A FRAMEWORK FOR DYNAMIC SCHEDULING BASED ON QUALITY OF SERVICE METRICS

By

Wilson Ernesto Lozano Rolón

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

July 10, 2006

Approved by:

Fernando Vega. Commitee, Ph.D Member, Graduate Committee	Date
Nayda Santiago. Commitee, Ph.D Member, Graduate Committee	Date
Wilson Rivera. Commitee, Ph.D President, Graduate Committee	Date
Yolanda Ruiz-Vargas, Ph.D Representative of Graduate Studies	Date
Isidoro Couvertier, Ph.D Director of the Department	Date

Abstract of Dissertation Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

A FRAMEWORK FOR DYNAMIC SCHEDULING BASED ON QUALITY OF SERVICE METRICS

By

Wilson Ernesto Lozano Rolón

July 10 2006

Chair: Wilson Rivera
Major Department: Electrical and Computer Engineering

In this thesis the scheduling process in production environments is improved through the development of a framework that implements a distributed dynamic scheduling methodology based on quality of service. Such methodology takes into account contingency, priority fluctuations and incorporates a scheduling algorithm referred to as Quality of Service-based Maximum Urgency First (QB-MUF) algorithm. The QB-MUF algorithm gives high priority to jobs with low probability of failing according to suitable failure probabilities for particular application environments. The contribution of this research is related to the use of quality of service metrics, calculated from the job meta-data, as part of the information used in the decision making process of the scheduling.

To validate the dynamic scheduling framework, two study cases are considered. First a simplified model of the digital publishing workflow is build upon the framework in order to observe the behavior of the proposed scheduling strategy. Second, a grid environment where resources are connected via two-level hierarchical networks is simulated. In this case, the first level is a wide area network connecting local area

networks at the second level. The implemented model represents the composition of two, wide-area distributed, image operators providing treatment of data images.

Experimental results, show that the QB-MUF algorithm outperforms traditional scheduling strategies such as the Minimum Laxity First and the First In First Out algorithms.

Resumen de Disertación Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

UN FRAMEWORK PARA PLANIFICACIÓN DINÁMICA BASADO EN METRICAS DE CALIDAD DE SERVICIO

Por

Wilson Ernesto Lozano Rolón

Julio 10, 2006

Consejero: Wilson Rivera
Departamento: Ingeniería Eléctrica y Computadoras

En esta tesis el proceso de planificación en entornos de producción es mejorado a través del desarrollo de un sistema que implementa una metodología dinámica de planificación basada en calidad de servicio. Dicha metodología toma en cuenta contingencias, fluctuaciones de prioridad e incorpora un algoritmo de planificación nombrado como "Quality of Service-based Maximum Urgency First (QB-MUF)". El algoritmo "QB-MUF" da mayor prioridad a aquellos trabajos con baja probabilidad de fallar de acuerdo a algunas probabilidades de falla que se adecuan para cada problema en particular.

La contribución de esta investigación esta relacionado con el uso de métricas para calidad del servicio, las cuales son calculadas a partir de de meta-datos extraídos del trabajo, como parte de la información usada en la toma de decisiones durante el proceso de planificación.

Para validar el sistema de planificación dinámica, dos casos de estúdio se consideraron. Primero, un modelo simplificado de flujo de trabajo en impresión digital se construye sobre el sistema para observar el comportamiento de la estrategia de

planificación propuesta. Segundo, se simula un ambiente de computación en malla donde los recursos están conectados por redes separadas jerárquicamente en dos niveles. En este caso, el primer nivel corresponde a una red de área amplia (WAN por su nombre en inglés) conectando redes de área local (LAN por su nombre en inglés) como segundo nivel. El modelo implementado representa la composición de dos operadores de imágenes, distribuidos en la red WAN, proveyendo tratamiento para datos de imágenes.

Resultados experimentales, muestran que el algoritmo "QB-MUF" supera otros algoritmos tradicionales como son "mínima laxitud primero" y "primero en entrar - primero en salir".

Dedicated to:

My Mother and my Father

My brother and sister

ACKNOWLEDGMENTS

I would like to tank professor Wilson Rivera, for giving me the opportunity to work in his research group.

Thanks to the people who walked on my side in this journey, my PDC lab partners, the ADMG people and to my friends.

The work in this thesis was partially supported by a grant from the Imaging and Printing Group (IPG) of Hewlett-Packard.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

APM	Automated Preflight Model
EDF	Earliest Deadline First.
DS	Dynamic Scheduling.
MUF	Maximum Urgency First Algorithm.
QOS	Quality of Service
QB-MUF	Quality of Service Based - Maximum Urgency First Algorithm.

LIST OF SYMBOLS

t	Time (seconds)
τ	Time between events.

CHAPTER 1
INTRODUCTION

1.1 Overview

Scheduling deals with the allocation of resources to tasks over time [1]. Tasks and resources can take different forms depending on the specific problem domain. For example, in a simplified model of a heterogeneous cluster of computers, the set of computers can be considered as a set of available resources $R = \{r_1,\ r_2, ..., r_m\}$. Each autonomous computer r_j can be weighted according to parameters, such as mean time to perform a unit of computation. In this model, the set of jobs to be executed, can be represented by a set of tasks, $\overline{\alpha} = \{\alpha_1,\ \alpha_2, ...\alpha_n\}$, which are independent of each other.

The scheduling problem can be formulated as the following optimization problem:

$$Minimize \sum_{j=1}^{n} w_j C_j \ , \tag{1.1}$$

for a given set of tasks $\{\alpha_j\}_{j=1}^{n}$, where w_j is a weighting factor and C_j is a metrics for each task $\alpha_j, 1 \leq j \leq n$. The meaning of the objective function can be changed according to the specific problem domain. For example, the optimization problem can be the minimization of the aggregated completion time of a number of jobs, each pondered with a priority factor.

There exists a variety of scheduling models, which adopt both deterministic and non-deterministic formulations. These models include single machine[2, 3], parallel machine[4, 5], and shop scheduling models[6–10]. The single machine model is the

simplest type of scheduling model and is a particular case among all other environments. It is often found in practice when there is only one service point or a single stage. Algorithms developed for single machine models provide a basis for design of exact algorithms and heuristics for more complicated machine environments. Basic single machine models, with regular objective functions, are relatively simple and solvable via simple priority rules. More advanced single machine models deal with non-regular and/or multiple objective functions. These models are either solvable in polynomial time through dynamic programming or using polynomial time approximation schemes. A generalization of the class of single machine models is the type of parallel machine models. In parallel machine models, a job requires a simple operation and may be processed on any of the m machines or on anyone that belongs to a given subset of machines. The class of shop scheduling models comprises the open shop, flow shop and job shop models. In an open shop model, the operations of a job can be performed in any order. In a job shop, they must be processed in a specific job-dependent order. A flow shop is a special case of job shop in which each job has exactly m operations, one per machine, and the order in which they must be processed is the same for all of the jobs. Shop models are strongly NP-hard[1] in their most general form. For the flow shop model, the case when there are more than two machines is strongly NP-hard, although the two machine version is polynomial solvable.

Production environments, such as print shops and grid computing, are subject to many sources of uncertainty. Stochastic models[11, 12] have been proposed to

[1] In computational complexity theory, NP-hard (Non-deterministic Polynomial-time hard) refers to the class of decision problems that contains all problems H, such that for every decision problem L in NP there exists a polynomial-time many-one reduction to H, written $L \leq_p H$. Informally, this class can be described as containing the decision problems that are at least as hard as any problem in NP.

model random features, such as job processing times, by specifying their probability distributions. Stochastic scheduling models, especially with exponential processing time, often contain more structure than their deterministic counterparts. Consequently, models that are NP-hard in a deterministic setting often allow a simple priority policy to be optimal in a stochastic setting. In production environments, scheduling is not merely an activity that ensures on-time delivery, but a scientific tool that ultimately impacts their profitability, customer satisfaction and competitiveness. As a result, a major driving force for this research is the need to incorporating dynamism into the scheduling process for such environments.

1.2 Problem Statement

Dynamic scheduling has the potential to relax some of the constraints imposed by strict static scheduling approaches [13, 14]. However, most approaches to dynamic scheduling are based on time criteria. Furthermore, those existing approaches targeting quality of service are based on external requirements of the jobs. This thesis deals with the problem of adding other dynamic factors, such as the quality of service extracted from the internal job metadata, in the decision making process.

1.3 Solution Approach

From our point of view, the scheduling process in production environments, may be improved through a new approach based on metrics targeting quality of service. In order to achieve such a scheduling methodology we have concentrated our research efforts on developing a scheduling algorithm that takes into account contingency (unexpected events) and priority fluctuations (changes in job priorities). Such a scheduling algorithm is a modification of the Maximum Urgent First[15] (MUF) algorithm. The new algorithm, referred to as Quality of Service Based - Maximum Urgency First (QB-MUF), gives high priority to jobs with low probability of failing

according to criteria defined in each workflow stage and allows diverse scheduling policies.

1.4 Research Objectives

The main goal of this research is to develop a framework that implements a distributed dynamic scheduling methodology based on quality of service. The specific research objectives are listed below:

- To define and implement a dynamic scheduling framework that incorporates management of contingency and priority fluctuations.
- To validate the dynamic scheduling framework in two study cases: digital publishing and grid computing.

1.5 Contributions

Most of the existing dynamic scheduling methodologies are time based and use external requirements of jobs as input for the decision making process. Algorithms based on a laxity factor, such as MUF, have shown good schedules in the time domain. However, normally these kinds of algorithm may deliver, with the same probability, jobs considered to be successful or unsuccessful at the end of a workflow. The contribution of this research is related to the use of quality of service metrics, calculated from the job meta-data, as part of the information used in the decision making process of the scheduling.

The main contributions of this research thesis can be summarized as follows:

- The mapping of job meta-data into quality of service metrics to be considered in the decision making process of the workflow scheduling.
- The modification of the existing Maximum Urgency First algorithm by the inclusion of a second dynamic factor to calculate the urgency criteria.

- The use of a distributed architecture together with a MUF based algorithm.

The importance of this new approach lies in an improved scheduling, because of the high dynamic priority given to those jobs with the best expectation of success at the end of the workflow.

1.6 Thesis Structure

The rest of this thesis is organized as follows: In chapter 2, related work previously done in the area is discussed. Chapter 3 describes in detail the proposed dynamic scheduling framework. The description includes the framework components, the QB-MUF algorithm, and the implementation issues. Chapters 4 and 5, present examples of the application of the dynamic scheduling framework on digital publishing and grid environments, respectively. A summary of results, conclusions and future work is presented in chapter 6. Finally a guideline for the framework software is presented in the appendix A.

CHAPTER 2
PRELIMINARY CONCEPTS AND RELATED WORK

This chapter describes the concepts concerned with our research including scheduling methodologies and the application to *digital publishing* and *grid computing*. Along with the presentation of the related work, a comparative description of our proposed scheduling strategy will be presented. This chapter also includes the description of the simulation tools used to implement the experiments during the different stages of the research. These simulation tools are *Gridsim*[16] and *SimJava*[17].

2.1 Scheduling

A scheduling problem is represented as a triplet $(A \mid P \mid F)$, where A describes the machine environment, P provides details of processing characteristics, and F defines the objective function to be minimized[1]. A case of interest is represented by $(Rm \mid Prec,\ S_{ijk} \mid F_{metric})$, where tasks exchange data with each other. The machine environment Rm consists of m parallel machines where machine i can process task j at speed v_{ij}, which means that the speed of the machine depends on the task to be undertaken. Precedence constraints ($Prec$) may require that one or more tasks be completed before another task is allowed to start its processing. The S_{ijk} represents the sequence dependent upon the set up time between tasks j and k on the machine i. Finally F_{metric} is an objective function that may be defined as a quality of service (QoS) metric.

Scheduling problems may appear under various sets of assumptions: off-line deterministic scheduling, stochastic scheduling, online scheduling, and stochastic online scheduling. In *off-line deterministic scheduling*, all information is known a priori including the number of jobs, release times, due dates, and weights. In *stochastic scheduling* the number of jobs is fixed and known in advance. The processing time of a job is drawn from a given probability distribution. Thus, actual processing times only become known when the processing is completed. Different jobs may have varying processing time distributions. Release dates and due dates may also be random variables from known distributions. In *online scheduling*, even less information is known ahead of time. When a job enters the system, its processing time is a random variable from an unknown probability distribution. Finally, in *stochastic online scheduling*, the type of the distribution of the processing time is known in advance. However, the values of the parameters of the distribution for each processing time are not known a priori.

The criteria to perform scheduling varies according to the performance goals. System schedulers (*high throughput schedulers*) promote the performance of the system over the performance of individual applications. Job schedulers and resource schedulers, for example, optimize the number of jobs executed by the system and the resource utilization of the system, respectively. On the other hand, application schedulers (*high-performance schedulers*) promote the performance of individual applications by optimizing application-centric cost measures. It is unrealistic to expect system schedulers to optimize application performance. Figure 2–1 illustrates a scheduling taxonomy in which both high throughput schedulers and high performance schedulers can be categorized as global or local, homogeneous or heterogeneous, static or dynamic. Scheduling developments have been more focused on static scheduling where execution, data dependency and resources must be determined prior to job launch and cannot be changed. Dynamic scheduling allows

modifications in response to loss of data, availability of resources or identification of new features in input/output. Adaptive scheduling represents a higher level of complexity because in addition to respond to environment fluctuations, it allows scheduling components to interact with each other to optimize scheduling results.

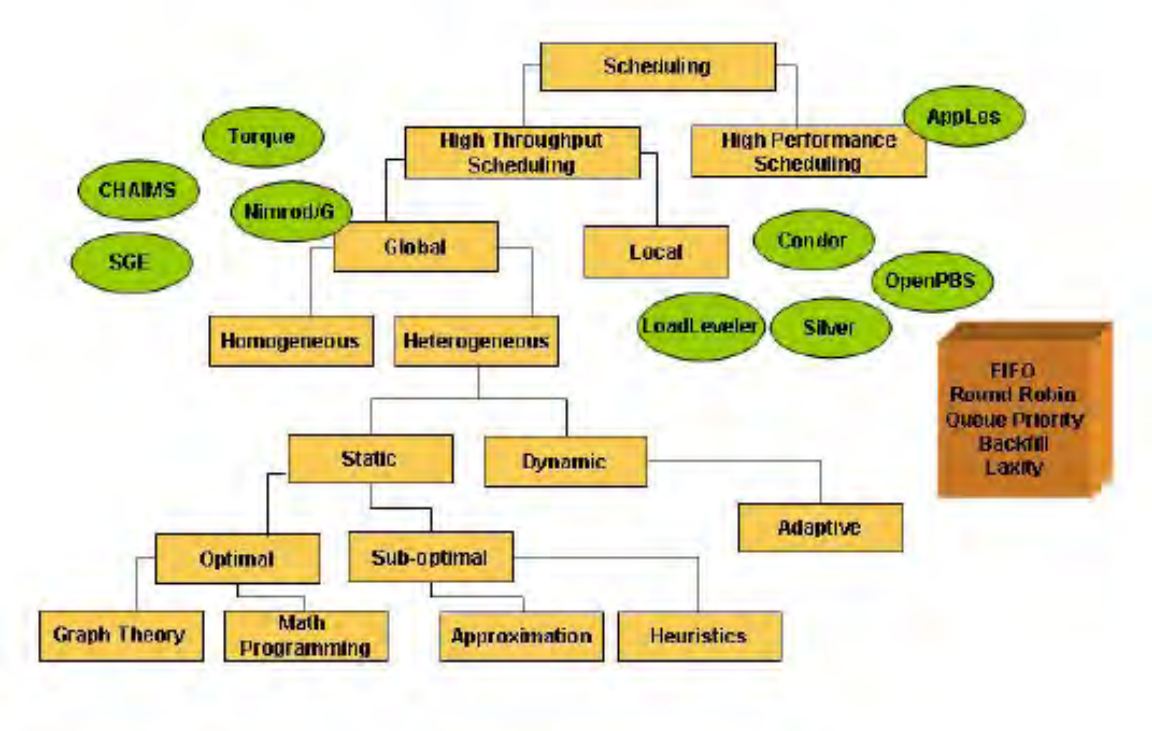

Figure 2–1: Scheduling Taxonomy and Tools

2.2 Scheduling on Digital Publishing

A number of commercial digital printing products provide job tracking and scheduling capabilities including Production Manager from Hewlett-Packard, Lean Document Production (LDP) from Xerox, Print Shop Pro Manager from EDU, Pinnacle from Parsec, and Electronic Planning Board (EPB) from Pace Systems Group, among many others. To the best of our knowledge the only product that provides truly dynamic scheduling capabilities is PrintFlow from EFI. PrintFlow was developed around the Theory of Constraints (ToC) [18] and was adapted to fit

the printing industry. It defines printing as a manufacturing operation comprising interdependent links where only a few constraints control the throughput, on-time delivery and the cost of the entire printing operation. We believe our approach, which diverges from the ToC approach, provides a more realistic scenario since it considers workflow priority fluctuations and contingency in a simplified formulation.

