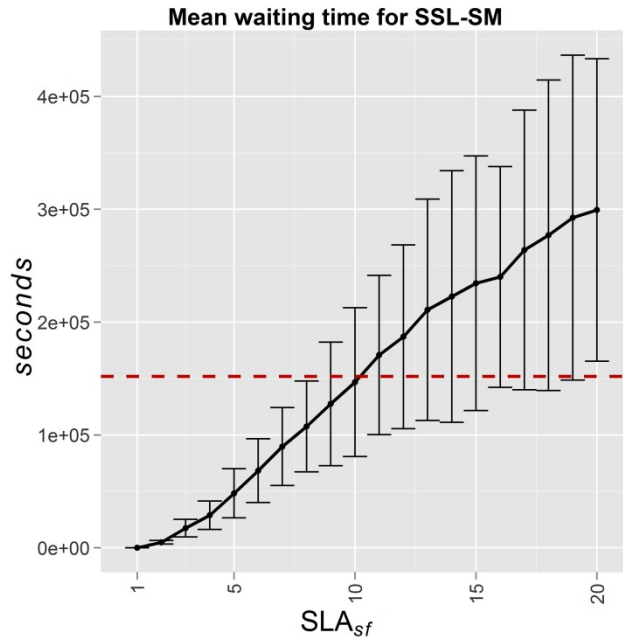


Figure 20. Mean waiting time for SSL-SM algorithm.

Figure 21 shows the mean bounded slowdown (*MBSD*) metric. As expected, this metric also increases as the slack factor increases. As mentioned in Chapter 3, the *MBSD* represents an approximation of the slack factor that the job had when it was completed by the system, thus, these values indicate that the average response of the jobs in terms of slack factor will be less than half of the slack factor used to submit the jobs. This difference might be considered positive by the user, who can obtain a faster response from the system compared with the one expected.

Figure 22 shows the machine efficiency (*ME*) metric. For this algorithm values of *sf* other than one, have an *ME* very close to one. This indicates that the machines are always executing a job at any time. When the $sf = 1$ the accepted jobs need to be executed immediately after were accepted (no waiting to start execution) so this efficiency value of 0.974 indicates the space between the completion time of the job in execution and the arrival of other.

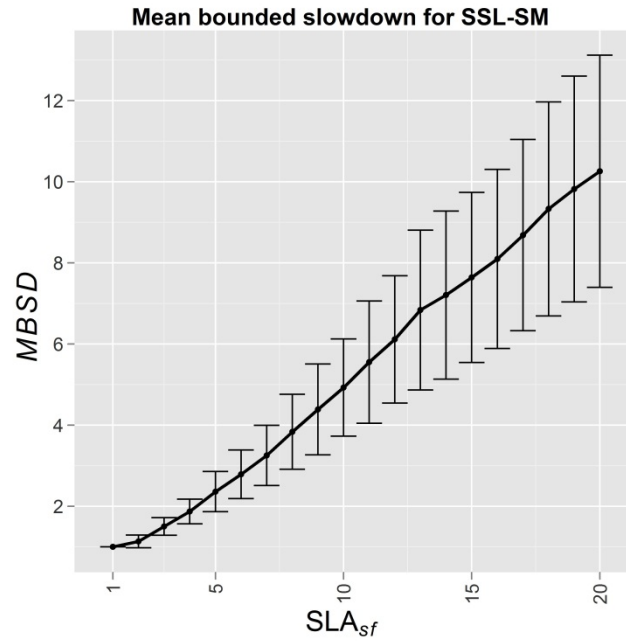


Figure 21. Mean bounded slowdown for SSL-SM algorithm.

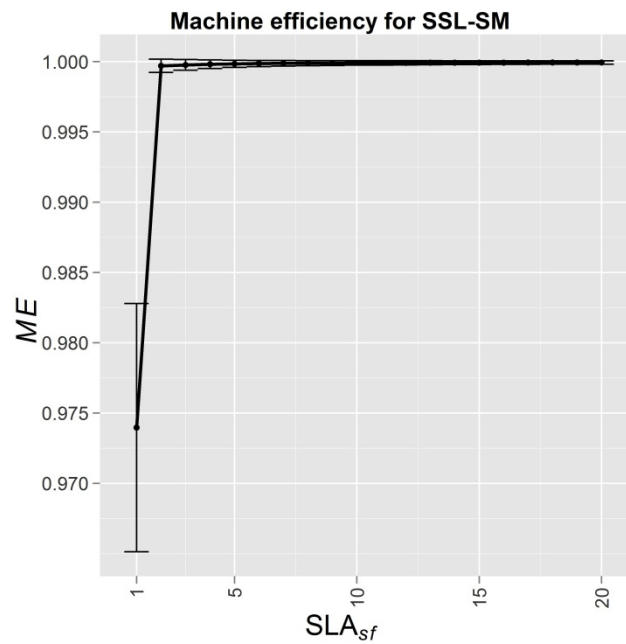


Figure 22. Machine efficiency for SSL-SM algorithm.

Finally, the mean number of interruptions per job (MIJ) metric is displayed in the plot of Figure 23. The trend shows that starting from a $sf = 1$ where no interruptions can take place, this metric jumps to a peak value of 0.72 at $sf = 2$ and then it gradually decreases until it remains stable at a value around 0.41 interruptions per job at $sf = 13$. The zero interruptions happens at $sf = 1$ since the job has to be completed before other job is accepted. Hence, this job never gets interrupted. We can explain the follow downswing as a result of the gained flexibility in delaying the execution of the jobs that allows keeping them waiting instead of preempting the current job in execution.

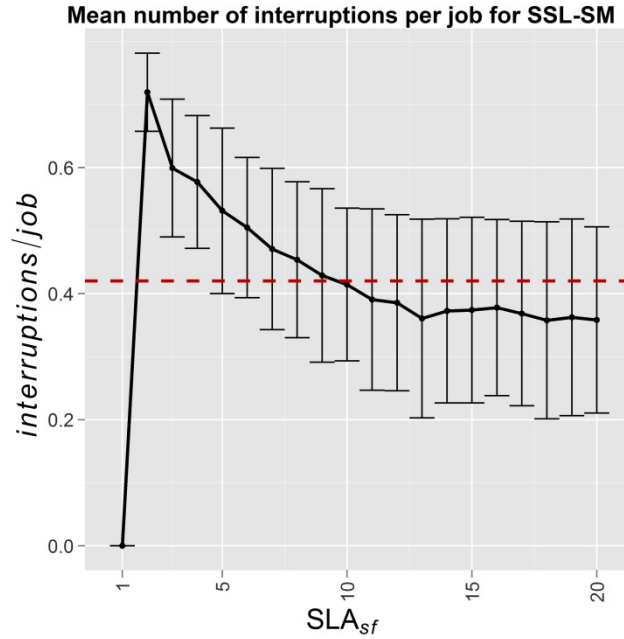


Figure 23. Mean number of interruptions per job for SSL-SM algorithm.

In general, the results of this algorithm show that the increment in the slack factor of the workload causes an increment in the TPT . This increment is not one-to-one in proportion to the increment in the sf . The increment in the sf also produces that the MWT and $MBSD$ increase. Additionally, a SSL-SM algorithm can only execute a fraction of the TPT in the workload, and accept a small percentage of the jobs

released. On the other hand, given the amount of jobs available, the SSL-SM algorithm allows that the scheduling efficiency is almost one regardless of the sf .

6.2 Single Service Level -Parallel Machines (SSL-PM) algorithm

In this section we present the results of the metrics for the SSL-PM algorithm. For this algorithm all the jobs arrive with the same SLA but the system has $m > 1$ machines to execute the jobs. For comparison purposes the results for the single machine system are also plot in the next graphs.

The TPT metric is shown in Figure 24. As in the case of the SSL-SM algorithm, for this metric instead of reporting the absolute values in time units, we plot them as a ratio of the $MSPT$. We see that, regardless of the number of machines that the system has, the TPT of the system increases as the sf of the SLA increases. The only regions that do not follow this trend are the systems with 32 and 64 machines receiving jobs with large sf . We see that in these regions, the TPT value gets too close to one, meaning that the TPT has reached its maximum possible and therefore without regarding of the increasing sf the TPT cannot further increase.

Although, the previous graph showed that there is not one-to-one proportion in the slack factor of the SLA and the TPT , plotting the TPT in terms of the processing time of the one machine system can help us to examine the effect that the increment in the number of machines has. Figure 25 shows this relative processing time where all the TPT values for the different machine system configurations were divided by the figures of the one machine system. This plot has line marks at 2, 4, 8, 16 and 32 so it is easier to appreciate if the increment in the number of machines produces a proportional increment in the processing time of the system.

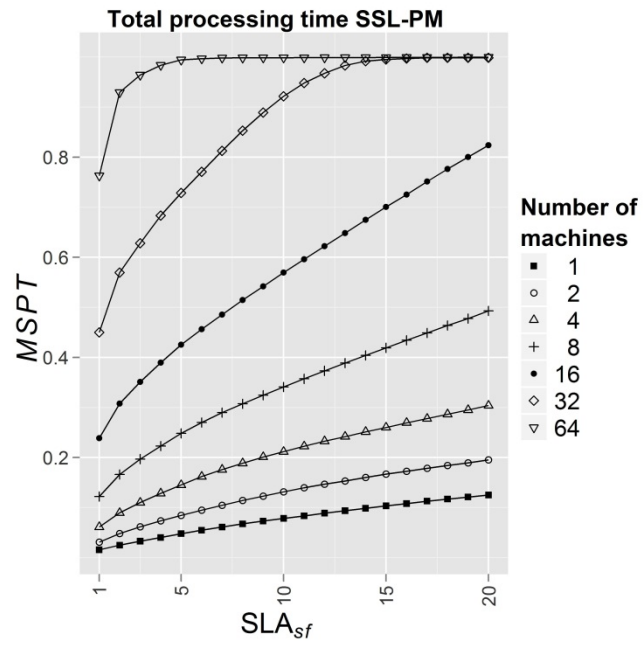


Figure 24. Total processing time for SSL-PM algorithm.