2.3 Scheduling on Grids

Although scheduling has been studied in various contexts, with the emergence of grid computing, unique challenges have arisen. Grids are shared infrastructures with no central control, where the applications compete for the best quality of service from remote resources. In addition, grids exhibit fluctuations in the availability of resources and communication latencies over multiple resource administrative domains. The emerging grid technology [19, 20] has led to the need of a new generation of applications capable of adapting its execution to changing conditions. Therefore, the development of an adaptive application scheduler has become a major challenge [21–24]. Research projects, such as AppLeS [25] and Nimrod/G [26], have demonstrated that periodic evaluation of the schedule in order to adapt it to changing Grid conditions and application dynamic demands can result in significant improvements in performance. The Application Level Scheduling (AppLeS) project primarily focuses on developing scheduling agents for individual applications. Thus, the AppLeS framework contains templates that can be applied to problems that are structurally similar and have the same computational mode. Templates have been developed for parameter sweep [27] and master/slave [28] type applications. Nimrod/G is a Grid resource broker that provides support for formulation of parameter studies on computational grids as well as facilities for resource discovery and scheduling. Prophet [29] is an automated scheduler for data parallel applications that utilizes a

performance model for predicting application performance on different resource combinations. Gallop [30] is a wide-area scheduling system that implements scheduling models across different sites.

High performance schedulers employ predictive models to evaluate the performance of the application on the underlying system; they use this information to determine the assignment of tasks, communication, and data to resources with the goal of leveraging the performance potential of the target platform. The high performance scheduling problem consists of the following steps: selection of a set of resources on which to schedule the application (resource allocation), assign application tasks to compute resources (partitioning), distribute data and computation (data placement), order tasks on computer resources (computation scheduling), and finally order communication between tasks (communication scheduling). *High performance schedulers* are software systems that use scheduling models to predict performance, determine application schedules based on these models, and take action to implement the resulting schedule. A *high performance scheduling model* consists of an *application model*, which abstracts the set of programs to be scheduled, a *performance model*, which abstracts the behavior of the application, and a *scheduling policy*, a set of rules for scheduling.

Application models for high performance schedulers represent the application by a data-flow-style application graph or by a set of application characteristics, which may or may not include a structural task dependency graph. SEA [31] and VDCE [32] represent an application as dependency graphs of coarse-grained tasks. MARS [33] builds application dependency graphs under the assumption that applications are partitioned into independent sequential phases. On the other hand, AppLes and I-SOFT [34] take the approach of representing programs in terms of their resource requirements. Dome [35] and SPP(X) [36] provide language abstractions that are compiled into a low-level application dependency graph.

Current performance models can be classified as scheduler-derived (e.g., Dome, SPP(X), MARS, VDCE) or userderived (e.g., AppLeS, I-SOFT, SEA). While scheduler-derived performance models are built using dynamic information combined with either language abstraction or dependency graphs, user-derived scheduler models are provided by the user. Providing predictable application execution times is a challenge due to fluctuations in resource capacities. Advanced reservation (e.g. Maui [37]) offers exclusive use of a portion of the capacity of a resource for grid jobs. The drawback of this approach is that not all local schedulers provide advanced reservation support. Predictive techniques (e.g. AppLes) use statistical extrapolation of historical data to forecast utilization and performance of the resources. It is assumed that the data of past executions on specific resources is available, which may be a limitation. Feedback control (e.g CHAIMS [38], GrADSoft [39]) compare measured performance with desired performance to provide correction dynamically. This approach requires extensive use of application development and profiling tools.

A number of sophisticated scheduling policies have been devised to address the scheduling problem [40–42]. Although scheduling policy is an area that has been investigated for a long time in distributed computing; grid computing systems present unique characteristics that demand reexamination.

We consider our approach provides an improved scenario for dynamic scheduling on grid computing by introducing the use of quality of service metrics.

2.4 The Maximum Urgency First algorithm

The Maximum Urgency First (MUF) scheduling algorithm was proposed by Stewart et. al.[15, 43, 44] as a flexible scheduler to support changing behaviors in sensor-based control systems. The proposed QB-MUF algorithm is the result of a modification of the manner in which the urgency criteria of a job is calculated. The original MUF algorithm gives each job an urgency factor defined as a combination

of two fixed priorities (criticality and user priority) and a dynamic priority (laxity). On the other hand, the QB-MUF algorithm proposes the change of a static priority on behalf of a second dynamic priority based on the quality of service for the job. See sections 3.2 and 3.3 for a detailed definition of the new urgency criteria and the proposed QB-MUF algorithm, respectively.

MUF combines the advantages of the Earliest Deadline First (EDF) and Minimum Laxity First (MLF) algorithms. EDF uses the deadline of a job as its priority. The job with the earliest deadline has the highest priority to be executed. MLF assigns a laxity to each job and selects the job with the minimum laxity to be executed next. The difference between EDF and MLF is that MLF considers the execution time of a job, while EDF does not.

A particular application of the MUF algorithm in avionics mission computing was proposed by Levine et. al. [14]. They introduced the use of the MUF as a dynamic scheduling strategy to address issues such as under-utilization of resources, variations in activities and flexible prioritization of jobs. Furthermore, the use of MUF intends to avoid the trade off carried by traditional dynamic scheduling strategies which include a higher run-time scheduling overhead and an additional application development complexity.

2.5 Related Work

The following are different efforts in the application of dynamic scheduling strategies as well as the comparison between their approaches and our proposed QB-MUF algorithm.

Kalogeraki et. al. [45] proposed a dynamic scheduling algorithm that monitors the computation times and resource requirements of a job to determine a feasible schedule of method invocations on processors. Such a schedule is driven by the laxity and the priority of the job. Ligang et. al. [46] proposed a dynamic framework with

local and global schedulers based on the EDF criteria. In this approach, jobs are rejected if their deadlines cannot be met under the condition of still guaranteeing the requirements of existing jobs. Zolfaghari et. al. [47] proposed an improvement for the MLF algorithm. Our approach is a departure from the above work providing a new formulation of the urgency criteria that includes information related to the relevance and laxity of the jobs as well as the probability of job failure. The later factor is defined as a QoS metric and represents the major difference of our approach.

Hartmann et. al. [48] presented a framework for data scheduling in packet-based wireless systems. This approach is based on the assumption that each user is characterized by a set of QoS requirements as his/her flow is accepted into the system. The authors define the residual time as a generalized measure for the urgency of the next packets over the flow. This residual time may be interpreted as a kind of laxity measure. Although the authors deal with QoS, the urgency of each packet is calculated based on the packet's residual time, leaving the QoS merely as a guide for the assignment of packets to the adequate cell. Our proposed QB-MUF algorithm deals with QoS parameters created from the dynamic characteristics of each job, generating a QoS Factor that is included together with a Laxity Factor into the urgency of a Job.

Yuan et. al. [49] conveyed an analysis of the QoS properties in Manufacturing Grids (MG). A MG workflow can be defined as the composition of manufacturing activities executed on heterogeneous and distributed manufacturing resources. The authors presented a scheduling algorithm based on QoS. The main difference with our approach is that ours takes into account the dynamic internal characteristics of a job to calculate its probability of success.

An approach that includes the concept of QoS degradable jobs was presented by Mittal et. al.[50] They proposed dynamic scheduling algorithms for integrated scheduling of hard and QoS degradable jobs in real-time multiprocessor systems.

The real-time jobs are represented by two workload models, imprecise computation and (m,k)-firm guarantee, which quantify the trade-off between schedulability and result quality. This trade-off analysis will be introduced in our dynamic scheduling framework as an additional feature.

The next sections make a brief description of *GridSim* and *SimJava*, two simulation tools used to implement a proof of concept and to complete the first version of our framework, respectively.

2.6 Gridsim

GridSim[1] is a Java-based discrete-event grid simulation toolkit, which allows modeling and simulation of entities in parallel and distributed computing systems. Those computing systems refereed to by *GridSim* as GridResources may be heterogeneous, using both time-shared and space-shared execution policies. In this way, *GridSim* allows the facility to simulate different heterogeneous resources, application, resource brokers and schedulers.

Some important features of *GridSim* are:

- It enables the representation of heterogeneous types of resources
- Resources can be modeled operating under space-shared or time-shared mode
- Resources can be booked for advance reservation
- It supports simulation of both static and dynamic schedulers
- Network speed between resources can be specified.

[1] http://www.gridbus.org/gridsim/

A simple scheduling model was implemented on *GridSim* as the proof of concept for the proposed *QB-MUF* algorithm. This model includes a source of jobs, a local scheduler and a resource with the capability to generate simple reports. The simplicity of this model avoids the requirement of a global scheduler or dispatchers.

The objective of the proof of concept is to observe the tendency of the output of the *QB-MUF* algorithm and to confirm that its output behavior points to the expected results.

During the implementation of the proof of concept model, the need to address an important quantity of information about the job according to each stage it passes through was notorious. In spite of the benefits of *GridSim* and its resource oriented model, it does not count with a generous managing of internal job characteristics.

The limited capacity to address job characteristics and the desire to construct a general framework, instead of just a grid-oriented framework, generates a search one level down in the architecture of *GridSim*. In this search, *SimJava* was found; *SimJava* is the discrete event simulation package where *GridSim* is implemented on.

The next section, further explores *SimJava* and its features.

2.7 SimJava

SimJava[2] is a process based discrete event simulation package for Java, with animation facilities. For SimJava, a simulation is composed of a set of entities each running in its own thread. Such entities are connected together and can stay in communication with each other by sending and receiving event objects. These event objects should be sent and received by output and input ports respectively which are predefined inside each entity.

[2] http://www.icsa.informatics.ed.ac.uk/research/groups/hase/simjava/

In *SimJava*, a central system class controls all the running threads, advances the simulation time, maintains the internal event queues, and delivers the events. Furthermore, entities synchronization issues are addressed by passing objects of the class Sim_event.

The following is a short description of some steps used to construct a simulation using *SimJava*:

- Write in code the behavior of each simulation entity by overriding their `body()` methods. Each simulation entity has to extend the standard `Sim_entity` class to inherit the body method.
- It is necessary to add an instance of simulation entity to the static `Sim_system` object by using its `add()` method.
- Link the adequate entities' ports together by using the method `link_ports()` of the `Sim_system` object.
- The last step is to activate the simulation by calling the `run()` method of the `Sim_system` object.

SimJava was used to implement the first version of the proposed dynamic scheduling framework, which supports the QB-MUF algorithm. The components of the framework architecture together with their behavior were mapped into entities of the *SimJava* class `Sim_entity`. Also, a main program which encases the logic to build the representation of a whole scheduling environment was implemented. This program supports scheduling algorithms including FIFO and a laxity based algorithm besides the proposed QB-MUF. Another feature of the implemented framework is the possibility of implementing other scheduling methodologies by implementing the abstract class `LocalScheduler` but restricted to using the job characteristics pre-defined with in the framework.

A more detailed explanation about the implementation of the proposed dynamic scheduling framework and the proposed QB-MUF algorithm will be provided in chapter 3.

CHAPTER 3
A DYNAMIC SCHEDULING FRAMEWORK

This chapter presents the developed framework in detail. Section 3.1 describes a general scheduling model used as a base for the scheduling framework. Section 3.2 introduces a new urgency criteria; it is a key concept used in the proposed QB-MUF algorithm. Section 3.3 describes the proposed QB-MUF algorithm. Finally, section 3.4 shows a full description of the framework architecture, their components and their functionality.

3.1 General Scheduling Model

When a Job to be processed arrives, the first step is to extract static meta-data from the job and customer information to map it into a characterized job that can be managed by the scheduler framework. This meta-data includes a Job Id, the deadline of the job, a relevance of the job and the existence or absence of faults in the original job file. Secondly, a job path is defined, step by step, based on the urgency (see the particular definition of urgency used in section 3.2) of the job and the on-time status of the resources of the system. Next, a workflow engine may move the job from one resource to another in order to execute the required processes. Each set of resources or stages of the path release meta-data that helps calculate the actual capability of the system and a measure of probability of error for the job in the following stages. This probability of error is calculated based on the occurrence of faults in the job (see section 3.2).

In each set of resources (stage) in the system, a local scheduler manages its own schedule into the stage according to the job urgency and the expected and unexpected events present within the resources of the system. Defining a model implies describing the most important components identified as part of the workflow process. The following sub-sections describe job, process, resources and set of resources in a general scheduling model.

3.1.1 Job

A job may be defined as any existent file that requires the execution of a sequence of processes supported by the stages of a defined workflow. Jobs are considered aperiodic, meaning that job arrivals are not known a priori. Each Job has particular static parameters that describe it. These parameters include job arrival time (Ja_i), job deadline (Jd_i), job relevance (Jr_i) and job size (Js_i). Also, dynamic parameters are calculated in real time according to the characteristics of the system. These dynamic parameters include job latency ($Jlat_i$) and job laxity ($Jlax_i$). Job latency is the time required to complete a job. Job laxity is the difference between the job deadline and the sum of the job latency and the current time at any moment of the job processing (see section 3.2 for more details). Furthermore, we take into account a special kind of job parameter, the faults. The occurrence of a fault in a job indicates the existence or absence of a specific feature according to a specific problem. Faults may be detected when a job arrives to the system as well as during the processing of the job. This means that processing of a job in the system can cause the elimination or generation of faults. Normally faults are considered to be the cause of failed jobs as an outcome from the system. The explanation of how job faults are mapped as possible job failures is presented in the section 3.2.

3.1.2 Process

A process can be defined as an action performed on a job. Commonly, a job with a defined path requires a specific order for the realization of a specific process. Processes are related to resources in the system; each resource can perform a specific process on a job. Furthermore, the system may have heterogeneous sets of resources serving as a unique process but on different levels of performance or quality.

3.1.3 Resource

A resource performs a process on a job. A resource may be a software tool, an expert or a machine executing a process on a job. Different kinds of entities exist that can execute similar processes but with different specifications. As an example, in *digital publishing*, different preflight tools may execute a preflight process but offer different levels of artifact[1] recognition services. A more detailed explanation about the application of the proposed framework on *digital publishing* is presented in chapter 4.

3.1.4 Set of Resources

A set of resources is defined as an entity that contains a number of homogeneous resources. However, the resources in different sets of resources may have different characteristics. This gives the framework the opportunity of addressing heterogeneous sets of resources.

Processes, sets of resources and resources may be formally defined as:

$$R = \{r_{jk_jl_{k_j}}\} \; \forall \; j = 1, ..., m \;\; \wedge \;\; k_j = 1, ..., n_j \;\; \wedge \;\; l_{k_j} = 1, ..., s_{k_j}, \tag{3.1}$$

[1] The name artifact is used to define any type of defects in digital documents.

where m is the number of processes (stages) in the system, n_j corresponds to the number of sets of resources available to perform the process P_j and s_{k_j} corresponds to the number of resources available in the set of resources k_j. In this way, $r_{jk_jl_{k_j}}$ is an identifier for each autonomous resource which can execute the process P_j. The different number of heterogeneous sets of resources that can execute the process P_j may be represented by the variation of the sub-index k_j between 1 and n_j where n_j is different for each process P_j. In the same manner, the different number of homogeneous resources that can execute the process P_j on a given set of resources k_j, may be represented by the variation of the sub-index l_{k_j} between 1 and s_{n_j} where s_{n_j} is different for each set of resources K_j.

On the other hand, the property of a set of resources that affects the behavior of the system and the outcomes of the scheduling process is the mean of the urgency of the jobs that compose the load of a set of resources (Rm_{jk_j}). This measure shows the tendency of the urgency of most jobs waiting for process in a set of resources R_{jk_j} at a specific time. The load of each resource can be managed by the scheduler through the process of generating a schedule for each job that requires the resource.

3.2 A New Urgency Criteria

Our scheduling strategy focuses on providing high priority to jobs with low probability of failing. To achieve this a new urgency criteria equation is introduced to account for relevance, laxity and probability of failures of incoming jobs. The probability of the failure of a job will be estimated according to the fault occurrence in a job because the presence of faults or the combination of some of these faults can eventually become errors, depending on their severity.