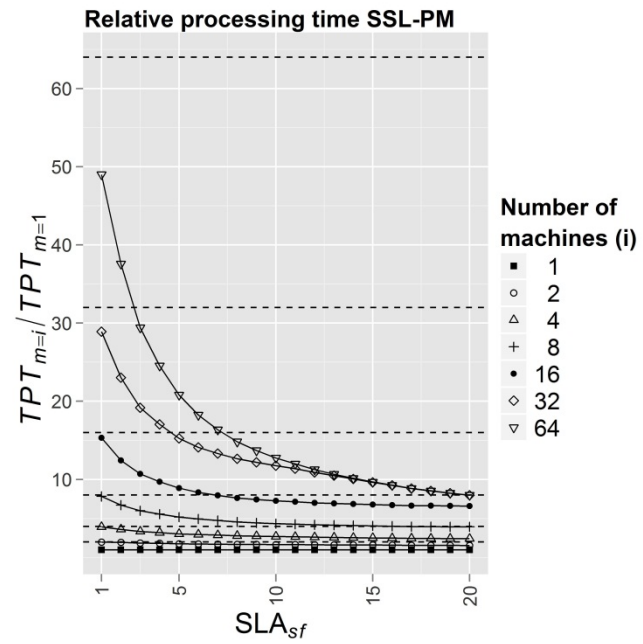


Figure 25. Relative processing time for SSL-PM algorithm.

Looking at this graph, we can easily appreciate that a desirable one-to-one increment is only produced for the two, four and eight machines system with a unitary slack factor. Once the sf is further increased the growth factor in the processing time is less than the increment in the number of machines added. This decrease is more dramatically observed for the system with 64 and 32 machines where the number of machines in the system cause that most of the jobs are processed, thus, the increment in the sf does not have the desirable increment in TPT due to the characteristics of the workload.

For the SSL-PM algorithm, the PRJ metric is showed in Figure 26. In contrast to the SSL-SM algorithm, we can more clearly observe that an increment in the processing time is the result of the acceptance of more jobs. However, when we compare the graphs of both metrics (Figure 24 and Figure 26, respectively) we can observe that the trends for the 1, 2, 4 and 8 machines are more separated for the TPT metric than for the PRJ . We consider that this effect is caused by the characteristics of the jobs in the workload and indicates that an increment in the number of machines allows jobs with largest processing time to be executed. This conclusion is based on the fact that small decrements in the rejection jobs cause largest increases in the processing time, inferring that the duration of the jobs being executed is larger than with restricted systems.

This metric also confirms the saturation effect seen in the TPT for 32 and 64 machine systems. The graph shows that for these systems when the TPT is almost one, the percentage of rejected jobs is close to zero.



Figure 26. Percentage of rejected jobs for SSL-PM algorithm.

Figure 27 shows the *MWT* metric for the SSL-PM algorithm. In this metric, when we compare the figures of the different systems models we can observe that as the system has more resources in terms of machines, the time that the jobs spend in the system decreases. This effect is completely understandable since more resources allow that accepted jobs can be more rapidly completed. Additionally, as we see in Figure 25, the proportion of *TPT* gained is less than the increment in the number of machines.

The graph shows that for all systems the increment of the *sf* produces an increment in the *MWT* values. This effect is explained as in the case of the SSL-SM algorithm due to the gained flexibility in delaying the execution of the jobs.

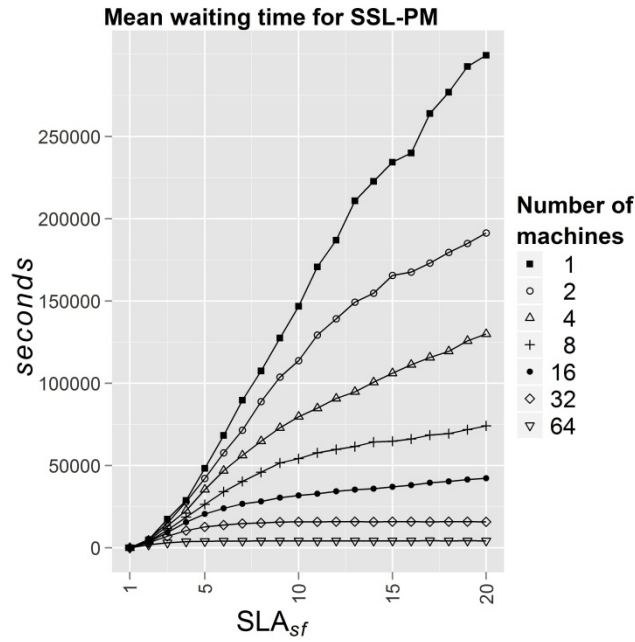


Figure 27. Mean waiting time for SSL-PM algorithm.

The *MBSD* metric for the SSL-PM metric is show in Figure 28. We observe that the main trend for all system models is that as the *sf* value of the SLA increases also the *MBSD* increases. Then, comparing the figures of the different machines for a given *sf*, we can observe that for all, as we increase the number of machines in the system, the value of *MBSD* decreases. This results and recalling the meaning of the *MBSD* tell us that the jobs will have faster response time in terms of *sf* as we increase the number of machines in the system.

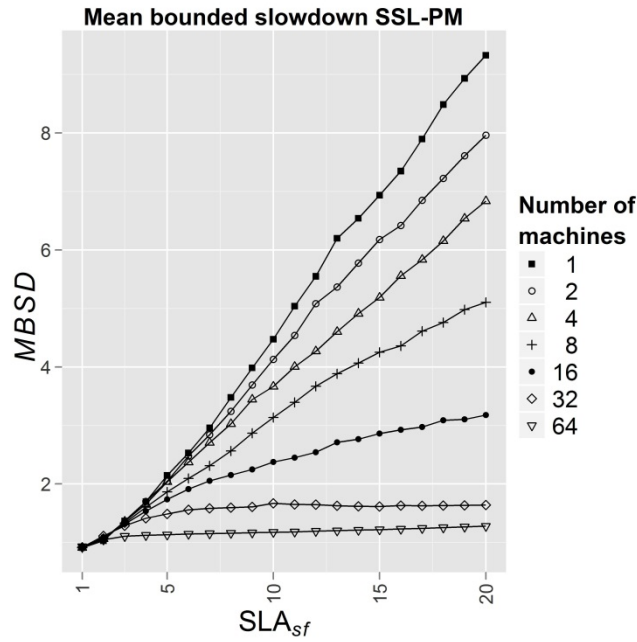


Figure 28. Mean bounded slowdown for SSL-PM algorithm.

Figure 29 shows the ME for the SSL-PM algorithm. In this graph we observe that contrary to the one machine system, $sf = 1$ performs better than $sf = 2$ for all the remaining machine configurations. The trend shown in the figures is very irregular and it is hard to explain its behavior. The only clear conclusion is that when we leave the single machine model, we cannot longer expect a 100% efficiency and there would be gaps in the interval of execution when the system would be idle in one or more of the machines. Finally, we expect that this efficiency get worst as more machines are added.

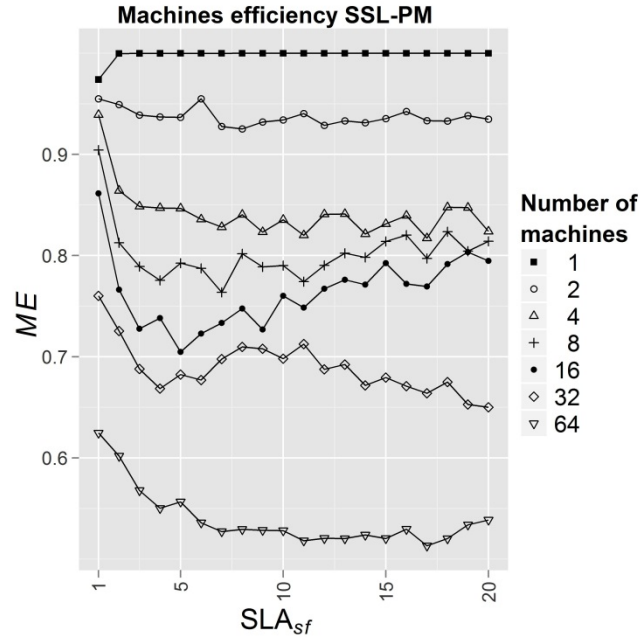


Figure 29. Machines efficiency for SSL-PM algorithm.

We conclude the presentation of the metrics for the SSL-PM algorithm with the *MIJ* metric showed in Figure 30. The plot shows that as expected with a $sf = 1$ the number of interruption for all the system configurations are zero. Once the sf is incremented the value of this metric abruptly jumps to a peak for the main system configurations. After the slack factor of two, similar to the SSL-SM algorithm, the systems with 2, 4, and 8 machines decrease the number of interruptions as the slack factor of the SLA increases. Instead, the systems with 16, 32 and 64 machines change this decreasing tendency once the slack factor reaches some value and keep this increment towards larger values of slack factor. We explain the behavior of these systems due to the steeper increment in the number of jobs accepted that can cause that more interruptions occur.

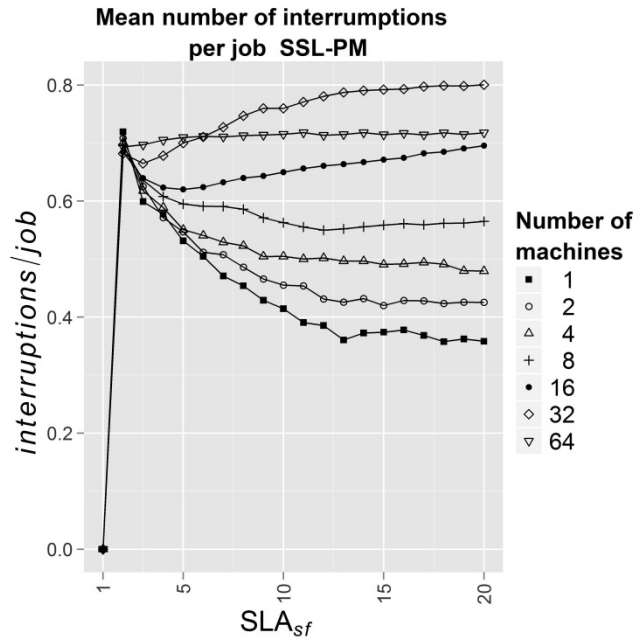


Figure 30. Mean number of interruptions per job for SSL-PM algorithm.