The proposed urgency criteria is based on one static and two dynamic parameters (See Figure 3–1) each pondered by a weighting factor, W_1, W_2 and W_3 respectively. These parameters are defined as follows:

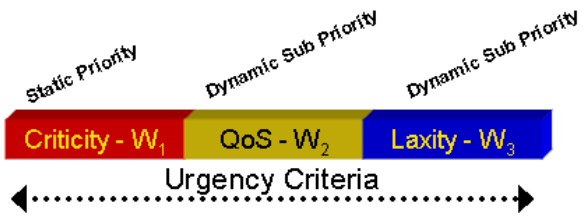

Figure 3–1: Urgency Criteria

1. **Criticality** (Relevance). This static factor is initially established by the user according to experience and/or job importance. Criticality values range between 0 and 100. The presence of this factor intends to account for the importance of a job at the time of generating a scheduling for a job. Since the research focuses on the contribution of merging QoS and time parameters, the scenarios used for our experiment do not consider variations for this factor in sets of incoming jobs. This consideration does not mean that the experiments will not present priority fluctuation for jobs. The priority fluctuation will be introduced for two specific situations:
   - The variations on laxity, caused by the simulation time going forward and by the changes in the eventual redefinitions of job's deadline, will generate possible variations of the priority of the jobs during a simulation.
   - The possible changes in the definition of the function for the quality of service factor ($QoSF$), in different processes, will produce priority fluctuation of jobs between different stages.

   The following two items in this list will deepen the explanation of the quality of service and the laxity factors.

2. **Quality of Service** (QoS). Scheduling involves matching of the job needs with the resource availability and capability, and addressing the concern of the quality of the match. In the scheduling strategy for this research, the QoS is defined as the probability of success for each job and its value ranges between 0 and 100.

This QoS will be used as the quality of service factor $QoSF$ in the function that defines the urgency of a job in the $QB\text{-}MUF$ algorithm. A way to define the probability of success for a job is in terms of its probability of failure. Thus, the $QoSF$ of a job can be defined as:

$$QoSF \;=\; 100 \;-\; JPF \;*\; 100 \tag{3.2}$$

where JPF is the job probability of failure. The JPF can be defined as a mapping of the presence of selected faults in a job. An example of a JPF definition where faults are mapped into a probability of failure is:

$$JPF \;=\; \begin{cases} [random() * 6/100] & if \quad f_1 = 0 \;\wedge\; f_2 = 0 \\ 40 + [random() * 16/100] & if \quad f_1 = 1 \;\wedge\; f_2 = 0 \\ 20 + [random() * 21/100] & if \quad f_1 = 0 \;\wedge\; f_2 = 1 \\ 50 + [random() * 21/100] & if \quad f_1 = 1 \;\wedge\; f_2 = 1 \end{cases} \tag{3.3}$$

where f_1 and f_2 are two kinds of faults present in a job and $random()$ is a function that returns a real random number Rn such that $Rn \;\in\; [0..1)$. In general, from 3.2 and 3.3, it is possible to define the the quality of service factor ($QoSF$) as a function of possible faults of a job:

$$QoSF = F(f_1, f_2, ...f_n), \tag{3.4}$$

where f_i are the faults that can become failures when present in a job. Such $QoSF$ definition can be different for each different scheduling problem.

On the other hand, different QoS metrics can be defined according to the objectives of the scheduling strategy in use. Since the QoS definition, in this case, is based on the probability of success for each job, the quality of service metrics for this strategy should be defined targeting to exhibit how the algorithms under evaluation address the probability of success of a job. Two metrics were

defined to observe during the experimentation. First, the number of successful jobs delivered. Second, the mean waiting time of successfully delivered jobs.

The number of successful jobs delivered can be represented by a graphic showing on the y-axis the number of successful jobs delivered and on the x-axis the simulation time. The objective of this graphic is to observe the behavior of the metric *number of successful jobs* delivered while the simulation time goes forward. The expected behavior of the metric is that for those algorithms that take into account QoS, the number of successful jobs delivered increases faster than in other algorithms, especially in the first part of the simulation.

The mean waiting time of successfully delivered jobs ($\overline{SJwt}$) is defined by a equation of the form:

$$\overline{SJwt} \;=\; \frac{\sum_{i=1}^{i=NJ} Jwt_i \;*\; Succes_i}{NSJ} \tag{3.5}$$

where NJ is the number of processed jobs, Jwt_i is the time that the job J_i has to wait since it was received until it starts processing, NSJ is the number of successful jobs at the end of the simulation and $Success_i$ can take only the values 0 or 1 according to whether J_i was failure or success, respectively.

The mean waiting time of successfully delivered jobs metric can be represented by a bars graphic. This graphic shows a bar with the value of the mean waiting time for each algorithm under evaluation. The expected behavior of the metric, is that for those algorithms that take into account QoS, the mean waiting time of successfully delivered jobs be shorter than the mean waiting time for the successfully delivered jobs in the simulation of other algorithms.

3. **Laxity**. This dynamic factor is defined as the difference between the job deadline and the sum of the job latency and the current time (t_{now}) at any moment of job

processing. Thus, laxity can be represented as:

$$Jlax = Jd - (t_{now} + Jlat), \tag{3.6}$$

where Jd is the Job deadline, t_{now} is the current time of calculation, and $Jlat$ is the expected latency of the Job. Figure 3–2 provides a graphic illustration of the meaning of laxity.

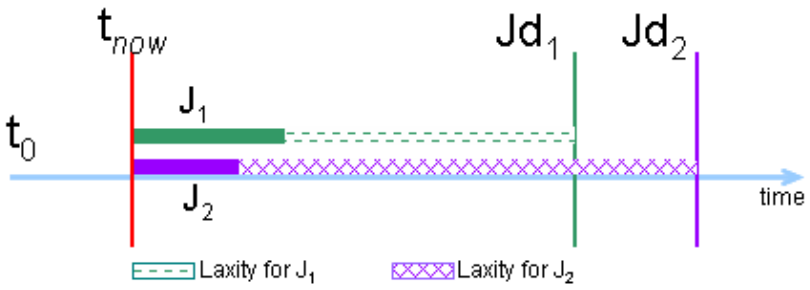

Figure 3–2: Laxity of a Job

Laxity as defined above is not bounded (it is as large as the difference between the job deadline and the sum of the job latency and the calculation time) and may lead to unrealistic urgency criteria values. The reasons for this are, first the meaning of the use of laxity as urgency criteria; second a possible definition of a laxity factor, which emerges from a requirement of itself, where its value has to range between 0 and 100. The reasons are explained in detail as follows:

(a) The strategies that take into a count the laxity as a way to prioritize jobs, should give a higher priority to jobs with a low laxity value. This shows that the relation between laxity and priority of a job are inversely related.

(b) From (a) and with the requirement of generating values of a laxity factor between 0 and 100, a initial definition for a laxity factor can be:

$$LaxityF \;=\; \frac{1}{Jlax} * 100 \qquad \forall\; Jlax \;\geq\; 1 \tag{3.7}$$

This definition presents a special situation; when the value of $Jlax$ is relatively large, all the values of $LaxityF$ will approach 0.

$$\lim_{Jlax \to \infty} \frac{1}{Jlax} * 100 \;=\; 0 \tag{3.8}$$

The figure 3–3 shows an illustration of this phenomenom.

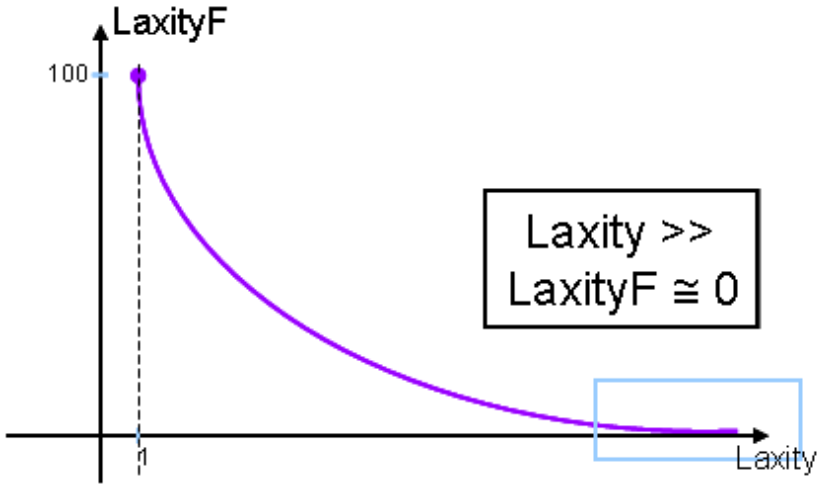

Figure 3–3: Laxity Factor of a Job

Since the range of values for laxity and units of time can vary for different scheduling problems, the situation revealed above may lead to laxity not achieving the desired effect in the urgency of a job. A worse case if that because of selecting a unit for time to small, the laxity values be always bigger therefore all the calculated values of laxity factor be 0.

One way to compensate for this is to define a modulator factor K (units of time). The magnitude of such factor should be properly defined (in units of time) according to the expected values of laxity for each particular scheduling problem.

The objective of the modulator factor is to normalize the values of laxity when they are larger than the modulator factor. Selecting a proper value for the modulator factor will lead to a laxity factor with the expected values between 0 and

100. Using the proposed modulator factor, the Laxity factor is defined as follows:

$$LaxityF = \begin{cases} (K/Jlax) * C & if \quad Jlax > 0 \ \wedge \ Jlax \geq K \\ C + (1 - (Jlax/K)) * (100 - C) & if \quad Jlax > 0 \ \wedge \ Jlax < K \end{cases} \tag{3.9}$$

where C is a constant. Figure 3–4 shows the behavior of the laxity factor according as defined in the equation 3.9.

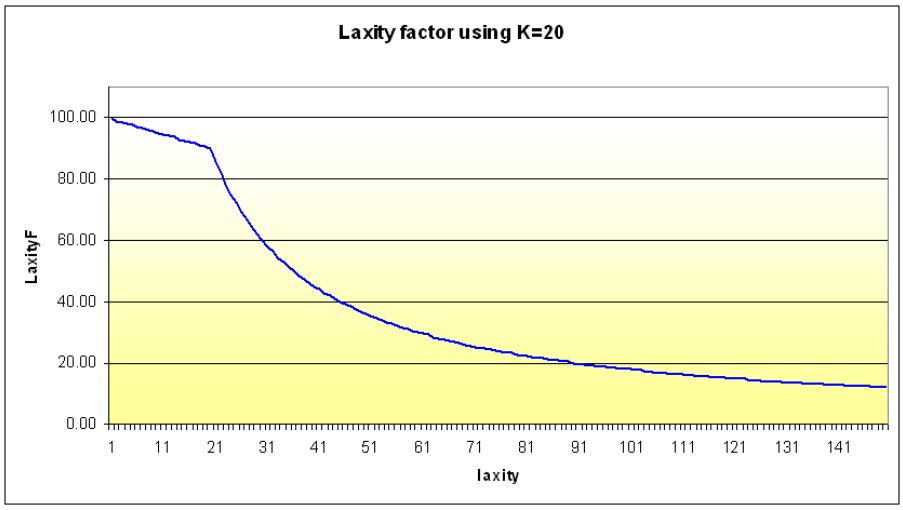

Figure 3–4: Laxity factor defined using K

In the case that $Laxity < 0$, a degradation of the quality of service on delivery time for a job will be permitted. Thereby a new deadline Jd must be generated by taking into account the probability of success of the job. The new Jd is calculated as $Jd = time_{now} + (f(Jlat) * MF)$, where $f(Jlat)$ is a function of job latency and MF is a multiply factor. The multiply factor MF works as a correction factor to penalize Jobs according to their quality of service; as a particular example, in one proposed scenario for *Digital Publishing* the MF is calculated according to the QoS of the job as follows:

$$MF = \begin{cases} 0.9 & if \quad QoS \geq 80 \\ 1.1 & if \quad 60 \geq QoS < 80 \\ 1.2 & if \quad QoS < 60 \end{cases} \tag{3.10}$$

The function for MF must be redefined to fit the requirements of each particular scheduling problem.

The resultant urgency criteria is defined in the following equation:

$$Urgency = Criticality * W_1 + QoSF * W_2 + LaxityF * W_3 \tag{3.11}$$

$$\text{where} \quad \sum_{i=1}^{3} W_i = 1 \; and \; , \; W_i > 0 \; \forall \; i \tag{3.12}$$

3.3 The QB-MUF Algorithm

Inside each set of resources in the system, a local scheduler runs the QB-MUF algorithm. The QB-MUF algorithm, in a iterative way, assigns jobs to resources by considering resource availability and the urgency factor of each job. If there are not available resources to process a job, then the job is sent to a waiting queue. To deal with resource contingencies (Unexpected resources getting out of order) each local scheduler keeps a list of the current active and available resources. Such list is updated for the resources by sending a message of activity to the *Local Scheduler* each time the resources finish a job. We assume that resources do not fail while executing a task and only have a probability of failure after executing a task and before to sending a message of activity to the *LocalScheduler*. If, for any reason, a *Resource* fails, the activity message is not sent to the *Local Scheduler* and the resource is not included into the list of active and available resources. In this case, The *Local Scheduler* will not send new jobs to a failing *Resource* until the *Resource* resumes and sends an activity message to the *Local Scheduler*. An extended pseudo code of a *Local Scheduler* is presented in figure 3–11.

The urgency criteria of each job in the waiting queue is updated, each time that a new job arrives or each time that a change on the waiting queue occurs, to insure the flow of jobs with high urgency first. This means, the flow of jobs with the best balance between high probability of success and job laxity. Such balance may be modified, according to the requirements of each particular environment, by changing the weighting factors (W_1, W_2, W_3). For instance, if the desired scheduling strategy wants to give more relevance to delivering jobs on time than to the quality of jobs at the end of the process, the weighting factor W_3 should be bigger than the weighting factors W_1 *and* W_2. This always under the constraint $\sum_{i=1}^{3} W_i = 1$ *and* , $W_i > 0 \, \forall \, i$. The general QB-MUF algorithm inside each set of resources can be viewed in figure 3–5.

```
While (Arrive a new job) or (There are jobs to schedule)
      if (There are available resources)
         for each job i to schedule
             calculate job urgency;
             allocate job;
         end for
      else
         if (Queue.length > 0 )
             update urgency factor
         end if
         insert ordered job to the queue
      end if
end while
```

Figure 3–5: Summarized pseudo code of the QB-MUF.

To deal with the priority fluctuation introduced by de definition of the laxity factor (see section 3.2) and by the variation on the definitions of quality of service,

```
calculate Job Urgency(Job objJob)
     Urgency = (W1*objJob.Criticality
     + W2 * objJob.QoS(actualStage)
     + W3 * objJob.LaxityF)
     if (Urgency <= 0)
        Urgency = 1 //to ensure Urgency > 0
     end if
     objJob.setUrgency(actualStage, Urgency)
end
```

Figure 3–6: Algorithm to calculate the urgency of a Job inside each stage.

the *Local Scheduler* uses the Urgency criteria, which includes the above mentioned two factors, to keep a waiting queue ordered. Ordering the waiting queue each time a new event arrives to the *Local Scheduler* provides a strategy to take into account the priority fluctuation in the scheduling strategy.

3.4 Framework Architecture

The proposed urgency criteria and scheduling algorithm are embedded into a dynamic scheduling framework. The main goal of this framework is to provide a reusable infrastructure to design and evaluate scheduling strategies. An important feature is that the framework has the flexibility to be used in different production environments including digital publishing and grid computing. The architecture of the framework is shown in Figure 3–7.

The general components of the framework are described in the following subsections. Some special components will be described inside the description of their containers as in the case of *Local Scheduler* and *Resources* which are embedded into each *ResourceSet*.