The results for the SSL-PM algorithm show that adding more machines to the system causes an increment on the TPT of the system. To match this increment, it would require in a single machine system a workload with very large sf . With the workload we used, we reach with a certain combination on the number of machines and sf , values where we are able to execute almost all the jobs in the workload, this causes values of TPT closest to the $MSPT$ and almost a 0% of job rejections. However, the increment on the number of machines is not proportional to the number of machines in the system; this is in part caused by the drop in the ME when the number of machines is increased. The increment in the number of machines causes that both, the MWT and the $MBSD$ decrease.

6.3 Multiple Service Levels-Single Machines (MSL- SM) algorithm

In this section we present the metrics obtained by the simulation of the MSL-SM algorithm. In order to easily see the difference between changing the sf value in a single SL and changing the number of SLAs each with a different sf value, we plot together the metrics of the MSL-SM and the SSL-SM.

Figure 31 shows the TPT for the MSL-SM algorithm. Again presented the values are relative to the $MTPT$. Plotted with a thinner line are the figures of the SSL-SM algorithm. We observe from the graph, that the TPT increases as the number of SLAs increases. This behavior can be explained due to the greater sf that are incorporated in each SLA added.

Comparing the values obtained by the MSL-SM and the SSL-SM algorithms we see that the TPT of the MSL algorithm is less than the one obtained when all the jobs arrive with the same SLA that has a sf value equal to the maximum of the multiple SLs. We came with the conclusion that when we incorporate more SLAs we cannot expect to execute the same TPT that the same workload with a single SLA with the maximum sf , but a lower value due to the effect of SLAs with lower sf .

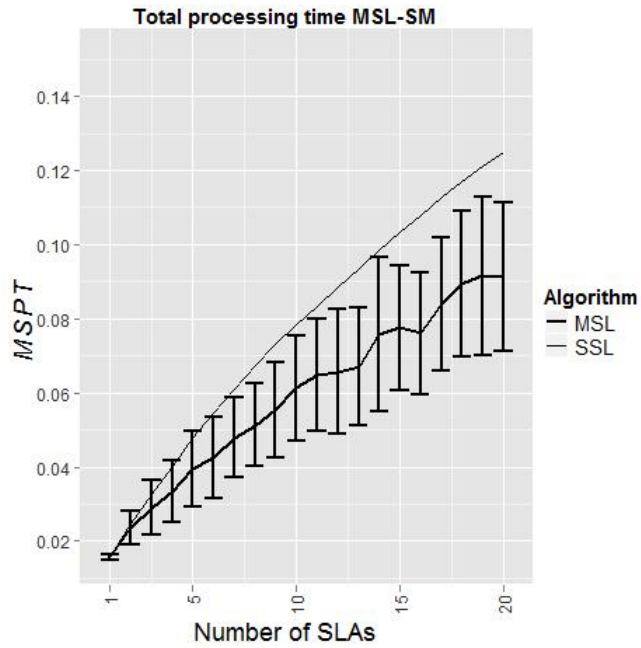


Figure 31. Total processing time for MSL-SM algorithm.

In order to better sustain the previous idea, we present in the Figure 32 a graph of how the processing time is distributed over the different SLAs for the MSL-PM algorithm. This graph shows the proportion of processing time that the system executes for the incoming SLA and the proportion of time that is executed by the previous SLAs. Thus, for example for a number of SLAs of 10, in lighter grey is shown the processing time executed by the SLA with $sf = 10$ and with darker grey the sum of processing time of the jobs executed with the SLAs with sf from 1 to 9. This graph shows us that the system accepts in a greater proportion SLAs with larger slack factors than with smaller ones. This phenomenon is especially clear when the number of SLAs is low, and the incoming SLA obtains a greater proportion of the TPT than the one that corresponds to this SLA.

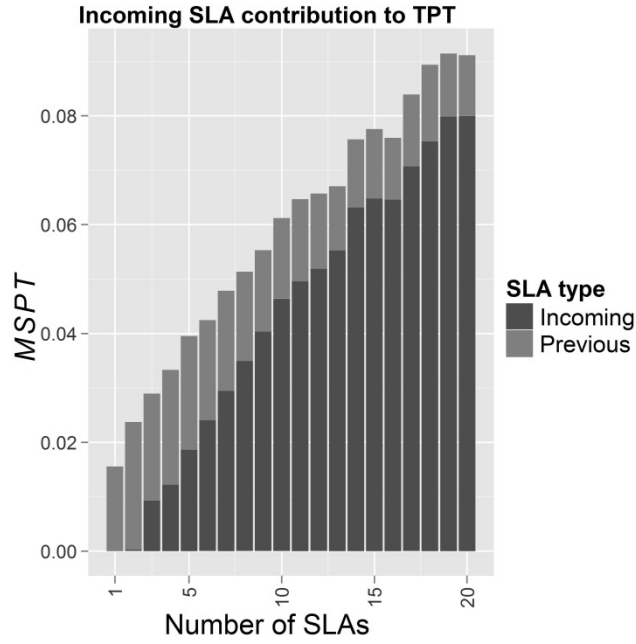


Figure 32. Incoming SLA contribution to TPT for MSL-SM algorithm.

The graph of percentage of rejected jobs for the MSL-SM algorithm is shown in Figure 33. In this graph we observe that increasing the number of SLAs does not cause a decrease in the number of jobs rejected by the system, in fact we see a little increase compared with its SSL counterpart, though this difference is negligible in absolute terms.

The graph in Figure 34 shows the MWT measure for the SSL-PM algorithm. We see that with the exception of some points, the main trend indicates that the MWT increases as more SLAs are included. By comparing these metrics with the ones from the SSL algorithm, we observe that the MWT is less than the suffered by the jobs with a single SLA. We can explain this phenomenon as an additive effect of the sf 's in the SLAs. Thus, we expect that the accepted jobs with a lower slack factors wait less, reducing the mean value of this metric compared with the result obtained by the jobs executed with the maximum sf in the set.

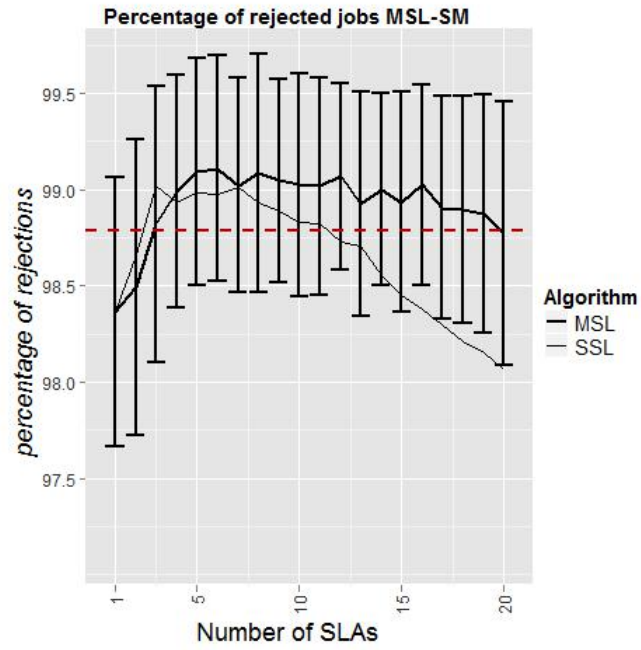


Figure 33. Percentage of rejected jobs for MSL-SM algorithm.

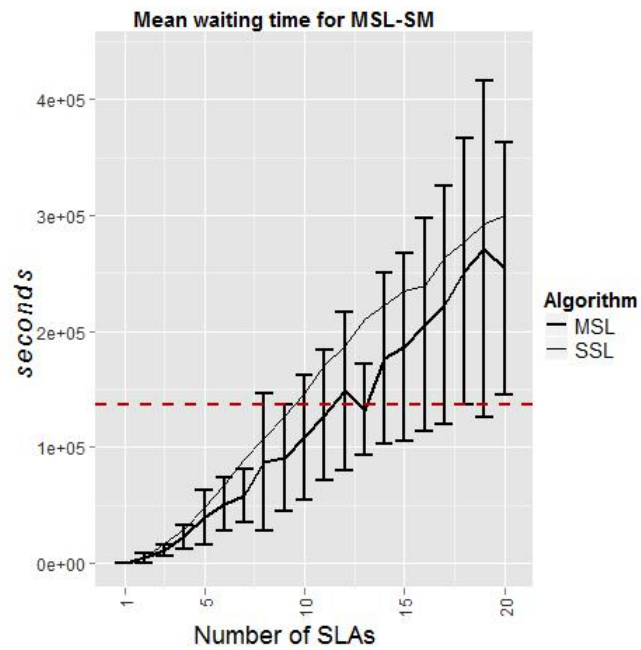


Figure 34. Mean waiting time for MSL-SM algorithm.

Figure 35 shows the *MBSD* metric. We see that this metric has a similar trend than the *MWT* metric where as the number of SLAs increases the *MBSD* value

increases. We see that for the MSL experiments the value of $MBSD$ is also less than for the SSL. This difference can also be inferred from the slack factors of the SLAs.

The machine efficiency of the MSL-SM algorithm is shown in Figure 36. We observe that the same trend as the SSL-SM algorithm is kept. Again the scheduling performed by the algorithm is very efficient for all the number of SLAs except for the SLA with $sf = 1$ where we have the same result as the SSL-SM algorithm. Although, the graph contains the metrics obtained by the SSL-SM we cannot appreciate any difference between both algorithms.

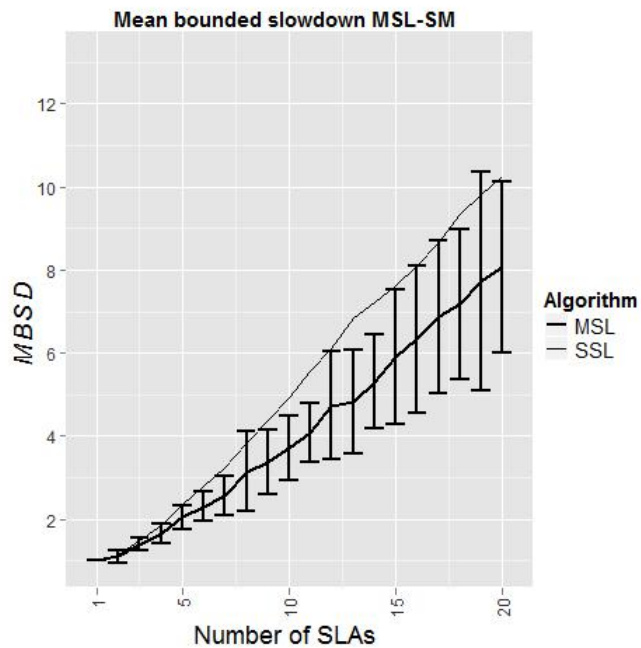


Figure 35. Mean bounded slowdown for MSL-SM algorithm.