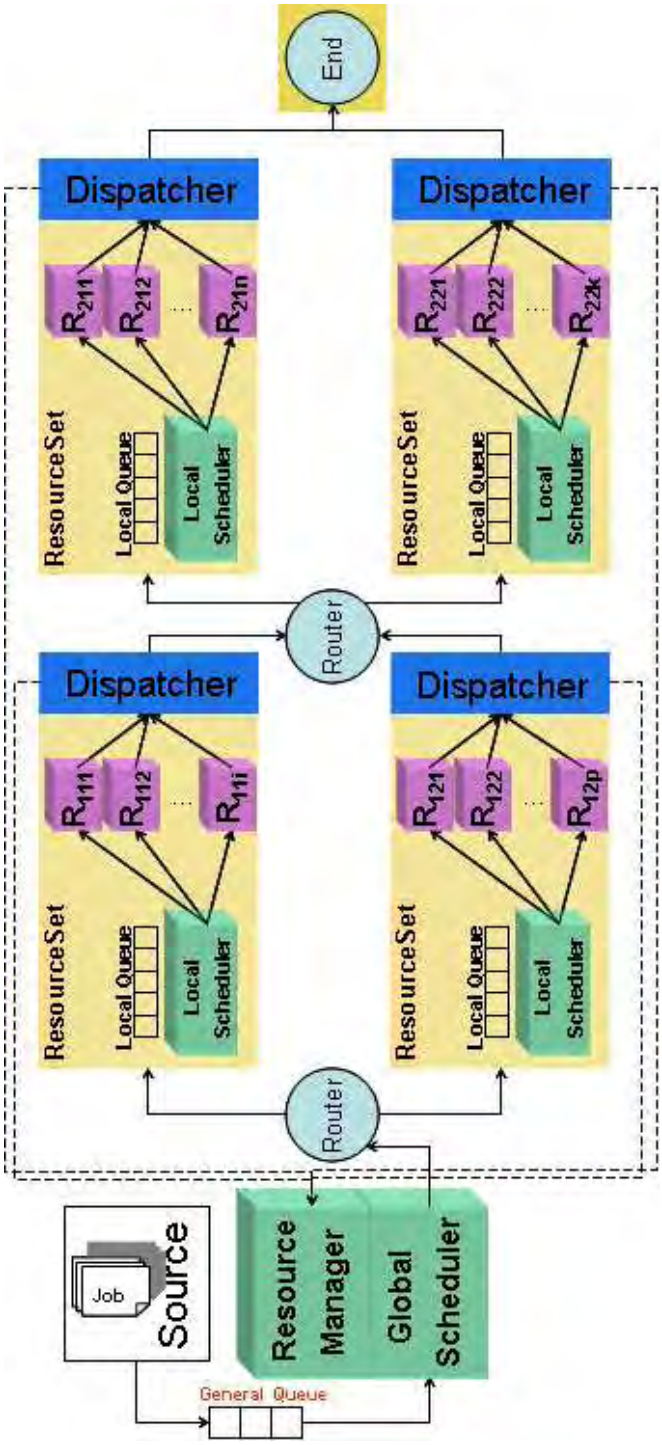

Figure 3–7: Framework Architecture

3.4.1 The Source

The *Source* component works as a unique entity generating new *Jobs* for the framework. Figure 3–8 shows the pseudo code of the body of each *Source* component in the framework.

```
void body()
     for i=1 to i=MaximumNumberofJobs
          create a new job
          initialize the job parameters(size, faults)
          send the job to the global scheduler
          pause(randomExpNdelay.sample())
     end for
end void
```

Figure 3–8: Pseudo-code of the body method in the Source Component.

Furthermore, the *Source* is in charge of generating Jobs with the characterization required for each specific problem. Such characterization includes:

- The Size of the job. This is a number representing an amount of data; the units (Bytes, Kb, Mb) are predefined according to the characteristics of the problem. To generate the size of a job, we used two of the available distribution functions provided by *SimJava*, a normal distribution and a exponential distribution. For instance, Job size (Js) can be defined as follows:

$$Js \quad = \quad randomNormal.sample(); \tag{3.13}$$

  where $randomNormal.sample()$ is a number generator which fits a normal distributions with fixed mean and variance.

- The elapsed time between the arrival of new jobs. This is the delay, in units of time, used by the source between job generations. This delay time is drawn from a random number generator, as in the case presented above, which is parameterized to fit either a normal distribution or exponential distribution. For instance, Job delay (Jdy) between job generations can be defined as follows:

$$Jdy \quad = \quad randomExpNdelay.sample(); \tag{3.14}$$

  where $randomExpNdelay.sample()$ is a random generator of numbers exponentially distributed with a fixed mean.

- The faults contained in each Job. Up to five faults can be considered to be present in a job running inside the framework. The presence/absence of each fault is generated according to a random number generator provided by *SimJava*. The presence/absence of a fault in a job is commonly defined as follows:

$$flt \quad = \quad 1 \; if \; randomGene() * 100 \; <= \; FT, \; flt \; = \; 0 \; otherwise; \tag{3.15}$$

  where $flt = 1$ means the fault is presented in the job, $randomGene()$ is a number generator capable of producing a sample uniformly distributed between 0 and 1, and FT represents a threshold below which a fault is considered as present in a job for a specific problem.

To generate samples from the required distributions in the *Source* component, a random[2] number generator, modified to fit a desired distribution is used. This modified random number generators are provided by *SimJava* through the distribution classes `Sim_*_obj`. These distribution classes include generators for a normal

[2] Indeed, the random number generators provided by SimJava are pseudo-random. This makes possible the generation of repeatable experiments in the framework.

distribution and a exponential distribution. When an entity, such as the *Source* component, requires a determined distribution, it has to make an instance of the desired distribution class and then uses the method `sample()` whenever a sample is required.

3.4.2 The Global Scheduler and Resource Manager

The *Global Scheduler* and the *Resource Manager* work as a whole central entity executing two tasks. Playing as *Global Scheduler* uses the information gathered while playing as *Resource Manager* to select the optimal next set of resources (*ResourceSet*) available for any job that requires the execution of a process. The *Resource Manager* gets the load and performance information, when required, from each set of resources in each stage of the system. The *Resource Manager* maintains on line the information of the best *ResourceSet* during any time of the simulation and in any stage of *The Framework*. The figure 3–9 shows the pseudo code of the body of the *GlobalScheduler* in the framework.

3.4.3 The ResourceSet

Each *Stage* of *The Framework* can be composed of a number of sets of resources, each one referred to as *ResourceSet*. A *ResourceSet* is composed of one *Local Scheduler*, one Dispatcher and one or more homogeneous *Resources*. The *ResourceSet* works as a whole entity capable of receiving a Job, processing and sending it to the next step through a *Router*. Furthermore, the *ResourceSet* must inform the *Global Scheduler* about changes in the work load caused by the processing of each *Job*, as well as to receive information of the best next stage to set it up in the dispatcher of the current *ResourceSet*. Figure 3–10 shows the pseudo code of the body of each *ResourceSet*.

The following is the description of each of the components mentioned above:

```
void body()
     While (Simulation is running)
           wait for a new event
           if new event is a new job event
              extract the job from the event
              set the job global arrival time
              stamp in the job the port-name of the next stage
              send the job to the next router
           end if
           if new event is a change on a ResourceSet
              extract the info from the event
              save the info of change of the ResourceSet
              if the new mean < smaller mean saved
                 save new mean as smaller mean of the stage
                 if the stage where the change occurs > 1
                    send info of the best ResourceSet
                     to stages prior to current ResourceSet stage
                 end if
              else
                 if ResSet with smallest mean == ResSet of new mean
                 find a new smallest mean
                 save the new smallest mean
                 if the stage where the change occurs > 1
                    send info of the best ResourceSet
                     to stages prior to current ResourceSet stage
                 end if
              end if
           end if
     end while
end void
```

Figure 3–9: Pseudo-code of the body method in the GlobalScheduler Component.

```
void body()
     While (Simulation is running)
           wait for a new event
           if new event is a new job event
              extract the job from the event
              set the job arrival time
              set the job stage to the current stage
              send the job to the LocalScheduler of
                the current ResourceSet
           end if
           if new event is change of next port event
              extract the new port from the event
              set the dispatcher nextPort with the new port
           end if
     end while
end void
```

Figure 3–10: Pseudo-code of the body method in the ResourceSet Component.

The Local Scheduler

A *Local Scheduler* works as an entity embedded in each *ResourceSet*. The *Local Scheduler* deals with the local allocation of available resources among jobs by using the *QB-MUF* algorithm as well as the information of the available *Resources* gathered from themselves. The *Local Scheduler* is also in charge of maintaining updated the mean of the quality of service of the jobs existing in the waiting queue. Additionally, the *Local Scheduler* updates a list with the currently available resources available inside of the *ResourceSet* where the *LocalScheduler* belongs to.

Figure 3–11 shows the pseudo-code of the body of each *Resource* in the framework.

The Resource

A *Resource* is an entity capable of executing a process on a job. The resource receives a Job to be processed from the *Local Scheduler*. The processing time of the *Job* is calculated based on the size of the *Job* plus a penalty in time units according to the probability of failure of the *Job*. The *Resource* will be busy and unavailable during the processing time calculated for the job that is currently being executed. After processing a *Job*, the *Resource* passes it to the *Dispatcher* of the *ResourceSet* and informs the *Local Scheduler* that it is available to process a new *Job*. Figure 3–12 shows the pseudo-code of the body of each *Resource* in the framework.

The Dispatcher

A *Dispatcher* in a *ResourceSet*, is an entity in charge of redirecting processed Jobs from any *Resource* in a *ResourceSet*, to the *Router* connected to all of the *ResourceSets* composing the next *Stage* in the *Framework*. The dispatcher also puts, in the form of a tag inside the *Job*, the information concerning the most favorable next step received from the *Global Scheduler*. Furthermore, the *Dispatcher* is in charge of receiving information about changes in the queue of the *ResourceSet*. This information, in form of the mean of the quality of service of the existing jobs in the

```
void body()
     While (Simulation is running)
           wait for a new event
           if new event is a new job event
              extract the job from the event
              calculate the job processing time
              calculate the job QoS
              calculate the job deadline
              calculate the job laxity
              calculate the job Urgency
              insert the job in the waiting queue ordered by Urgency
              calculate the changes in the mean of the QoS
               of the current jobs in the waiting queue
              send the information of change through the Dispatcher
           end if
           if new event is a free resource event
              extract the id of the free resource from the event
              add the id of the free resource at the end
               of list of free resources
           end if
           if listOfFreeResources.size > 0
              if jobWaitingQueue.size > 0
                 extract the first job in jobWaitingQueue
                 send the job to the first resource
                   in listOfFreeResources
                 remove the resource in listOfFreeResources
                 calculate the changes in the mean of the QoS
                  of the current jobs in the queue
                 send the information of change through
                   the Dispatcher
              end if
           end if
     end while
end void
```

Figure 3–11: Pseudo-code of the body method in the LocalScheduler Component.

```
void body()
     While (Simulation is running)
           wait for a new event
           if new event is a new job event
              extract the job from the event
              set the job start execution time
              generate a delay during the processing time
              set the job finish execution time
              send the job to the Dispatcher of the ResourceSet
              announce to the LocalScheduler that
               this is a resource available
           end if
     end while
end void
```

Figure 3–12: Pseudo-code of the body method in the Resource Component.

queue, is redirected to the *GlobalScheduler* in order to inform about the status of the *ResourceSet*. Figure 3–13 shows the pseudo code of the body of each *Dispatcher* in the framework.

```
void body()
     While (Simulation is running)
           wait for a new event
           if new event is a Job event
              extract the job from the event
              stamp inside the job the name
               of the port of the next stage
              send the job to the next router
           end if
           if new event is a change of the queue
              extract the information of change from the event
              send the information of the change to
               the GlobalScheduler
           end if
     end while
end void
```

Figure 3–13: Pseudo-code of the body method in the Dispatcher Component.

3.4.4 The Router

In the *Framework*, a *Router* transfers a *Job* between the *Global Scheduler* and the first Stage, between two consecutive *Stages* of the *Framework* or between the last *Stage* of the *Framework* and the "final" *Stage* of the *Framework*. The "final" *Stage* of the *Framework* does not perform any process on the *Job*; it only receives the *Job* and makes a report with the information of the process of the Job in the whole *Framework*. To perform the redirection of the job toward the correct next

stage, the *Router* uses the name of the port which was previously stamped in the job either by the dispatcher on the previous stage or by the global scheduler. The figure 3–14 shows the pseudo code of the body of each *Router* in the framework.

```
void body()
     While (Simulation is running)
           wait for a new event
           if new event is a redirection event
              extract the job from the event
              send the job to the next stage through
              the port name stamped in the job
           end if
     end while
end void
```

Figure 3–14: Pseudo-code of the body method in the Router Component.

3.5 Developing tools

3.5.1 Programming language

The framework was built using the Java 2 Platform, Standard Edition (J2SE)[3] . Java was built by Sun Microsystems programmers, it was formally announced in public at a major conference in May 1995. Java is now one of the most widely used programming languages, mainly on networks and for web service.

The major benefits of using Java as the programming language for our framework software include:

[3] http://java.sun.com/j2se/1.5.0/

- Java is object oriented, which increases the flexibility and maintainability of the programming process.
- The multi-threading is built in. This facilitates multi-threading programming.
- Java is multi-platform, this makes possible to exchange platforms during both, the programming process and the running program process.

3.5.2 Eclipse

The Integrated Development Environment (IDE) used to build our framework was Eclipse[4] SDK 3.1, a complete development environment for Eclipse-based tools. We decide to use Eclipse because it allows easy programming in Java, among other programming languages, and also provides a good environment to code, build, run, and debug applications. Some of the relevant features considered to select Eclipse include:

- Eclipse is open-source.
- Once the setup process of a project is done, the project is automatically compiled, all the time.
- The debugging lets to find and fix bugs faster.
- The compilation errors are identified immediately while coding.
- Eclipse is compatible with other object oriented design tools, such IBM Rational Software Architect.

[4] http://www.eclipse.org/eclipse/

CHAPTER 4
QOS BASED DYNAMIC SCHEDULING APPLIED TO DIGITAL PUBLISHING

4.1 Digital Publishing

Digital publishing permits the linking of printing presses to computers, thereby bypassing the need for film and/or plate. As a result, digital publishing offers the potential to raise the quality level for short-run printing, and enables the printing of documents that are highly variable in data content and layout. However, the realization of this potential has, to date, been seriously hampered by a number of difficulties. These include the problem of getting the document to print correctly without artifacts on the press and the difficulty of managing the increasingly complex workflow that results from shorter run jobs that must be completed in less time. Thus, digital publishing not only opens up new business but also requires new business models which lead to new workflow designs. The fact that information remains digital from the design stage all the way to printing leads to the potential automation of processes that in traditional workshops are still manually executed. The typical stages in a digital publishing workflow are summarized in Table 4–1.

For artifact detection, the traditional and the digital print shops have two processes before sending any job to the printers: preflight and proofing. The purpose of these processes is to make sure that the document is free of errors before it is sent to the press. Nowadays the preflight process is almost done automatically, but the proofing stage is still executed by human operators. In order to achieve a fully automated end-to-end digital publishing workflow, a variation to the digital publishing

Table 4–1: Digital Publishing Workflow Stages

Stage	**Description**
Pre-flight	Check if the digital document has all the elements required to perform well in the production workflow. These elements include page file format, image resolution, font types, safety margins, mismatched colors.
Trapping	Overlap colors to compensate for press registration. Registration is the accurate positioning of two or more colors of ink in a printed sheet.
Imposition	Arrangement of individual pages on a press sheet, so that when folded and trimmed, the pages are correctly oriented and in the proper order.
Proofing	Check an output before printing. Conventional: film-based; Soft proof: calibrated monitor; and digital proof (digital proofing printer).
Ripping	It decodes PostScript, creates an intermediate list of objects and instructions, and finally converts graphic elements into bitmaps for rendering on an output device.

workflow model was proposed by Santos [51]. Such modification, called The Automated Preflight Model (APM), is a framework for artifact detection before sending a job to ripping. As shown in Figure 4–1, the proposed APM framework is composed of the *Intent* process (1), the *Preflight* Tool (3), and the *Artifact Recognition* tool (4). Such stages together with the *Ripping* process (6) and the *Printing* processes (7), conform a modified full digital publishing workflow.

From the proposed modified full digital publishing workflow, a simplified model was built on our framework to observe the behavior of our proposed scheduling framework based on the QB-MUF algorithm in the digital publishing environment. The simplified model is conformed by taking the *Intent* process (1), the *Preflight* Tool (3), the *Artifact Recognition* Tool (4) and the *Ripping* process (6), under the following assumptions:

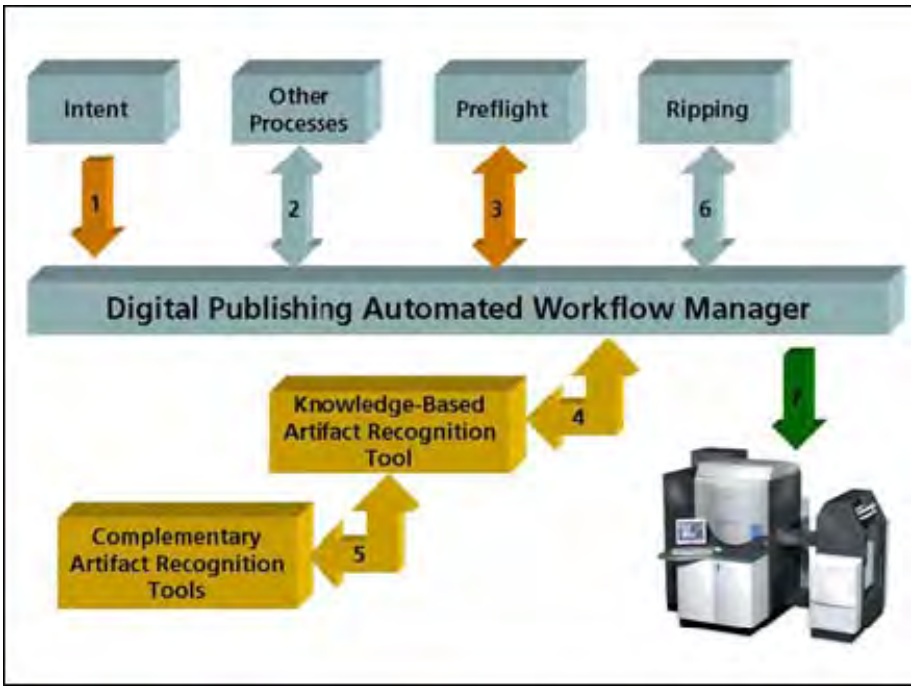

Figure 4–1: The Automated Preflight Model

1. Since the *Intent* process and the *Preflight* process are the first processes that extract important information about the properties and content of each *Job*, the simplified model assumes that those two processes have already been done. At this point, the framework will take into account enough information, which gathered from the *Intent* process and the *Preflight* process, to generate a first schedule for the *Artifact Recognition* and the *Ripping* processes.

2. The *Artifact Recognition* process is the first stage of the digital publishing workflow to be included in the framework to generate its schedule. To generate such a schedule, two specific characteristics of the *Job* gathered from the preflight process are used. Such characteristics are the faults related to fonts not embedded and the faults related to a wrong color base. The mean probability of those two faults, of being present in a *Job*, were obtained from the study of a variety of

PDF documents [52]. In such study, the PDF documents were analyzed by different preflight tools in order to identify common faults in printing documents. The mean of the probabilities of presence for the two selected faults in a *Job*, provided in such study are used to define thresholds in the equation that defines the presence of a fault in a *Job*. Such thresholds are defined as follows:

$$FT1 \ = \ 67, \ FT2 \ = \ 25, \tag{4.1}$$

where $FT1$ and $FT2$ are the threshold for the faults related to fonts not embedded (f_1) and the faults related to a wrong color base (f_2), respectively. The values for $FT1$ and $FT2$, 67 and 25 respectively, were drawn from the study mentioned above. This values corresponds to the mean probability that a job contains a fault related with fonts not embedded or a fault related with wrong color base, respectively. From equation 4.1, the presence of the faults f_1 and f_2 in a job, is defined as:

$$\begin{aligned} f_1 \ &= \ 1 \ if \ random(100) \ <= \ FT1, \ f_1 \ = \ 0 \ otherwise \\ f_2 \ &= \ 1 \ if \ random(100) \ <= \ FT2, \ f_2 \ = \ 0 \ otherwise; \end{aligned} \tag{4.2}$$

where $random(100)$ is a function that returns an integer number I such that $I \ \in \ [0, 99]$. Using equation 4.2, the function mapping the presence of f_1 and f_2, in a *Job*, into a probability of failure (JPF) is defined as follows:

$$JPF \ = \begin{cases} [random() * 6] & if \quad f_1 = 0 \ \wedge \ f_2 = 0 \\ 40 + [random() * 16] & if \quad f_1 = 1 \ \wedge \ f_2 = 0 \\ 20 + [random() * 21] & if \quad f_1 = 0 \ \wedge \ f_2 = 1 \\ 50 + [random() * 21] & if \quad f_1 = 1 \ \wedge \ f_2 = 1, \end{cases} \tag{4.3}$$

where $random()$ is a function that returns a real random number Rn such that $Rn \ \in \ [0, 1)$. The objective of the JPF function is to define the probability of failure of a job according to the kind of faults to the severity of such faults.

In this particular definition, the faults related to fonts not embedded (f_1) are considered more severe than the faults related to a wrong color base (f_2). In this way, the values selected for the constant in the definition of JPF are chosen to generate the following values:

- $JPF \in [0, 5]$, if there are no faults present in the job.
- $JPF \in [20, 40]$, if a fault related to a wrong color base is present in the job.
- $JPF \in [40, 55]$, if a fault related to fonts not embedded is present in the job.
- $JPF \in [50, 70]$, if both kind of faults are present in the job.

Finally, in order to calculate the quality of service factor $QoSF$, the JPF defined in equation 4.3 is used as follows:

$$QoSF = 100 - JPF \tag{4.4}$$

The $QoSF$ is used in the calculation of the urgency of each *Job* as shown in equation 3.11.

3. The *Ripping* process is the second stage of the digital publishing workflow to be included in the framework for the simulation. According to Santos [51], the *Artifact Recognition* tool will provide two special features, the *Artifact Name* and the *Severity Rating*. The first one contains the name given to the *Artifact* by the case base engine of the *Artifact Recognition* tool. The second one is an integer number between 0 and 5. This rating is equivalent to: no artifact, tolerable artifact, low effect artifact, high effect artifact, severe artifact, and critical artifact, respectively. For the simulated scenarios, two existent artifacts in the case base engine of the artifact recognition tool were selected, overlapping and type-face-change. Overlapping refers to the case when a component in a document or their equivalent component in the approved instance of a document, overlaps

over other components. Type face change indicates the changes in font family, size, and style, between a text component in one document instance and their equivalent text component in the approved instance of a document. The severity selected for such artifacts were 2 (low effect artifact) and 3 (high effect artifact), respectively.

Since there is no available a study similar to the one realized for the preflight process, it was assumed the same values for mean of probabilities of the faults related with fonts not embedded and the faults related with a wrong color base and, for the faults related to the overlapping and type-face-change, respectively. In the same way a $QoSF$ function, similar to the one defined for the artifact recognition process, was defined in the ripping process in order to generate a schedule.

At this point it is important to emphasize that it is not necessary to use real values of variables or function's definitions since it is possible to change such definitions in the source code of the framework. For example, it was possible to run the experiments described in this chapter with arbitrary values from the artifact recognition process. However, along with further simulations the definition and values for this process can be changed for realistic values or other assumed values in order to compare the behavior of the algorithms under evaluation.

4. For all the stages defined in the framework, the definition of the laxity factor ($LaxityF$) uses equation 3.9. Such definition indirectly uses the variable job latency ($Jlat$). The definition of $Jlat$, for all of the algorithms under evaluation, includes a penalty in processing time according to the probability of failure of each job. The equation for $Jlat$ is defied as follows:

$$Jlat = \frac{Js}{Rpr} * (1 + CONS * (\frac{JPF}{100})) \tag{4.5}$$

where Js is the size of a job, Rpr is the processing rate of the resource and $CONST$ is a constant factor used in to calculate the $Jlax$ in all sets of resources of the framework. The meaning of the factor $(1 + CONS * (\frac{JPF}{100}))$ is to introduce a penalty in the latency to those jobs with high probability of failure (JPF). The $CONST$ value can be modified to fit the requirements of each particular scheduling problem. In other words, the $CONST$ value can be used to give a larger or lower penalty in the latency to those jobs with high probability of failure. $CONS = 1$ for the experiments presented in this chapter, which means that $1 <= (1 + CONS * (\frac{JPF}{100})) <= 2$.

5. According to the characteristics of each scenario, it is possible that some jobs are not processed before the estimated deadline. For this case, the framework proposes a kind of degradation in time for the quality of service. Such degradation implies the generation of a new job deadline with a penalty in time according to the quality of service calculated for each job. Such penalty intends to generate larger job deadlines (Jd) for those jobs with low probability of success. Jobs with larger deadlines, imply jobs with a low laxity factor ($LaxityF$) and therefore a low urgency for low quality of service. The function to define the new job deadline is:

$$Jd = time\_now + Jlat * (CONST2 * MF), \forall\, CONST2 >= 2; \quad (4.6)$$

where the factor $(CONST2 * MF)$ intends to introduce larger deadlines to those jobs with low probability of success. The function of the $CONST2$ value is to ensure a minimum value of the new deadline longer than the value of $(time\_now + Jlat)$. In our case, the $CONST2$ value selected was 3. There exist different ways

to define the value of MF. In our case MF is defined as follows:

$$MF = \begin{cases} 0.9 & if \quad QoS \geq 80 \\ 1.1 & if \quad 60 \leq QoS < 80 \\ 1.2 & if \quad QoS < 60 \end{cases} \tag{4.7}$$

Again, the values of the MF factor intends, together with $CONST2$, to generate longer deadlines to those jobs with low probability of success but, keeping a minimum deadline value not too short in relation to the $Jlat$ value.

6. Throughout the experimentation, a comparison of the behavior of the proposed QB-MUF algorithm with respect to two other scheduling approaches is performed. These scheduling approaches are the Minimum Laxity First, denoted as Laxity, and the well known First In First Out (FIFO) scheduling algorithm.

7. In order to provide a fair scenario for all the algorithms under evaluation, all the algorithms will run under the same conditions, penalties, quality of service definitions, including the urgency definition. In this order, the FIFO and laxity algorithms will be generated by changing the values of the weighting factors ($W1,\ W2,\ W3$) properly. The value for the weighting factors will change as follows: For the QB-MUF algorithm,

$$W1 = 0.2,\ W2 = 0.45,\ W3 = 0.35 \tag{4.8}$$

for the FIFO algorithm,

$$W1 = 1.0,\ W2 = 0.0,\ W3 = 0.0 \tag{4.9}$$

and finally for the Laxity algorithm,

$$W1 = 0.0,\ W2 = 0.0,\ W3 = 1.0 \tag{4.10}$$

8. Since the purpose of the *Artifact Recognition* process includes the detection of artifacts (errors) in a *Job* before it is sent to the *Ripping* and the *Press* processes, the treatment of the *Jobs* will be in the following order: first *Artifact Recognition*, second *Ripping* and finally the *Press* process.

9. Initially, in the scenarios for simulation, the faults found at a stage in each *Job* will not be fixed during any of the following stages. However, this could be changed for different scenarios.

10. The Printing process will be the final stage of the simplified model. Since, for our purpose, there is no further information offered by this process than the failure or success of a *Job*, the processing time considered for any *Job* is zero. On the other hand, other processes for reporting purposes will be run during this stage of the framework.

11. Despite that in the simplified model the number of stages considered is always two, the number of sets of resources (*ResourceSet*) per stage can be varied. Moreover the number of resources (*Resource*) inside each *ResourceSet* will also be considered variable.

4.2 Description of Variables

The name and description of those variables considered to be inputs in the simulation are presented in the table 4–2. The results of the scenarios of experimentation discussed in section 4.3 are presented in three different graphical representations as described in the following sub-sections.

4.2.1 Successful *Jobs* on simulation time

This graphic shows on the y-axis the number of successful jobs and on the x-axis the simulation time. The measure of the number of successful Jobs is completed

Table 4–2: Input Variables considered in the framework simulation

Variable	**Description**
Number of Jobs (NJ)	NJ corresponds to the number of *Jobs* that will be generated for processing during simulation.
Arrival rate of Jobs	The arrival rate of *Jobs* for the system will be guided by three parameters: a distribution function (AJ), a mean for the distribution ($AJfm$) and a variance ($AJfv$) in case the selected distribution requires it
Size of the Jobs (SJ)	The size of the *Jobs* arriving to the system will be generated from a Normal Distribution function with a mean $SJfm$ and a variance $SJfv$
Resource processing rate (Rpr)	The processing rate of a resource (Rpr) is the number of units processed per unit time. This rate is used in the function which calculates the processing time for a Job on a resource
Multiply factor (MF)	The multiply factor, MF, works as a correction factor to penalize the processing time of the Jobs according to their quality of service. An example of the definition for the MF is shown in the equation 3.10
Fault threshold (FT)	The fault threshold is a value for each of the 5 faults considered ($FT1$, $FT2$, $FT3$, $FT4$, $FT5$) in the framework. This value is used to compare a random number generated for each fault at each job, if the value is larger than its threshold, then it is considered that the job presents such fault.
Quality of Service factor (QoSF)	The QoSF defines the probability of success for a job. Such QoSF is defined for a function ($QoSF = F(f_1, f_2, ...f_n)$) based on those faults considered for each particular system. The values of each fault f_i are considered as 1 if the fault is present in the Job and 0 if it is not present.
Weighting factor for Job relevance (W1)	$W1$ is a variable included in the Job urgency function; a large value of this $W1$ makes the urgency function give more urgency to those Jobs with a larger criticality factor
Weighting factor for Job QoS (W2)	$W2$ is a variable included in the Job urgency function; a large value of this $W2$ makes the urgency function give more urgency to those Jobs with a larger QoS factor
Weighting factor for Job laxity (W3)	$W3$ is a variable included in the Job urgency function; a large value of this $W3$ makes the urgency function give more urgency to those Jobs with a larger laxity factor
Modulator factor (K) for the Laxity Factor function	The modulator factor (K) is included in the Laxity Factor ($LaxityF$) function to normalize the unbounded outcomes of traditional laxity functions.
Number of ResourceSets (n)	The number of *ResourceSets* n corresponds to the number of sets of resources available to perform a process in a specific stage of the workflow.

periodically, with the same period for all algorithms under evaluation in the framework. The objective of this graphic is to observe the behavior of the metric *number of successful jobs* delivered while the simulation time goes forward. The expected behavior of the metric is that in the QB-MUF algorithm the number of successful jobs delivered increases, especially in the beginning of the simulation, faster than in the other algorithms.

4.2.2 Mean waiting time only for successful *Jobs*

In this graphic, the bars represent the mean waiting time of the successful Jobs in the simulation. The graphic shows a bar with the value of the mean waiting time for each algorithm under evaluation. The expected behavior of the metric mean waiting time of successful jobs, is that in the QB-MUF algorithm the mean waiting time of successful delivered jobs be shorter than the mean waiting time for the successfully delivered jobs in the simulation of the other algorithms.

4.2.3 QoS related to the order of exit of Jobs

In this graphic, the y-axis represents the QoS of the jobs and the x-axis represents the order of exit of jobs in the simulation. Unlike the metric number of successful jobs, this graphic intends to show the QoS (independently if the job was success or failure) of all the delivered jobs. The expected behavior of the graphics is that the jobs with large quality of service get service first than those jobs with low quality of service.

4.3 Scenarios of Experimentation

The following subsections explain the conditions established to run each experiment. Each sub-section also describes the resulting behavior for each algorithm under evaluation according to the graphics described in the sub-sections 4.2.1, 4.2.2 and 4.2.3. For the first experiment (sub-section 4.3.1) a table with the whole set

of variables and its values is presented. Further experiments will show a table with only the most relevant variables of the scenario together with those variables which suffer a change.

4.3.1 Experiment 1

Table 4–3 shows the set of values of those variables considered as inputs in order to run the first experiment. Figures 4–2 and 4–3 correspond to the successful *Jobs* vs the simulation time and to the mean waiting time only for successful *Jobs*, respectively.

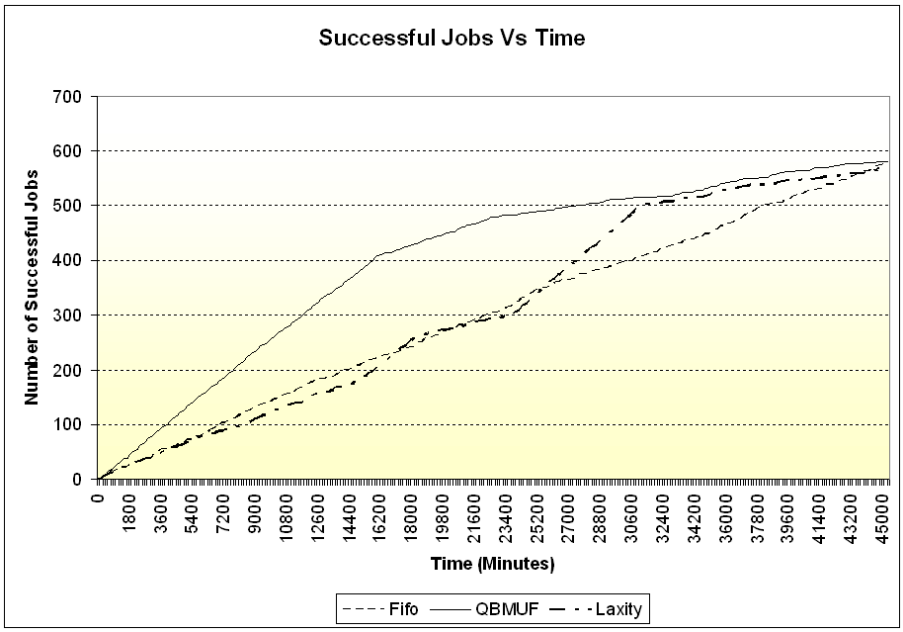

Figure 4–2: Number of successful jobs vs. time in experiment 1

Figure 4–2 shows a better behavior of the QB-MUF algorithm in terms of successful jobs delivered. Such improvement is notorious during certain time of the simulation near the 17,000 units of time. Figure 4–3, shows a better behavior of the QB-MUF algorithm in terms of waiting time for delivered successful jobs.