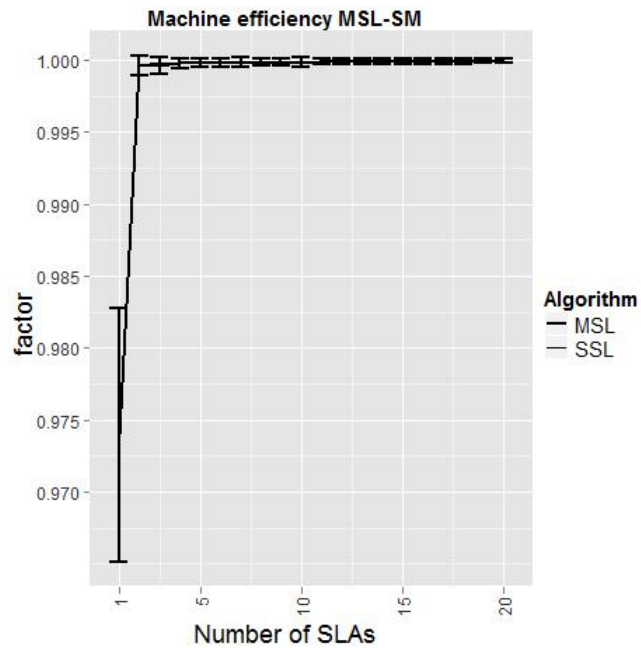


Figure 36. Machine efficiency for MSL-SM algorithm.

Finally for the results of the MSL-SM algorithm, the plot in Figure 37 shows the *MNI* metric. We see the same trend as the SSL algorithm; with only one SSL the number of interruptions is zero then the metric has a peak with two SLAs at approximately 0.74 and then decreases for the following number of SLAs. The mean value in the interval is 0.42 which is about the same as the results for the SSL algorithm. By comparing the values of the *MNI* metric for both algorithms we note that there is not a significant difference between them.

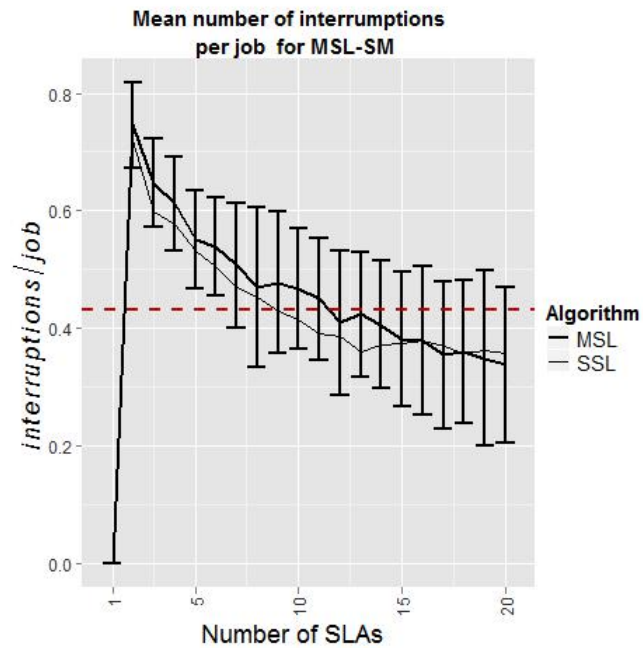


Figure 37. Mean number of interruptions per job for MSL-SM algorithm.

The main conclusion that we can make considering the MSL-SM metrics is that this algorithm shows that adding SLAs to the system does not have a dramatic effect over the metrics considered. The way we built the different sets for the MSLA, causes that we obtain an additive effect of lower and higher values of sf . This means that we will expect a system response proportional to the number and kind of the SLAs accepted by the system. We can conclude that when receiving jobs with multiple SLAs the system would be inclined to accept jobs with largest slack factors. This causes that the response of the MSL and SSL to be more similar for low numbers of SLAs, when the number of SLAs increases the proportion of SLAs accepted trend to be more homogeneous making that the response of the MSL and SSL diverge.

6.4 Multiple Service Levels - Parallel Machines (MSL-PM) algorithm

In this section we present the results of the metrics for the MSL-PM.

Figure 38 shows the *TPT* for the MSL-PM algorithm. The value shown is relative to the *MSPT*. We see that the number of machines in the system is the variable that more significantly influences the processing time that the system obtains. Additionally, systems with the larger number of machines are the ones which have a greater increment in the *TPT* when more SLAs are added. Finally, for the MSL-PM algorithm only the system with 64 machines obtains *TPT* equal to the maximum processing time (*MSPT* value), this differs from the SSL-PM algorithm where the 32 machines system also reach this value.

Figure 39 shows the *PRJ* for the MSL-PM algorithm. We see that a low number of machines has large percentage of rejections. This value is kept constant regardless of the number of available SLAs. In contrast, systems with more machines present more variation when the number of SLAs changes. For those systems we see three intervals. The first one shows the drop in the *PRJ* from one to two SLAs. The second one, right after the two SLAs, shows the *PRJ* has an upswing trend, and finally, in the third one it declines after the 5-6 SLAs. The decline and the values obtained are less than the ones presented in this region for the SSL-PM algorithm

Figure 40 shows the *MWT* metric for the MSL-PM. We see that the main trend is a decrease of this metric as the number of machines grows and also as the number of SLs decreases. The values are also stabler when we have fewer machines.

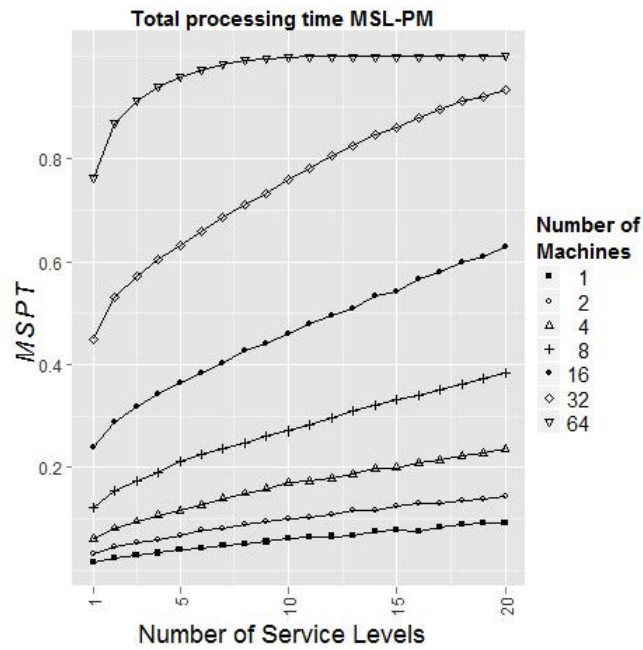


Figure 38. Total processing time for MSL-PM algorithm.

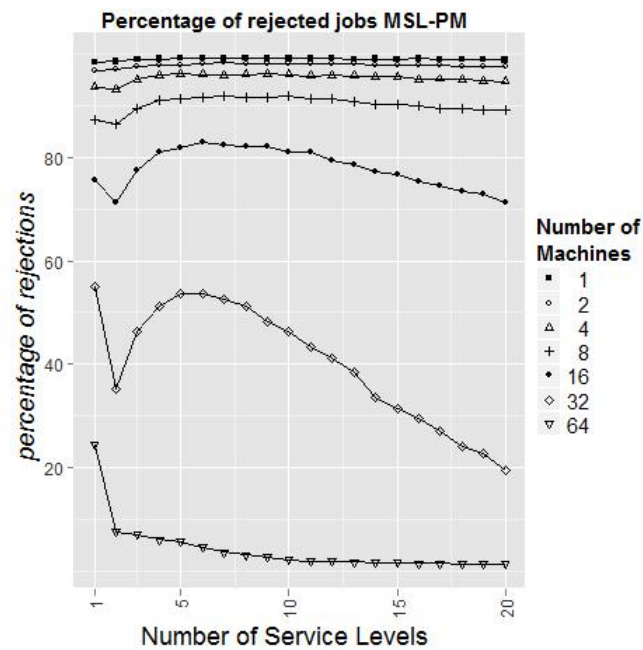


Figure 39. Percentage of rejected jobs for MSL-PM algorithm.

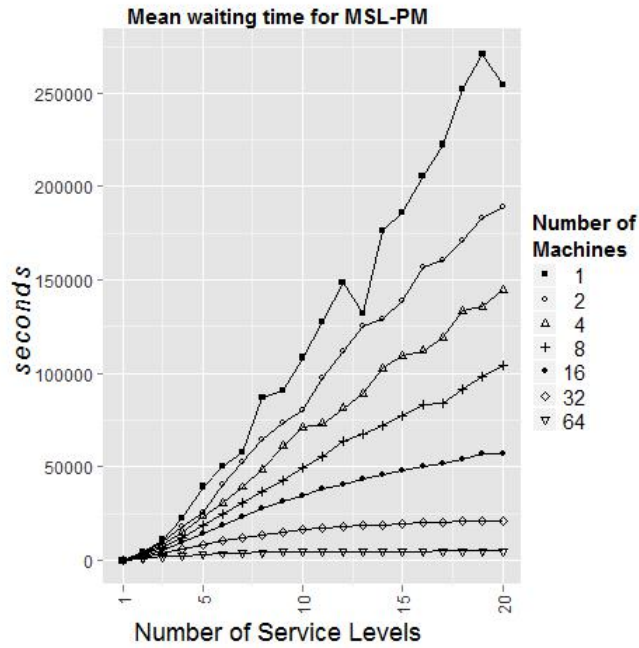


Figure 40. Mean waiting time for MSL-PM algorithm.

Figure 41 shows the *MBSD* for the MSL-PM algorithm. Similar to the *MWT* metric, we see that this metric descends as the number of machines increases and also as the number of SLAs decreases. For these experiments, it is more difficult to determine what slack factors are better for users to submit their jobs. However, we see that the mean value of *MWT* is less than the 50 percent of the maximum slack factor for all machine systems, and the different number of SLAs with the exception when one SLA is used.

In Figure 42 we present the *ME* for the SSL-PM algorithm. We observe that for all points the value of the metric depends more on the number of machines in the system and less on the number of SLAs. With the exception of the single machine system, the value of one SL has the maximum efficiency. Then the value decreases but very slowly until it gets stable. Only the 64 machines system maintains a decline in the machine efficiency for larger number of SLAs.

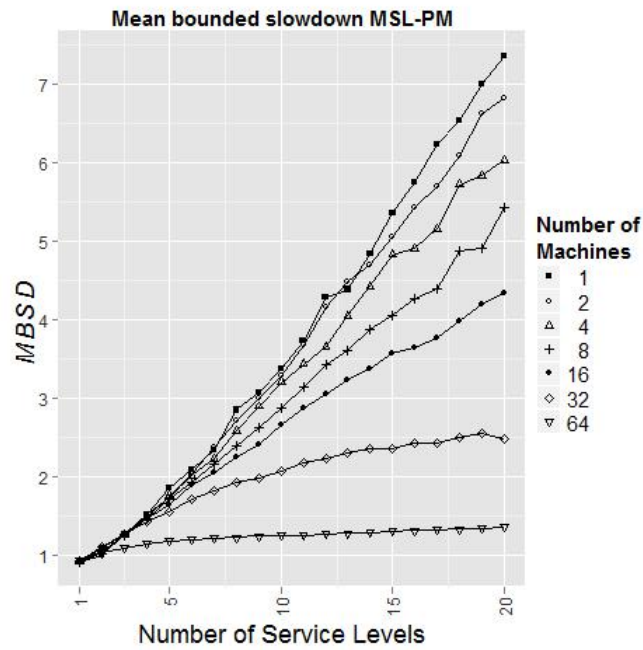


Figure 41. Mean bounded slowdown for MSL-PM algorithm.

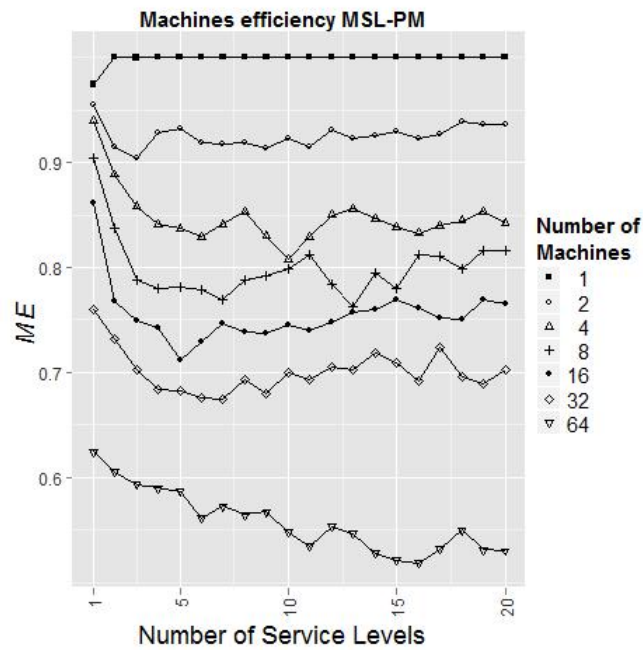


Figure 42. Machines efficiency for MSL-PM algorithm.

Finally, in Figure 43 we present the *MNI* for the SSL-PM algorithm. We see in the graph that all the systems have a similar tendency with the exception of the systems with 32 and 64 machines. The general trend is that starting from zero interruptions when there is only one SLA with $sf = 1$, the *MNI* has its maximum value with two SLs. Then, the metric monotonically decreases, this decrease being proportional to the number of machines. Thus, the higher the number of machines in the system, the lower would be its *MNI* decrease.

In the case of the system with 32 machines we see that for the last number of SLAs, instead of a decrease in the *MNI* its value increases. Finally, the 64 machines system has a completely different trend, from zero interruptions with one SLA the *MNI* value jumps with two SLAs and then it asymptotically increases to a constant value of 0.7 for the remaining number of SLAs. We conclude that the behavior of this system is caused by the reach of the maximum processing time available in the workloads.

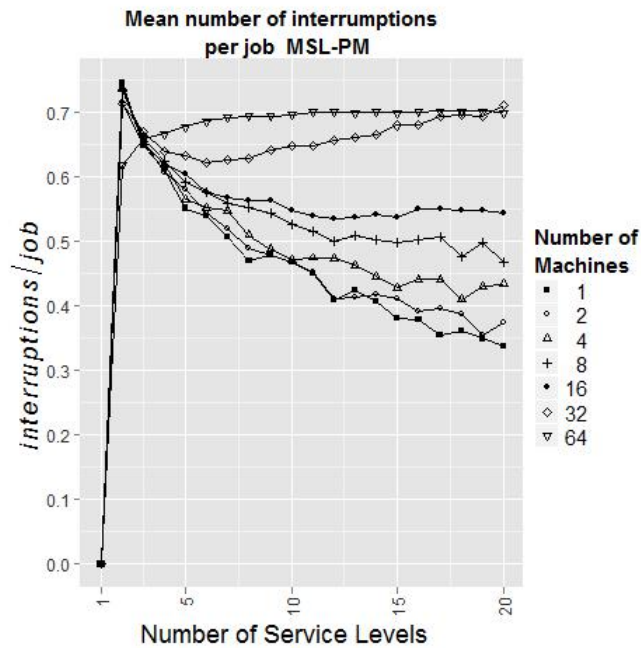


Figure 43. Mean number of interruption per job for MSL-PM algorithm.

In this chapter we have seen the trend of the metrics we took for the different algorithms. The general conclusion is that the slack factor variation has the main effect of allowing more jobs to be accepted by the system. If the system receives jobs with different SLAs, the jobs accepted will be mainly those with the relaxed slack factor. The acceptance of jobs with relaxed slack factor cause that they wait inside the system for a longer time, and allowing the system to accept more jobs. This cause that the system is able to increase the total processing time, while the mean waiting time and bounded slowdown increases.

The parallelization of the system causes an increment of the processing resources resulting in the increment of total processing time and the decrement in the number of rejected jobs. An increment in the resources causes that the scheduling efficiency drops. This means that machines are not all the time being used and there are idle gaps. Finally the increment in the number of interruption as more machines are used tell us that the accepted jobs are more urgent that the one currently being executed.

Chapter 7

Criteria Analysis

7.1 Group of metrics analysis

So far we have presented plots of the metrics for the evaluated algorithms. These plots are very helpful if we want to determine the best algorithm for a single criterion. For example, a system provider who wants to maximize the profit can select the best algorithm as well as its best parameters. However, if the provider wants not only to maximize its economic return but also his customer's satisfaction, the plots presented so far do not give a perspective that can help finding the best algorithm for both metrics. Therefore a multi-criteria analysis of the algorithms is necessary.

For this analysis, we select three groups of metrics classifying them according with their importance to satisfy scheduling goals of a specific actor. Three groups are distinguished: system-centric, user-centric and algorithm-centric. The system-centric metrics group includes those metrics, which provide a competitive advantage to the system against other providers. We consider that the service provider benefit metric and the percentage of rejected jobs are included in this group. On the other hand, the user-centric metric group includes metrics that satisfy user objectives: waiting time and faster response in terms of the mean bounded slowdown. Finally, the algorithm-centric group includes metrics that measure the scheduling algorithm performance in the form of a competitive factor. Table 3 summarizes the metrics considered for these three groups.

Table 3. Group metrics for the criteria analysis.

Groups of metrics	Metrics
System-centric	G and PRJ
User-centric	MWT and $MBSD$
Algorithm-centric	c_V

This criteria analysis is performed based on the degradation technique as explained in Chapter 3. In the Appendix A, we present the degradations plots for all metrics. Bellow, we focus only on the results of the groups, and present the results for the 1, 2, and 4 machines systems.

7.2 Criteria analysis results

The results are presented in the form of plots. Each one includes only three best algorithms for each scenario to easily obtain conclusions.

Considering the system-centric metrics group presented in Figure 44, the best sf has no significant difference with changing the number of machines. The best values are obtained for the four machines system with a $sf = 8$, followed by a $sf = 9$, and a $sf = 7$. The system that follows is the system with two machines that has its best values with a $sf = 9$ followed by a $sf = 8$ and, a $sf = 10$. Finally the worst values were obtained by the system with a single machine with sf values of 9, 10, and 8. Here we can see that from the system perspective having a greater number of machines is beneficial. When choosing the sf values, the system takes more advantage of SLs with sf within the interval of 7 to 10.

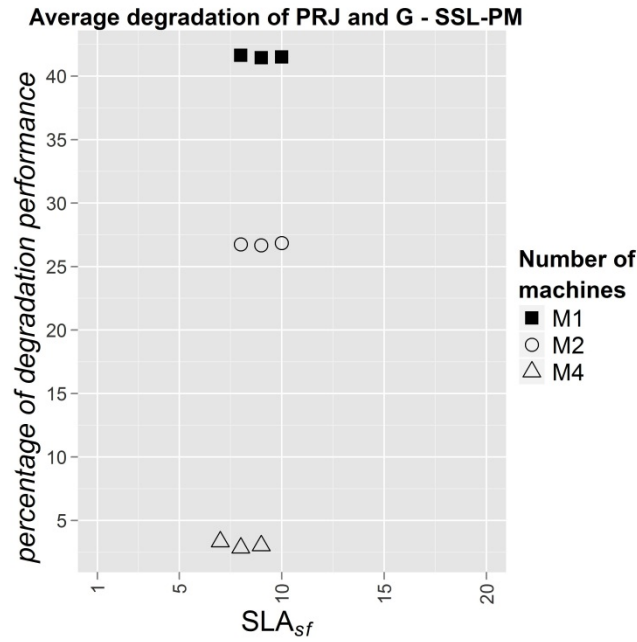


Figure 44. System-centric metric group for the SSL-PM algorithm.

Figure 45 shows the user-centric metrics group for the SSL-PM algorithm. We can see that an increment in the number of machines is also beneficial to the user because the systems with larger number of machines are the ones that obtain lower degradation values. We can see that for the three machine configuration, sf equals to one, two and three provides less degradation values. It means that for the user its always better to select the smallest sf .