Table 4–3: Values of the variables considered as inputs in order to run experiment 1

Variable	**Value**
Number of Jobs (NJ)	1000
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 700$ $SJfv = 160$
Resource processing rate (Rpr)	20 (size units / time units)
Multiply factor (MF)	$MF = \begin{cases} 0.9 & if \quad QoS \geq 80 \\ 1.1 & if \quad 60 \geq QoS < 80 \\ 1.2 & if \quad QoS < 60 \end{cases}$
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge f_2 = 1 \end{cases}$
Weighting factor for Job relevance (W1)	$W1 = \begin{cases} 0.2 & for \quad QB - MUF\ algorithm \\ 0.0 & for \quad Laxity\ algorithm \\ 1.0 & for \quad FIFO\ algorithm \end{cases}$
Weighting factor for Job QoS (W2)	$W2 = \begin{cases} 0.45 & for \quad QB - MUF\ algorithm \\ 0.0 & for \quad Laxity\ algorithm \\ 0.0 & for \quad FIFO\ algorithm \end{cases}$
Weighting factor for Job laxity (W3)	$W3 = \begin{cases} 0.35 & for \quad QB - MUF\ algorithm \\ 1.0 & for \quad Laxity\ algorithm \\ 0.0 & for \quad FIFO\ algorithm \end{cases}$
Modulator factor (K) for the Laxity Factor function	40
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

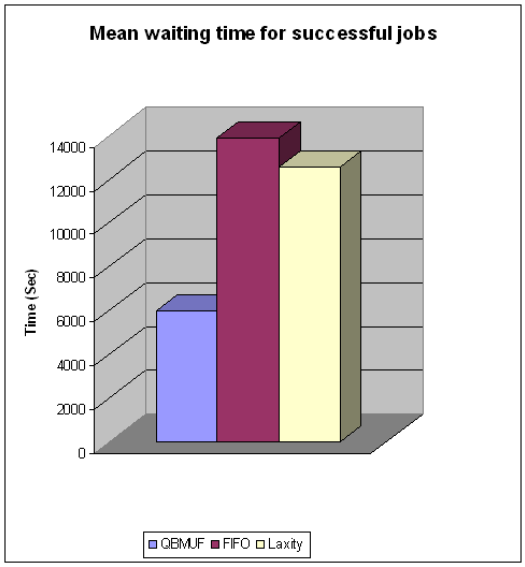

Figure 4–3: Mean waiting time for successful jobs in experiment 1

4.3.2 Experiment 2

In this experiment the number of jobs was reduced from 1000 to 100. The behavior of figures 4–4 and 4–5 was similar to the behavior in experiment 1.

Table 4–4: Values of the variables considered as inputs in order to run experiment 2

Variable	**Value**
Number of Jobs (NJ)	100
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 700$ $SJfv = 160$
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge f_2 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

In this experiment Figure 4–6 shows quality of service (QoS) vs order of execution. This figure verifies that the QB-MUF algorithm, gives a high priority to those jobs with higher quality of service.

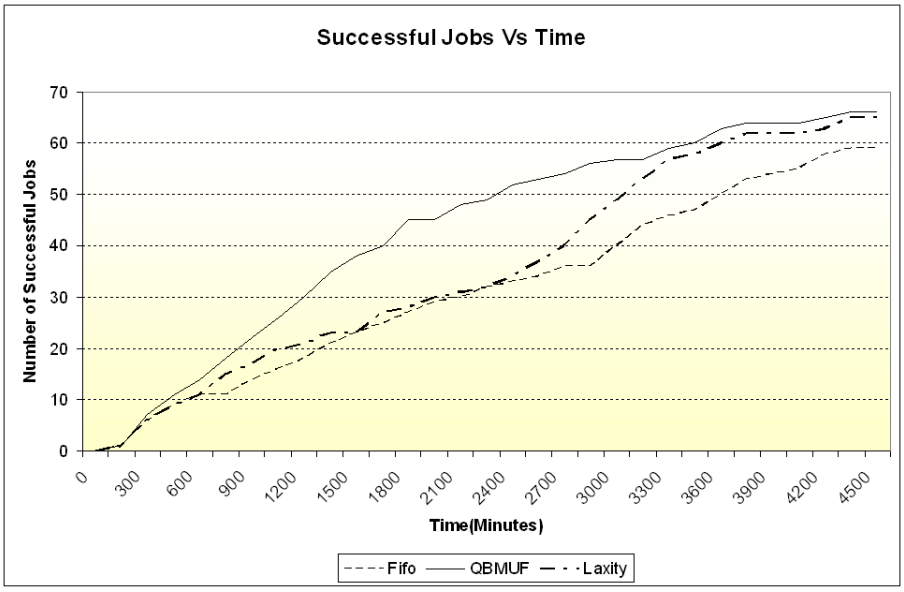

Figure 4–4: Number of successful jobs vs. time in experiment 2

Figure 4–5: Mean waiting time for successful jobs in experiment 2

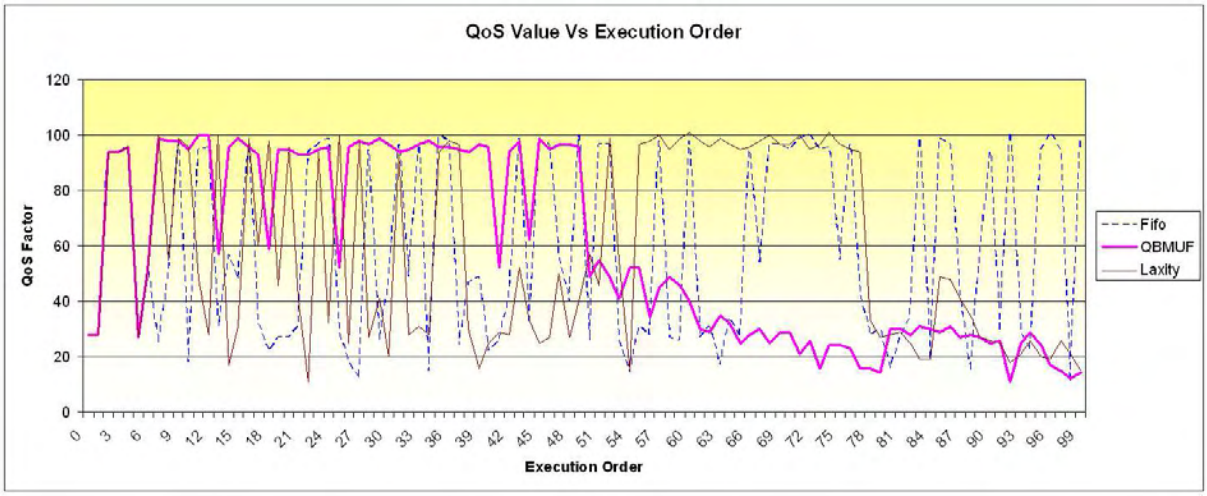

Figure 4–6: Quality of service vs. execution order in experiment 2

4.3.3 Experiment 3

For this experiment, the arrival rate of the jobs was increased in order to generate a large queue in each stage of the framework.

Table 4–5: Values of the variables considered as inputs in order to run the experiment 3

Variable	**Value**
Number of Jobs (NJ)	1000
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 10.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 700$ $SJfv = 160$
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge \ f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge \ f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge \ f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge \ f_2 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

With longer queues the space possible decisions increase for all of the algorithms. The QB-MUF takes advantage of this, which can be noticed by the increase of the gradient of the number of successful jobs delivered in the beginning of the simulation.

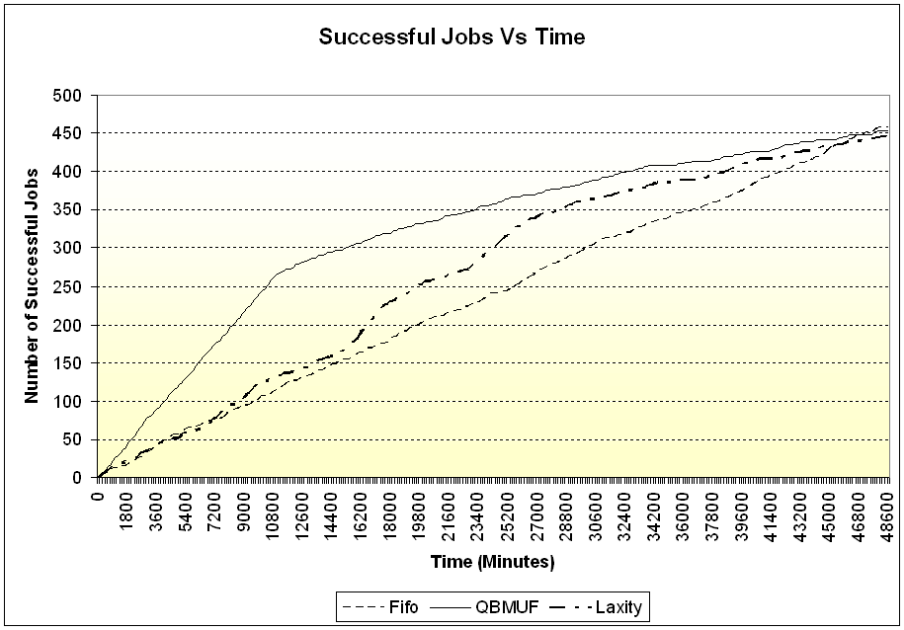

Figure 4–7: Number of successful jobs vs. time in experiment 3

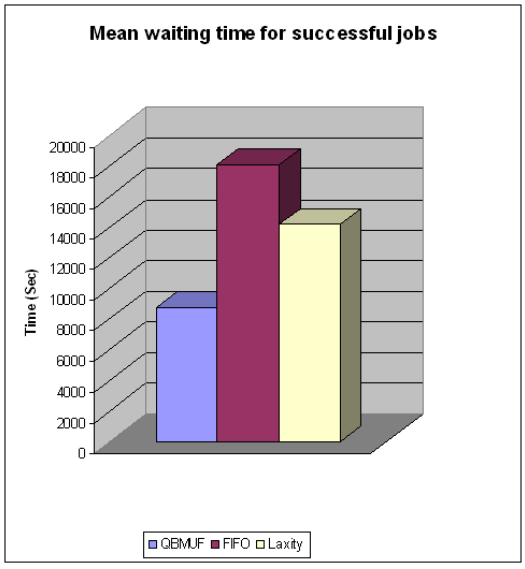

Figure 4–8: Mean waiting time for successful jobs in experiment 3

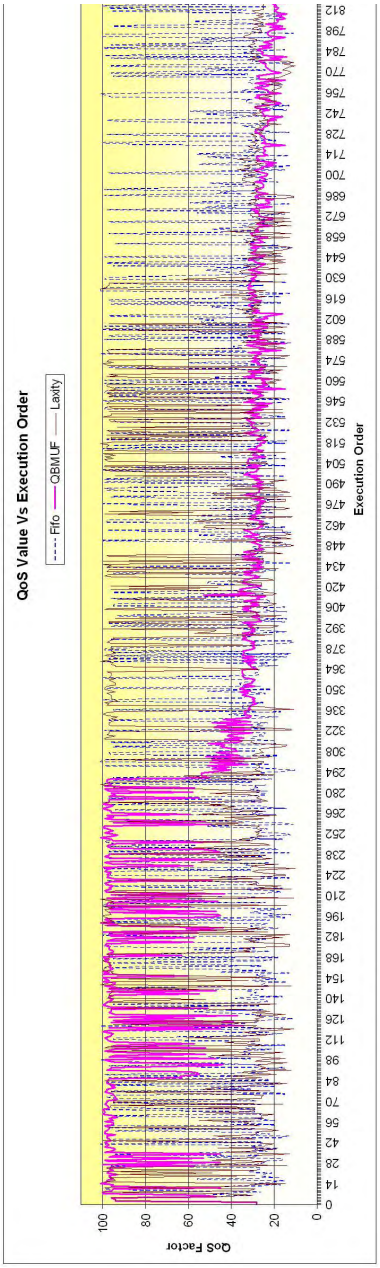

Figure 4–9: Quality of service vs. execution order in experiment 3

4.3.4 Experiment 4

In this experiment the number of jobs was reduced from 1000 to 100, but the arrival rate of the jobs was the same as in experiment 3.

Table 4–6: Values of the variables considered as inputs in order to run the experiment 4

Variable	**Value**
Number of Jobs (NJ)	100
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 10.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 700$ $SJfv = 160$
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge \ f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge \ f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge \ f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge \ f_2 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

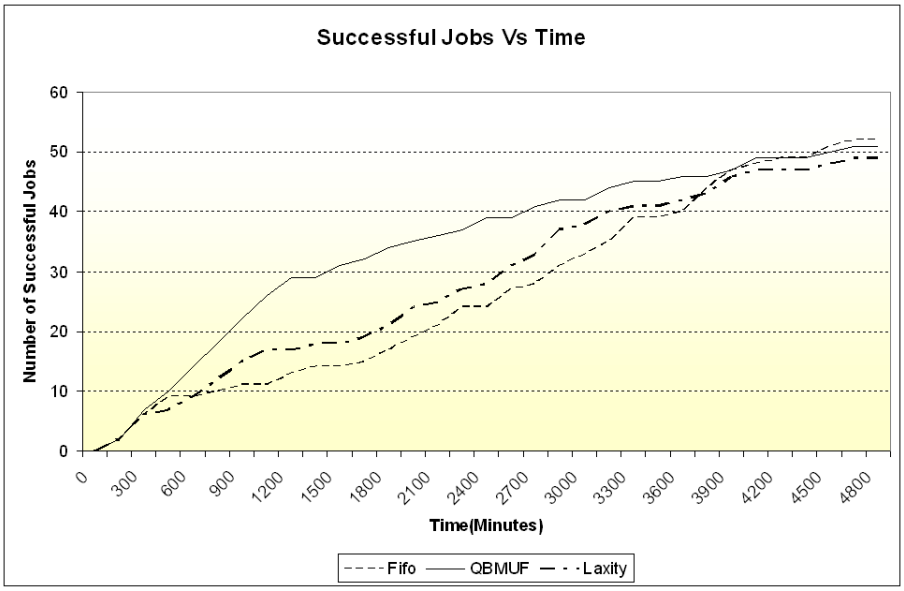

Figure 4–10: Number of successful jobs vs. time in experiment 4

Figure 4–11: Mean waiting time for successful jobs in experiment 4

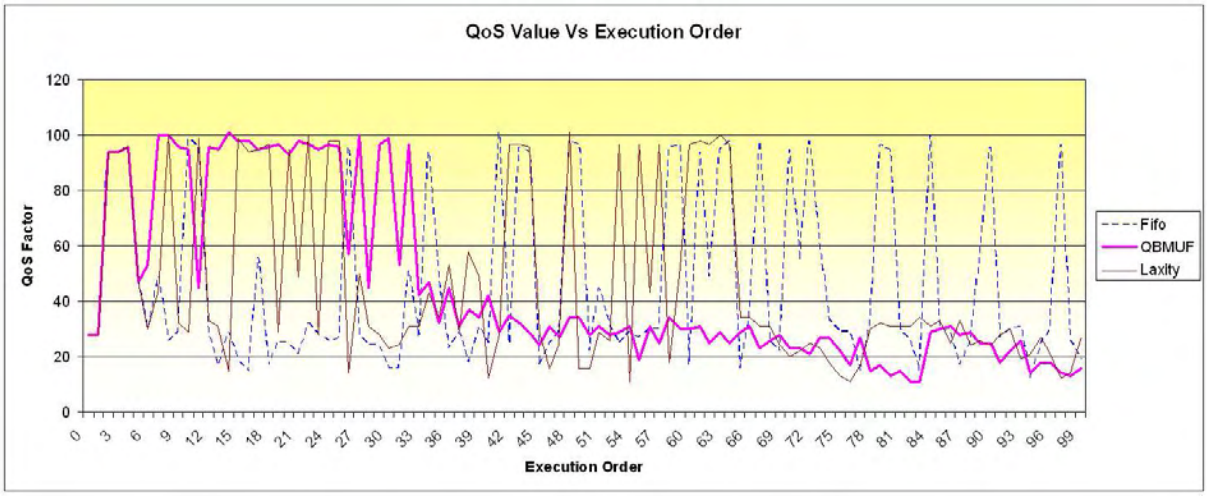

Figure 4–12: Quality of service vs. execution order in experiment 4

4.3.5 Experiment 5

In this experiment the mean of the size of the jobs is chosen larger than in the precious experiments.