Figure 46 shows the algorithm-centric metric group for the SSL-PM algorithm. Contrary to the other groups, we see that an increment in the number of machines is not beneficial. Here, sf 's that have the top places are the ones with values of one, two and three.

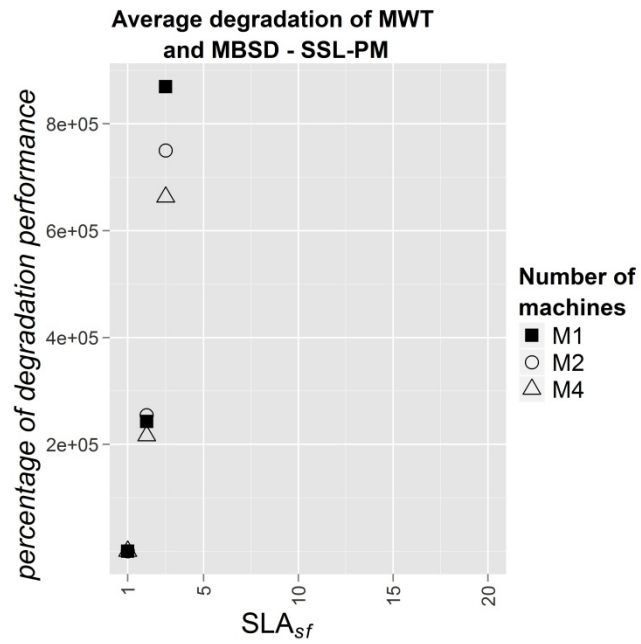


Figure 45. User-centric metric group for the SSL-PM algorithm.

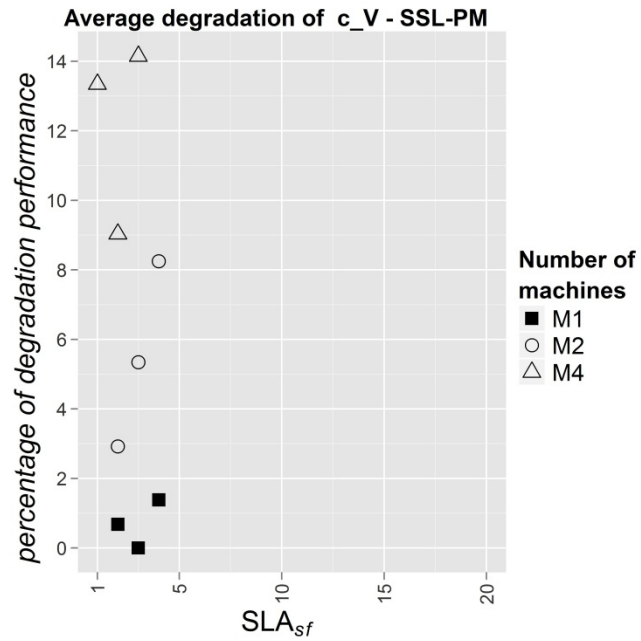


Figure 46. Algorithm-centric metric group for the SSL-PM algorithm.

We now present the plots for the multi-criteria analysis of the MSL algorithms. Once more, we present the three groups of metrics and a description of the results.

Figure 47 shows the multi-criteria results for the system-centric metrics group. We see the system with one machine that corresponds to a MSL-SM system has its best value with 14 SLAs, and its second and third value on 11 and 10 respectively, when the number of machines that the system has increases, which corresponds to the MSL-MM algorithm, we see that the best values are in the middle region and as the number of machines increases the values are kept in a middle number of SLAs.

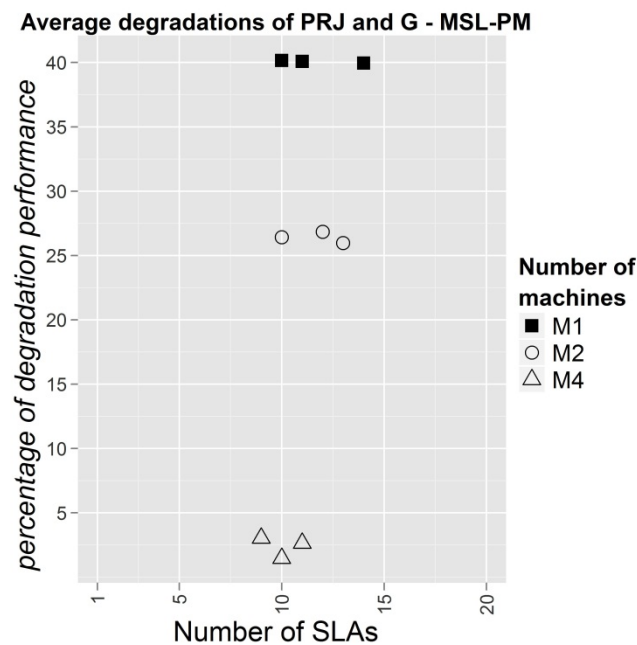


Figure 47. System-centric metric group for the MSL-PM algorithm.

Figure 48 shows the results for the user-centric metric group now for the MSL-PM algorithm. We see that increasing the number of machines allows the system to provide the user with a better response. Also, as in the case with the SSL algorithm, the best number of SLAs is where there is a small number which have low sf .

Figure 49 shows the degradation analysis for the algorithm-centric metrics group. Here we see that the values with less degradation are the ones with the lowest

number of SLAs. Additionally, here we can cleaner see that increasing the number of machines causes greater increments in the degradation value, therefore, the algorithm does not benefit of the increment in the system resources and obtains a worse performance comparing with the upper bound of the optimal algorithm.

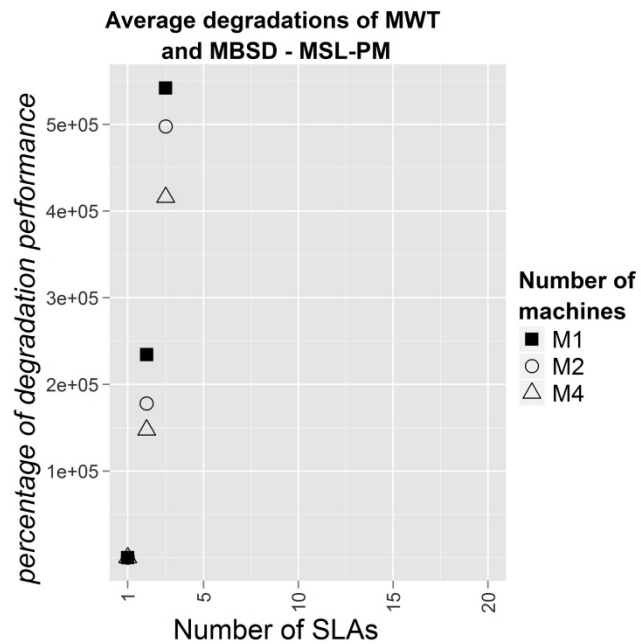


Figure 48. User-centric metric group for the MSL-PM algorithm.

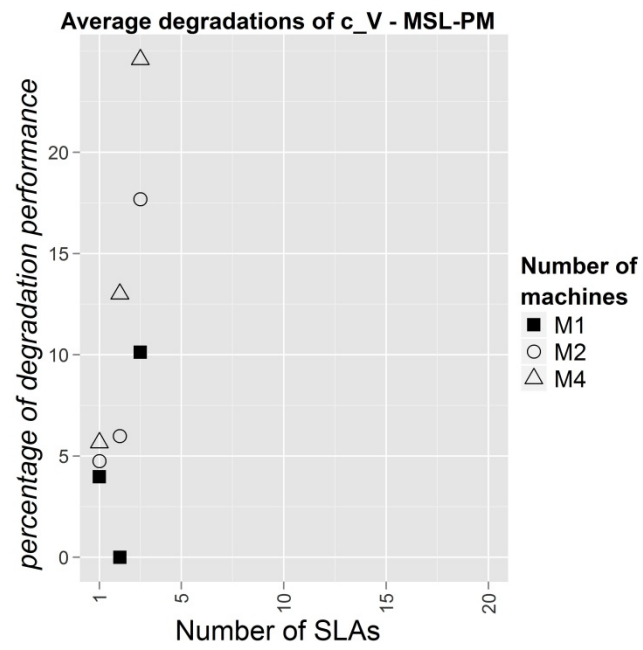


Figure 49. Algorithm-centric metric group for the MSL-PM algorithm.

Conclusions

In this chapter, we present the conclusion of this thesis and the future work.

In recent years, the vision of computing as a utility is emerging. The elasticity of a cloud and the utility model match the need of users providing services that tailor necessities of each one. IaaS providers require mechanisms with QoS-driven scheduling. New IaaS models and scheduling algorithms as the one we present in this thesis contribute with ideas on how this utility computing could be taken to HPC users. We consider that this model takes a step further because it transfers the QoS of the service from the user to the provider. Thus the provider can perform a better management of the resources improving the efficiency which in turn could lower the prices, be ecological friendlier, among others advantages.

To the best of our knowledge, this work is the first in the literature that simulates IaaS clouds with different SLs on single and parallel machines. This is also the first work that analyzes the costs of execution, which is not covered by studies, made for real time systems.

The theoretical foundation of this work is based on one metric, the competitive factor, which may not be enough for a more complete evaluation of the algorithms. This work provides an extensive experimental analysis considering a variety of metrics.

We found for the competitive factor two experimental minimum bounds that consider unlimited and limited resources systems models. The limited resources bound have a greater approximation to the real competitive factor in low values of slack factor, being it even tighter than the theoretical bound, however its performance degrades as the sf increases. In the case of the unlimited resources its performance is bad in the whole interval. Nevertheless, both bounds converge

to the optimal value when the system has enough resources to execute the whole workload. The existence of values where the theoretical bound of the competitive factor is tighter than the experimental bounds obtained gives room to find tighter bounds that can represent in a better way the performance of the system.

With the results of the SSL-SM algorithm we conclude that the slack factor is a parameter that effectively allows the acceptance of more jobs, leading to the increment of processing time but causing that jobs need to wait longer. Low values of slack factor are the ones where our competitive factor minimum bound for limited resources obtains a better approximation to the actual competitive factor, even better than the theoretical bound.

From the experiments of the SSL-PM algorithm we conclude that this algorithm does not scale well in terms of processing time when we add more machines, this is mainly caused by the diminution in the scheduling efficiency. This disadvantage can be overcome in a real cloud environment by dynamically assigning resources to cover the demand of the workload and force the algorithm to work in optimal conditions. We showed how with enough resources we can obtain a competitive factor very close to one, by executing nearly all jobs in the workload. However, the dynamic provision of resources is necessary to avoid misusing resources that may cause a drop in the system's benefit.

The results for the MSL-SM show that the distribution of the different SLAs in the set of SLAs accepted by the system is the major factor that defines the response of the system. With the distribution and the sets of SLAs used, the response is attenuated by the addition of SLAs with lower slack factors.

The MSL-PM results were the ones where we can conclude less, since its response is a combination of the SSL-PM with the MSL-SM. That means a parallel machine response attenuated due to the effect of the SLAs with lower slack factor.

Incorporating additional metrics has allowed enriching the analysis of the algorithms defining different metric groups. We can conclude from this analysis that only for the user, the best values in terms of sf and number of SLAs correspond to the maximum values of the experimental competitive factor. The reason is that the algorithm provides a better quality of service when the sf is small and also is when the algorithm response is closest to the optimal. The response of the algorithms to system goals depends on the price function. Because of the chosen price function maximizes the gain for sf with middle values of sf , these sf perform better from the system perspective. The parallelism introduced to the SSL and MSL algorithms is beneficial for the system and the user. However, when algorithms are compared with an optimal income bound, we see that this parallelism degrades their response. Theoretically, this degradation should not exist but the experimental lower bounds we define, does not allow us to have a tighter approximation that can reinforce this fact.

Another characteristic of our work that previously was not considered is the confidence of the system in the form of percentage of rejected jobs. This analysis let us consider an execution environment where the system is not only interested to increment the benefit but also the confidence that is providing to the user. We show that the number of machines in the system is the major factor for determining the confidence that the user receives by the system.

Thanks to this analysis, we see the importance of the characteristics of the workload for the evaluation of the algorithms. Even further, a real implementation of these algorithms should consider an appropriate workload for the system. In a real environment, considering beforehand these parameters is impossible, hence a dynamic adjustment of the system characteristics (number of machines available, SLAs offered, etc.) becomes necessary to keep the system operating efficiently.

This work can be expanded in several ways, variations of the characteristics of the MSL models, the distribution of SLAs, price function. Additionally, the model can be enriched providing elements such negotiation mechanism, dynamic price models and dynamic resource adjustment. These elements would provide a model closer to be implemented.

Bibliographic references

- Andrieux, A., Czajkowski, K., Dan, A., et al.(2004). Web services agreement specification (WS-Agreement). *Global Grid Forum*.
- Baruah, S.K., Haritsa, J.R.(1997). Scheduling for overload in real-time systems. *IEEE Transactions on Computers*, 46(9), 1034–1039.
- Bhardwaj, S., Jain, L.(2010). Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and Information Technology*, 2(1), 60–63.
- Buyya, R., Abramson, D., Giddy, J., & Stockinger, H.(2002). Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15), 1507–1542.
- Buyya, R., Yeo, C. S., & Venugopal, S.(2008). Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. *10th IEEE International Conference on High Performance Computing and Communications*, IEEE Comput. Soc, 5–13.
- Cosmin Silaghi, G., Dan Șerban, L., & Marius Litan, C.(2010). A Framework for Building Intelligent SLA Negotiation Strategies under Time Constraints, In: J. Altmann & O. Rana (eds.). *Economics of Grids, Clouds, Systems, and Services*, Berlin, Springer, 48–61.
- Feitelson, D.(2008). Parallel Workloads Archive. *Accessible on the project web-site, <http://www.cs.huji.ac.il/labs/parallel/workload/>*.
- Feitelson, D., Rudolph, L., Schwiegelshohn, U., Sevcik, Kenneth C., & Wong, P.(1997). Theory and practice in parallel job scheduling. *Job Scheduling Strategies for Parallel Processing*, 1291, 1–34.
- Garg, S., Gopalaiyengar, S.(2011). SLA-based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter. *Algorithms and Architectures for Parallel Processing*, 371–384.
- Gupta, B. D., Palis, M. A.(2001). Online real-time preemptive scheduling of jobs with deadlines on multiple machines. *Journal of Scheduling*, 4(6), 297–312.
- Hirales Carbajal, A., Tchernykh, A., Roblitz, T., & Yahyapour, R.(2010). A Grid simulation framework to study advance scheduling strategies for complex workflow

applications. *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, IEEE Comput. Soc, 1–8.

IBM.(2010). Review and summary of cloud service level agreements. *Accessible on IBM's web-site, <http://public.dhe.ibm.com/software/dw/cloud/library/cl-rev2sla-pdf.pdf>*, 1–10.

Iosup, A., Li, H., Jan, M., et al.(2008). The Grid Workloads Archive. *Future Generation Computer Systems*, 24(7), 672–686.

Islam, M., Balaji, P., Sadayappan, P., & Panda, D. K.(2003). QoPS : A QoS based scheme for Parallel Job Scheduling, *Job Scheduling Strategies for Parallel Processing*, Berlin, Springer, 252–268.

Macías, M., Smith, G., Rana, O., Guitart, J., & Torres, J.(2010). Enforcing Service Level Agreements Using an Economically Enhanced Resource Manager, *Economic Models and Algorithms for Distributed Systems*, Birkhäuser Basel, 109–127.

Mell, P., Grance, T.(2011). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *National Institute of Standards and Technology*, NIST SP(800-145), 7.

Patel, P., Ranabahu, A., & Sheth, A.(2009). Service Level Agreement in cloud computing. *Cloud Workshops at OOPSLA*, ACM Press, 10.

Ramírez Alcaraz, J. M., Tchernykh, A., Yahyapour, R., et al.(2011). Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids. *Journal of Grid Computing*, 9(1), 95–116.

Sakellariou, R., Yarmolenko, V.(2008). Job Scheduling on the Grid : Towards SLA-Based Scheduling. *Advances in Parallel Computing series*, 16, 207–222.

Schwiegelshohn, U., Tchernykh, A.(2012). Online Scheduling for Cloud Computing and Different Service Levels. *26th IEEE International Parallel & Distributed Processing Symposium*, 1061–1068.

Tsafrir, D., Etsion, Y., & Feitelson, D. G.(2007). Backfilling Using System-Generated Predictions Rather than User Runtime Estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6), 789–803.

Wu, L., Kumar Garg, S., & Buyya, R.(2011). SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. *Cluster, Cloud*

and Grid Computing (CCGrid), 11th IEEE/ACM International Symposium on, IEEE Comput. Soc, 195–204.

Appendix

A.1 Degradations metrics plots

Multiple Machine - Single Service Level

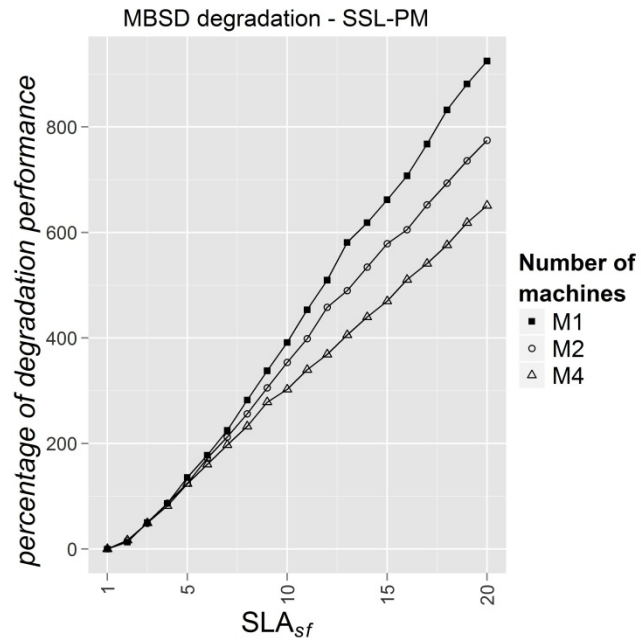


Figure 50. Mean bounded slowdown degradations for SSL-PM algorithm.

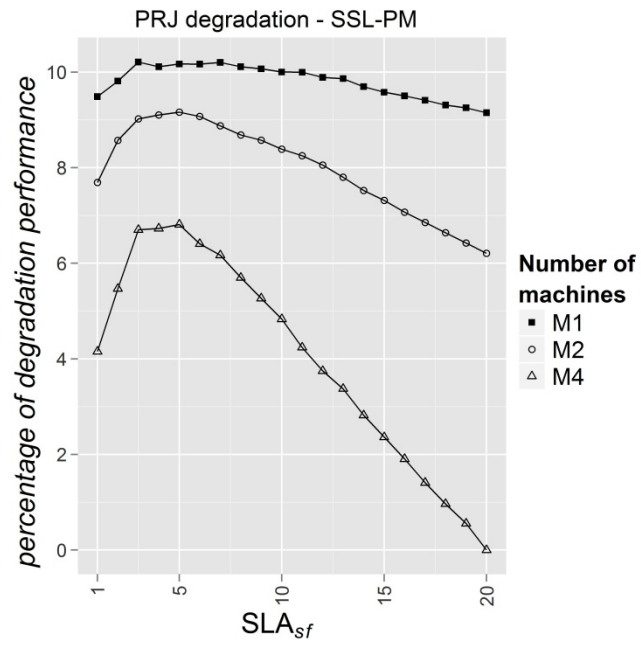


Figure 51. Percentage of rejected jobs degradations for SSL-PM algorithm.

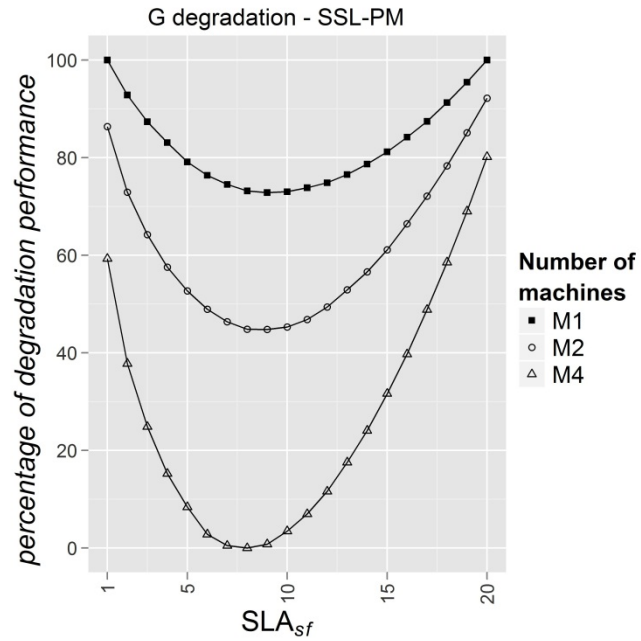


Figure 52. Service provider benefits degradations for SSL-PM algorithm.