Table 4–7: Values of the variables considered as inputs in order to run experiment 5

Variable	**Value**
Number of Jobs (NJ)	1000
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge \ f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge \ f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge \ f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge \ f_2 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

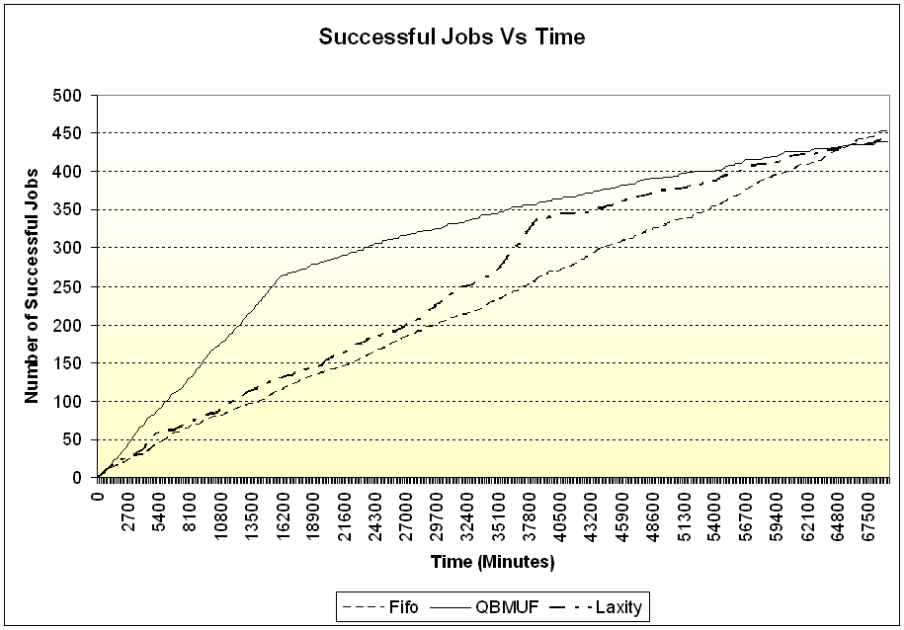

Figure 4–13: Number of successful jobs vs. time in experiment 5

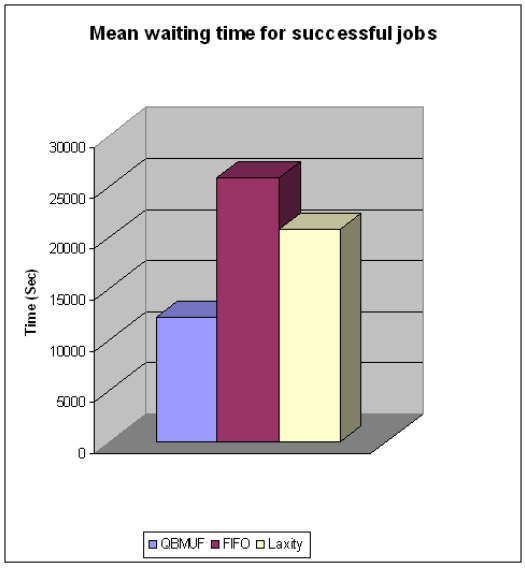

Figure 4–14: Mean waiting time for successful jobs in experiment 5

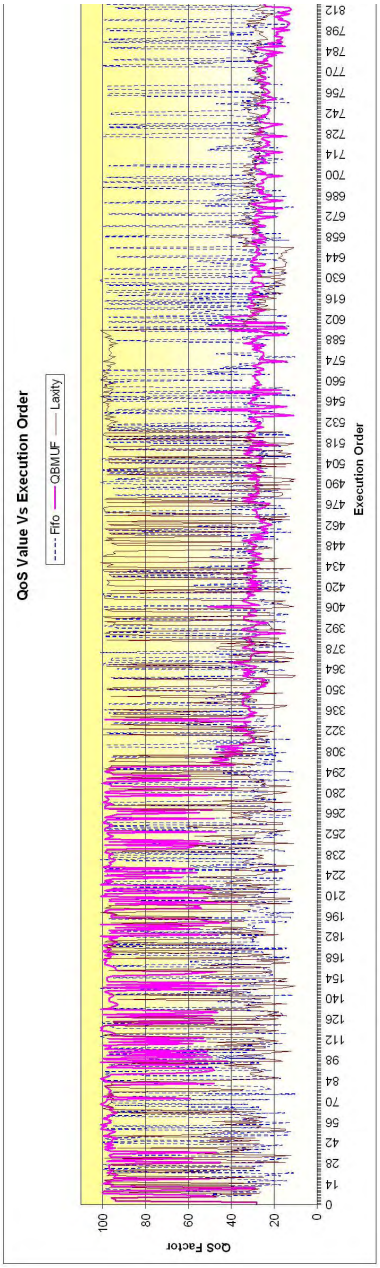

Figure 4–15: Quality of service vs. execution order in experiment 5

4.3.6 Experiment 6

In this experiment the number of jobs was reduced.

Table 4–8: Values of the variables considered as inputs in order to run experiment 6

Variable	**Value**
Number of Jobs (NJ)	100
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge f_2 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

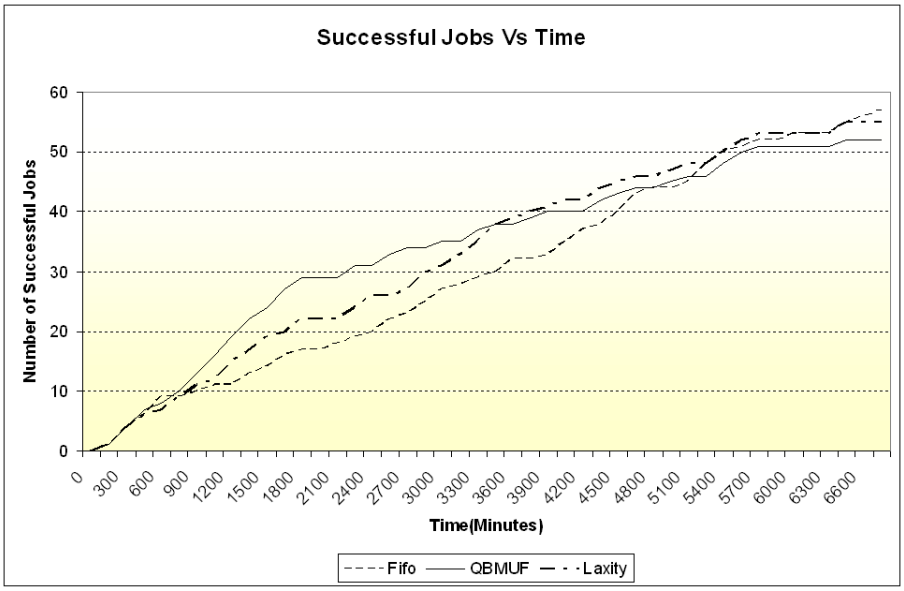

Figure 4–16: Number of successful jobs vs. time in experiment 6

Figure 4–17: Mean waiting time for successful jobs in experiment 6

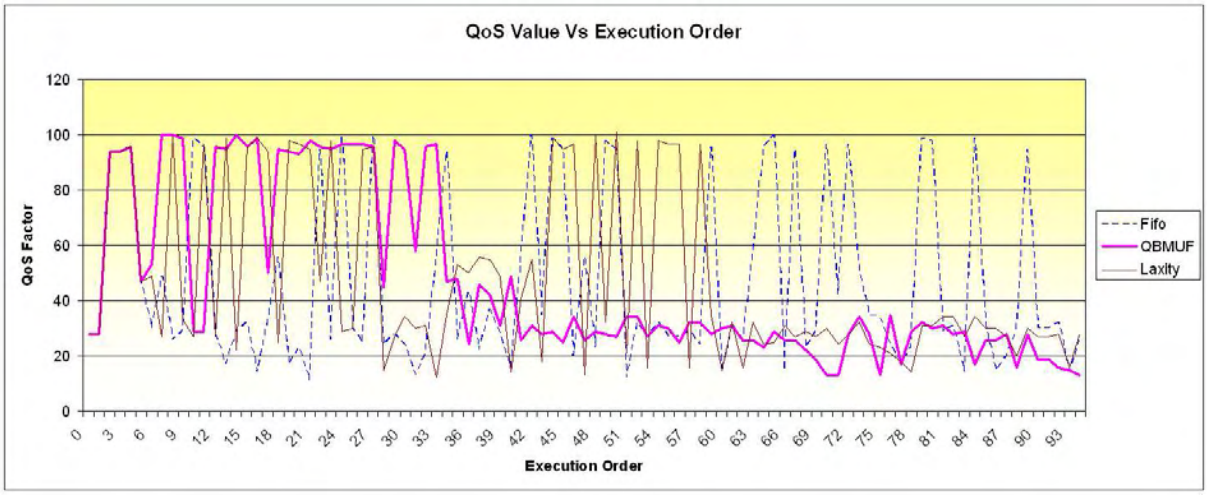

Figure 4–18: Quality of service vs. execution order in experiment 6

4.3.7 Experiment 7

In this experiment the rate of processing of the resources was increased.

Table 4–9: Values of the variables considered as inputs in order to run experiment 7

Variable	**Value**
Number of Jobs (NJ)	1000
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Resource processing rate (Rpr)	25 (size units / time units)
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge \ f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge \ f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge \ f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge \ f_2 = 1 \end{cases}$
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

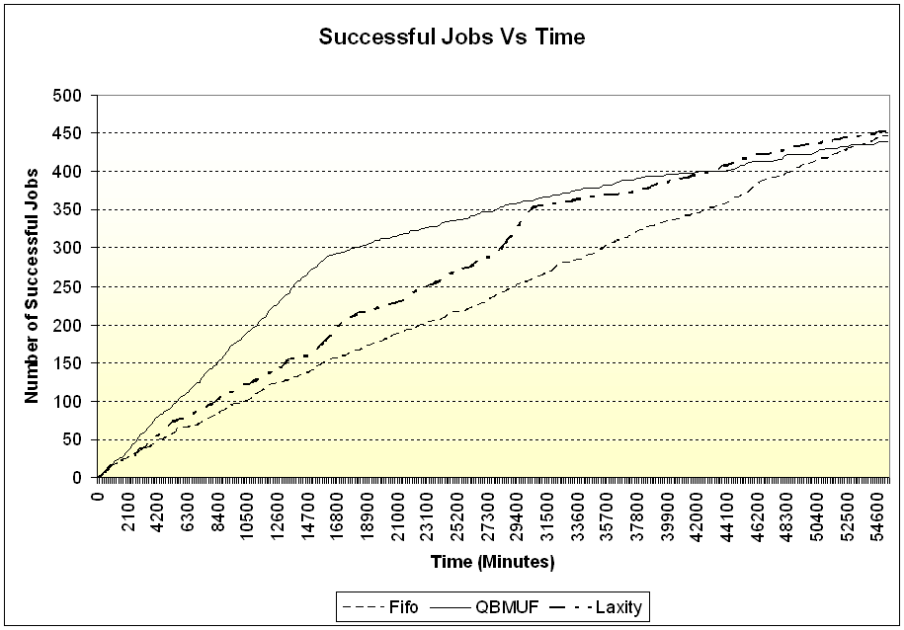

Figure 4–19: Number of successful jobs vs. time in experiment 7

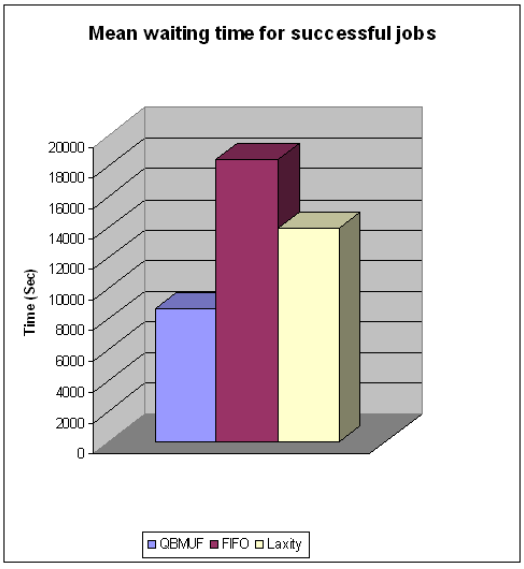

Figure 4–20: Mean waiting time for successful jobs in experiment 7

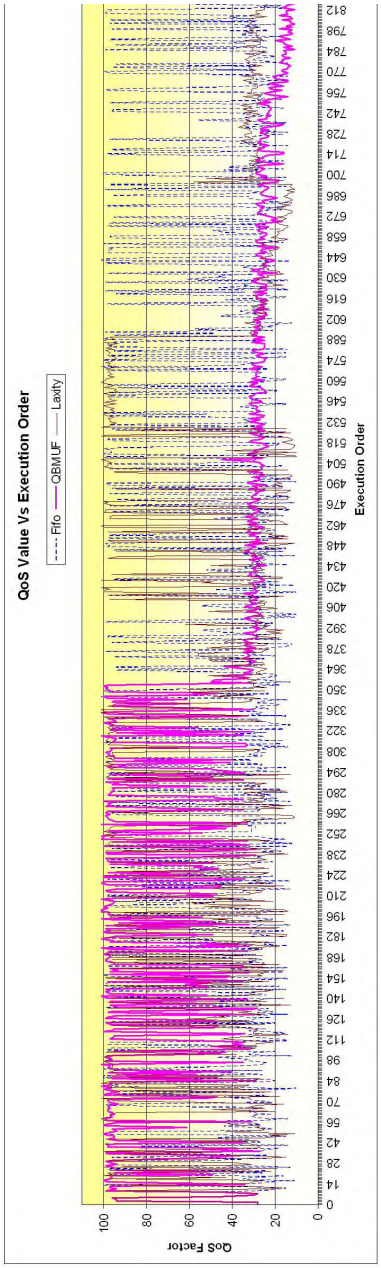

Figure 4–21: Quality of service vs. execution order in experiment 7

4.3.8 Experiment 8

In this experiment the number of jobs was reduced.

Table 4–10: Values of the variables considered as inputs in order to run experiment 8

Variable	**Value**
Number of Jobs (NJ)	100
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Resource processing rate (Rpr)	25 (size units / time units)
Fault threshold (FT)	$FT1 = 67,\ FT2 = 25$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 5] & if \quad f_1 = 0 \wedge f_2 = 0 \\ 60 - [random() * 16] & if \quad f_1 = 1 \wedge f_2 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 0 \wedge f_2 = 1 \\ 50 - [random() * 21] & if \quad f_1 = 1 \wedge f_2 = 1 \end{cases}$
Number of Resource-Sets (n)	two (2) per stage with one (1) resource each.