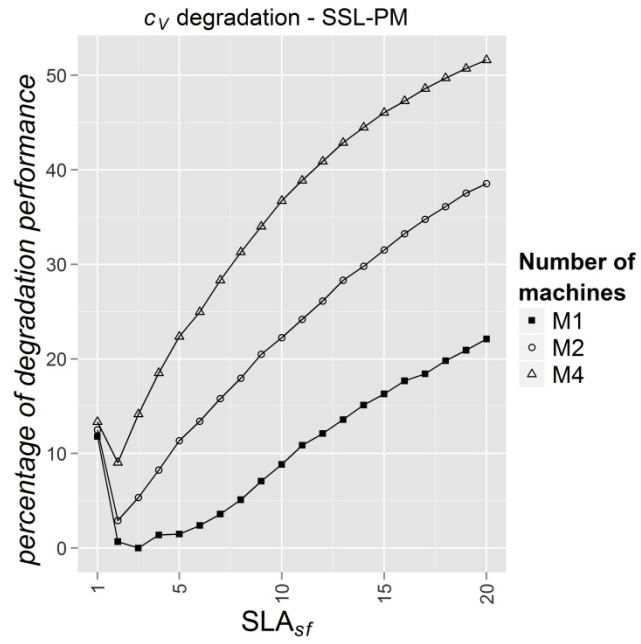


Figure 53. Competitive factor degradations for SSL-PM algorithm.

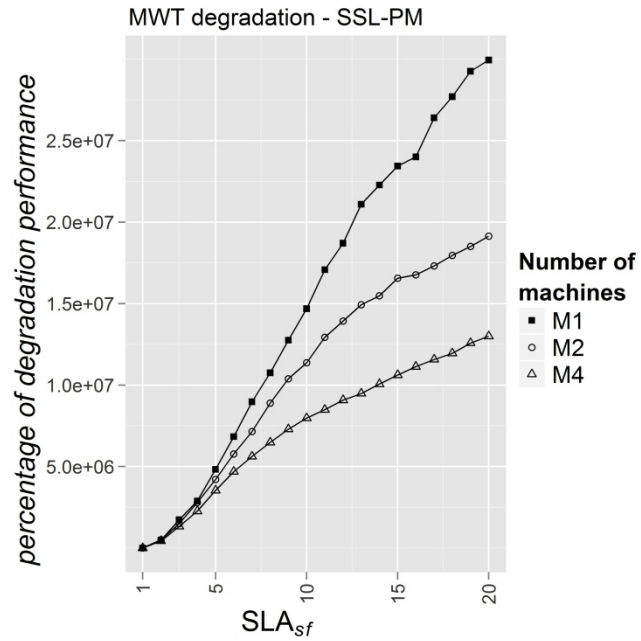


Figure 54. Mean waiting time degradation for SSL-PM algorithm.

Multiple Machine – Multiple Service Level

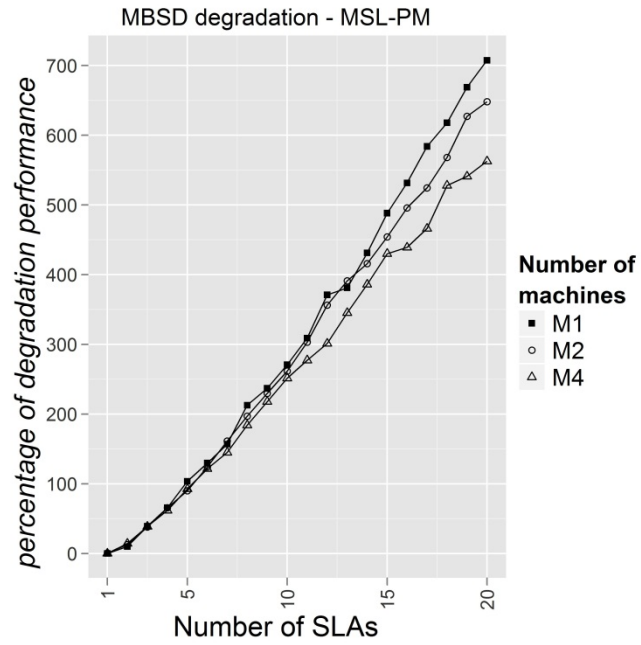


Figure 55. Mean bounded slowdown degradations for MSL-PM algorithm.

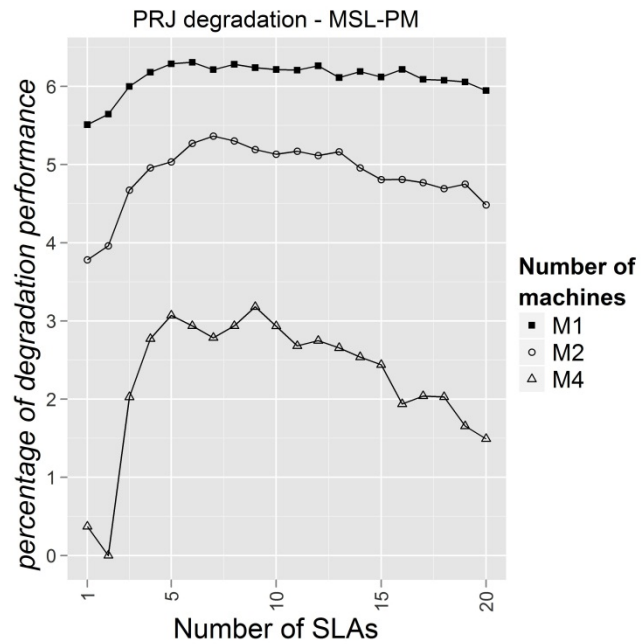


Figure 56. Percentage of rejected jobs degradations for MSL-PM algorithm.

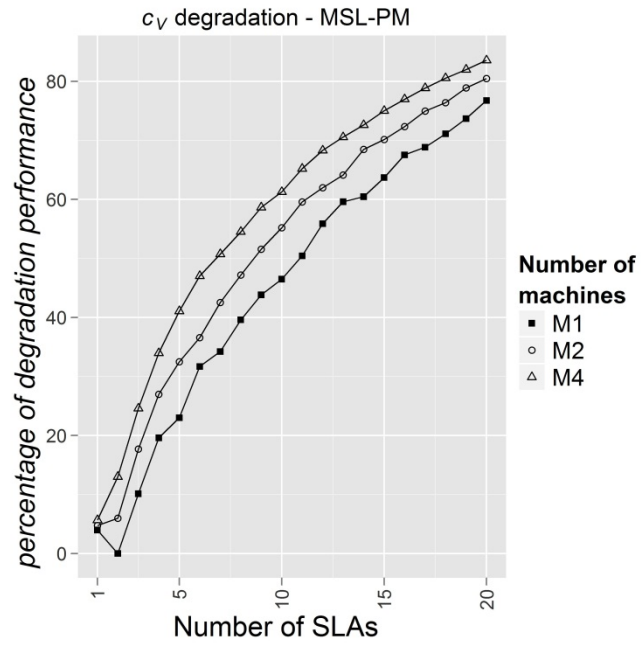


Figure 57. Competitive factor degradations for MSL-PM algorithm.

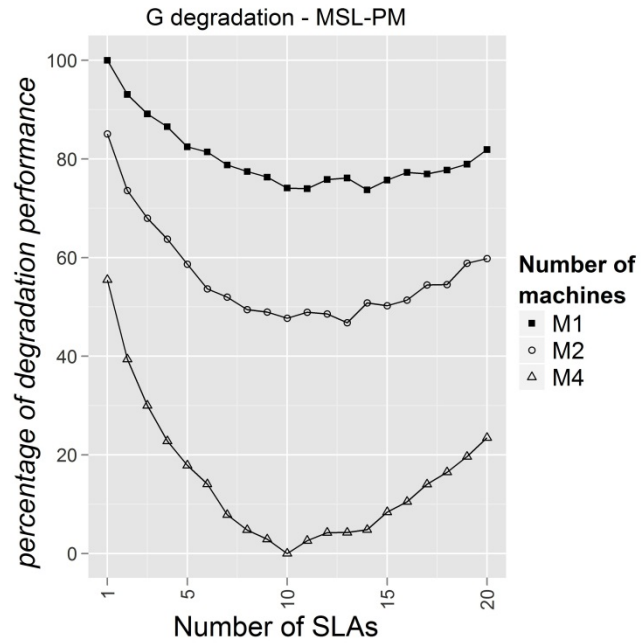


Figure 58. Service provider benefits degradations for MSL-PM algorithm.

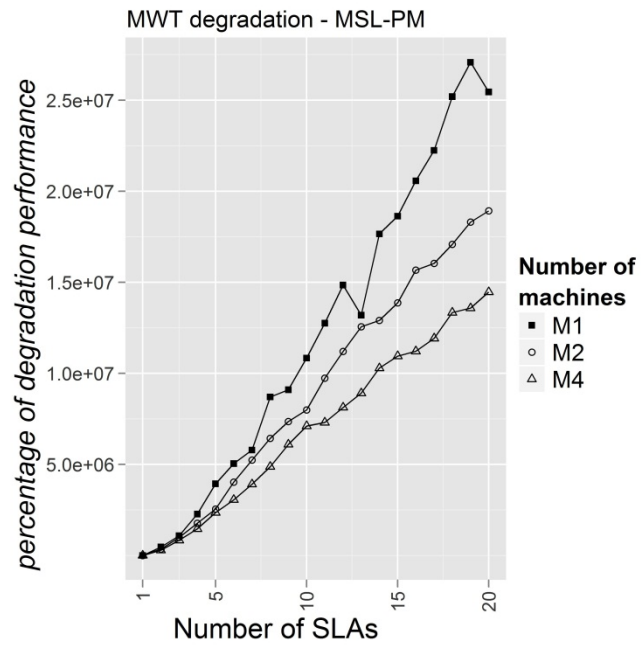


Figure 59. Mean waiting time degradation for MSL-PM algorithm.