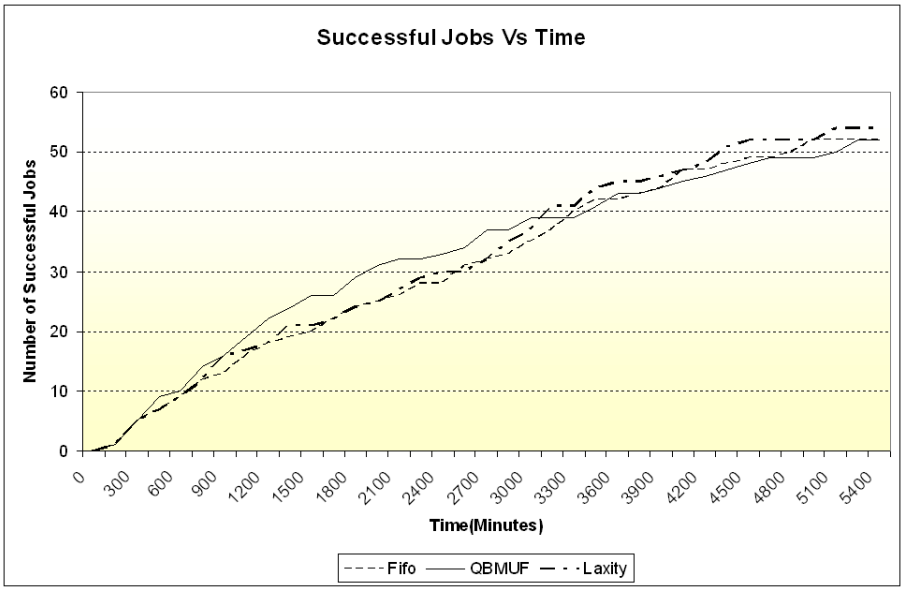

Figure 4–22: Number of successful jobs vs. time in experiment 8

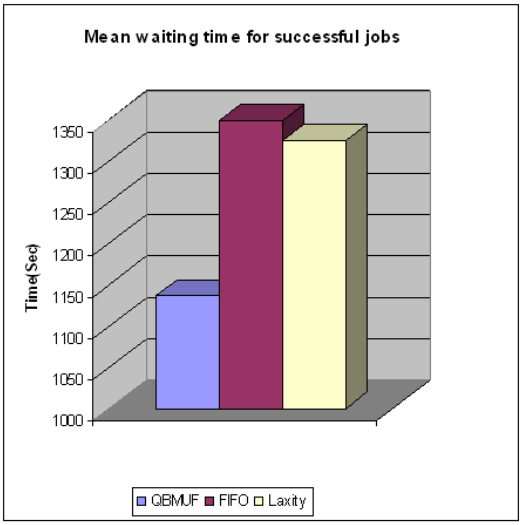

Figure 4–23: Mean waiting time for successful jobs in experiment 8

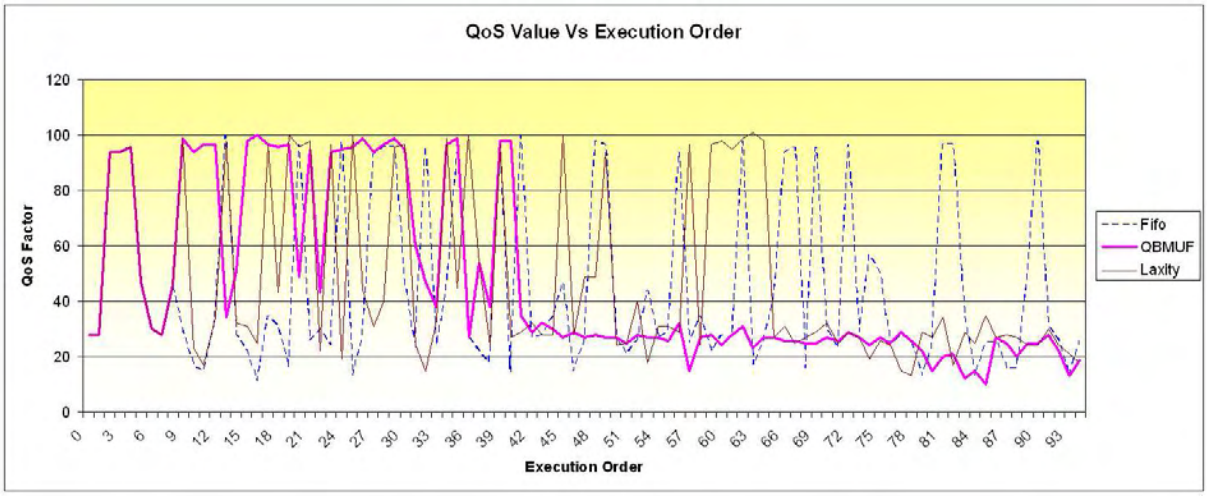

Figure 4–24: Quality of service vs. execution order in experiment 8

4.4 Summary of Results

Two metrics were observed throughout the experimentation: the number of successfully jobs delivered and the mean waiting time of successful delivered jobs. The whole set of experiments ran under the same conditions, except in those explicitly mentioned cases.

In general, results show a reduction of waiting processing time of the QB-MUF over laxity and FIFO approaches in those simulation scenarios where the generated queue is relatively large. The reason is that large queues give to the QB-MUF algorithm a better opportunity to affect the making decision process. In a similar way and for the same reasons, the QB-MUF algorithm exhibits a better behavior in terms of the number of successful jobs over time compared to the traditional approaches.

In order to observe changes in the simulation behavior, different scenarios where generated by inducing changes in simulation parameters. For example, after reducing the number of generated jobs from 1000 to 100, the following changes in metrics behavior were noticed:

- Despite that the mean waiting time is still lower for the QB-MUF algorithm than for the other algorithms, the difference between the mean waiting time of the QB-MUF algorithm and the other algorithms presents a significant reduction.
- The advantage of the QB-MUF algorithm noticed in the graphic of successful jobs on simulation time is less notorious.

The reduction in the number of jobs generated in the simulation yield short waiting queues and thereby less opportunities of acting for the QB-MUF algorithm.

Other changes in simulation parameters include the increase of the arrival rate of jobs (Experiments 3 and 4) and the increase of the mean size of the jobs (Experiments 5 and 6). The increase of these parameters consequently enlarge the queue in the

beginning of the simulation, thereby increasing the opportunity of the QB-MUF for taking decisions. As a result, the advantage of the QB-MUF algorithm is more evident in the graphics of successful jobs on simulation time and mean waiting time.

In Experiments number 7 and 8, the processing rate of the resources together with the mean of the job size were increased in comparison with the Experiments 1 to 4. In experiment 7, the number of jobs (1000) and the increased mean size of the jobs reduce the possible effect of the increased processing rate of resource (reduce the job queue). However, in Experiment 8, the relative small number of jobs used, mixed with the increased processing rate of the resources, produce a scenario with short queues and fast resources that reduces the opportunities of the QB-MUF algorithm for taking action in the decision making process. As a result, the advantage of the QB-MUF algorithm in the graphics of successful jobs on simulation time and mean waiting time of successful delivered jobs were considerably reduced, but it was still an advantage.

In almost all the scenarios, except the experiment 8, the graphic of successful jobs on simulation time presents a point near the first third part of the simulation time, where the curve shows an evident change in its gradient. This point can be considered as the point where the QB-MUF have done its best effort on pumping the jobs with better QoS toward their execution in a resource. The decreasing of the curve gradient is because after this point the number of remaining jobs in queue has low QoS. At this point, the number of arriving jobs with god QoS compared to the number of jobs with low QoS remaining still in queue, is not enough to maintain sending jobs with good QoS to execution in a resource.

CHAPTER 5
QOS BASED DYNAMIC SCHEDULING APPLIED TO GRID COMPUTING

5.1 A Grid Environment

The availability of powerful computers and high-speed network technologies as low-cost commodity components has changed the way we solve large scale problems. These technology opportunities have led to the possibility of using geographically distributed computers as a single, unified computing resource. Grid computing enables coordination, storage and networking of resources across geographically dispersed organizations in a transparent way for users. The first generation of grid technologies has demonstrated the feasibility of grids for addressing challenging large scale problems (e.g. Grid Physics Network (GridPhyN)[1] and Teragrid[2] among many others). Next generation of grid applications will be increasingly dynamic. This implies that the current static infrastructures will not be adequate unless adaptive functionalities are provided.

For example, the Wide Area Large Scale Automated Information Processing (WALSAIP)[3] project aims at developing an infrastructure for the treatment of signal-based information arriving from physical sensors in a wide-area, large scale

[1] http://www.griphyn.org/

[2] http://www.teragrid.org/

[3] http://walsaip.ece.uprm.edu

environment. As illustrated in figure 5–1, signals are acquired from sensors, through a sensor array structure (SAS), on one end of the model and sequentially treated until information is extracted and delivered at the other end of the model structure to an information rendering system (IRS). This standard model structure is improved with the formulation of a new conceptual model which accentuates a distributed space-time processing format, which permeates all other system substructures such as distributed sensor networks for signal acquisition, distributed databases for database management, and distributed signal processing, as well as overall distributed computing.

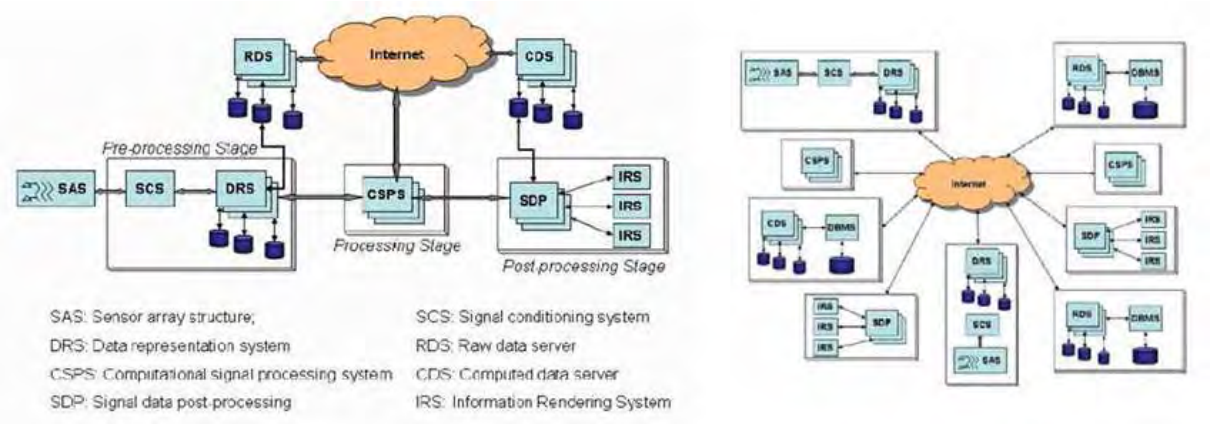

Figure 5–1: Conceptual framework for wide area large scale automated information processing

The conceptual framework of WALSAIP centers primarily on the manner in which systems, tools, and applications are being integrated, under a computing and information processing (CIP) environment infrastructure to deliver end-to-end information relevant to users through tailored requests. Formally, CIP environment deals with the algorithmic treatment of signal-based large scale content in order to extract information relevant and important to a user. In this regard a CIP environment can be thought of as the aggregate of the following seven (7) components: A

set of input entities, a set of output entities, a database infrastructure, a set of generalized computing and information processing operators, a set of composition rules for these operators, a set of actions rules for the operators to act on input entities in order to produce targeted and desired output entities, and a user interface.

We focus on a specific component of the CIP environment, the computing and information processing operators for data images. The operators are in charge of the treatment of sensing data information (Data images specifically). An image operator takes an image and performs certain tasks such as noise reduction, edge detection, feature enhancement, spectral analysis, etc.

In order to build a simplified model, which makes possible to observe the behavior of our scheduling framework based on the QB-MUF algorithm in a grid computing environment, the composition of two specific data images unary operators[4] were used.

The simplified model was conformed by taking the Sharpen and Emboss operators [53], under the following assumptions:

1. The composition of the two selected data operators, should be performed always in the same order. The sharpen operator followed by the emboss operator.

2. Despite that in the simplified model the number of stages considered is always two. The number of sets of resources (*ResourceSet*) per stage can be varied. Moreover, the number of resources (*Resource*) inside each *ResourceSet* is considered variable.

3. Each framework *ResourceSet* will represent a LAN in the scope of the grid environment selected.

[4] Unary operators are those that operates over just one image.

4. Each *ResourceSet* will contain a number s of available *Resources* to perform an operator over a job. All *Resources* inside a *ResourceSet* are considered as homogeneous resources. This means that the resources have the same processing rate.

5. The whole set of *Resourcesets* (LANs) represents a Wide Area Network.

6. Different LANs, performing the same operator can operate with different rate of processing.

7. Preemption is not considered as a possibility in the model. When a job arrives to a *Resource*, the job will use the whole resource processing capacity available.

8. Since the size of a *Job* is an important factor for operators, it is considered into the definition of the presence of a fault in the *Job*.

9. The criterion used to select a path between the existing *Resourcesets* is a function which combines two factors: the completion ratio of a *Resourceset* and the QoS mean of the waiting queue of each *Resourceset*.

10. For all the stages defined in the framework, laxity factor ($LaxityF$) is defined by equation 3.9. The variable job latency ($Jlat$) is indirectly used in this definition. The definition of $Jlat$, for all of the algorithms under evaluation, includes a penalty in processing time according to the probability of failure of each job. The equation for $Jlat$ is defied as follows:

$$Jlat = \frac{Js}{Rpr} * (1 + CONS * (\frac{JPF}{100})), \tag{5.1}$$

where $CONS = 1$ for the experiments presented in this chapter. However the $CONS$ value can be modified for each particular scheduling problem.

11. According to the characteristics of each scenario, it is possible that some jobs are not processed before the estimated deadline. For this case, the framework

proposes a kind of degradation in time for the quality of service. Such degradation implies the generation of a new job deadline with a penalty in time according to the quality of service calculated for each job. Such penalty intends to generate larger job deadlines (Jd) for those jobs with low probability of success. Jobs with larger deadlines, imply jobs with a low laxity factor ($LaxityF$) and therefore a low urgency for low quality of service. The function to define the new job deadline ii:

$$Jd = time\_now + (Jlat \ * \ CONST2 \ * \ MF) \tag{5.2}$$

where MF is defined as follows:

$$MF = \begin{cases} 0.9 & if \quad QoS \geq 80 \\ 1.1 & if \quad 60 \geq QoS < 80 \\ 1.2 & if \quad QoS < 60 \end{cases} \tag{5.3}$$

12. Throughout the experimentation, is done a comparison of the behavior of the proposed QB-MUF algorithm with respect to two other scheduling approaches: The Minimum Laxity First, denoted as Laxity, and the well known First In First Out (FIFO) scheduling algorithm.

13. In order to provide a fair scenario for all the algorithms under evaluation, all the algorithms will run under the same conditions, penalties, quality of service definitions, including the urgency definition. In this order, the FIFO and laxity algorithms will be generated by changing the values of the weighting factors ($W1$, $W2$, $W3$) properly. The value for the weighting factors will change as follows: For the QB-MUF algorithm,

$$W1 \ = \ 0.2, \ \ W2 \ = \ 0.45, \ \ W3 \ = \ 0.35 \tag{5.4}$$

for the FIFO algorithm,

$$W1 = 1.0, \quad W2 = 0.0, \quad W3 = 0.0 \tag{5.5}$$

and finally for the Laxity algorithm,

$$W1 = 0.0, \quad W2 = 0.0, \quad W3 = 1.0 \tag{5.6}$$

14. Initially, in the scenarios for simulation, the faults found at a stage in each *Job* will not be fixed during any of the following stages. However, this could be changed for different scenarios.

5.2 Description of Variables

The name and description of those variables considered to be inputs in the simulation are those already defined in the table 4–2. The results of the scenarios of experimentation discussed in section 5.3 are presented by the three same graphical representations described in chapter 4, sections 4.2.1, 4.2.2 and 4.2.3.

5.3 Scenarios of Experimentation

The following subsections explain the conditions established to run each experiment. Each sub-section also describes the resulting behavior for each algorithm under evaluation according to the graphics described in the sub-sections 4.2.1, 4.2.2 and 4.2.3. For the first experiment, a table with the whole set of variables and its values is presented. Further experiments will show a table with only the most relevant variables of the scenario together with those variables which suffer a change.

5.3.1 Experiment 1

Table 5–1 shows the set of values of those variables considered as inputs in order to run the first experiment. Figures 5–2 and 5–3 correspond to the successful

Jobs vs the simulation time and to the mean waiting time only for successful *Jobs*, respectively.

Table 5–1: Values of the variables considered to be inputs in order to run the experiment 1

Variable	**Value**
Number of Jobs (NJ)	1000
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Resource processing rate (Rpr)	20 (size units / time units)
Multiply factor (MF)	$MF = \begin{cases} 0.9 & if \quad QoS \geq 80 \\ 1.1 & if \quad 60 \geq QoS < 80 \\ 1.2 & if \quad QoS < 60 \end{cases}$
Fault threshold (FT)	$FT1 = SJfm = 1000$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 10] & if \quad f_1 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 1 \end{cases}$
Weighting factor for Job relevance (W1)	$W1 = \begin{cases} 0.2 & for \quad QB - MUF\ algorithm \\ 0.0 & for \quad Laxity\ algorithm \\ 1.0 & for \quad FIFO\ algorithm \end{cases}$
Weighting factor for Job QoS (W2)	$W2 = \begin{cases} 0.45 & for \quad QB - MUF\ algorithm \\ 0.0 & for \quad Laxity\ algorithm \\ 0.0 & for \quad FIFO\ algorithm \end{cases}$
Weighting factor for Job laxity (W3)	$W3 = \begin{cases} 0.35 & for \quad QB - MUF\ algorithm \\ 1.0 & for \quad Laxity\ algorithm \\ 0.0 & for \quad FIFO\ algorithm \end{cases}$
Modulator factor (K) for the Laxity Factor function	40
Number of Resource-Sets (n)	two (2) per stage. The first one with one (1) resource and the second one with two (2) resources.

Similarly to Experiments in chapter 5, Figure 5–2 shows a better behavior of the QB-MUF algorithm in terms of successfully jobs delivered. Again, such improvement is notorious during certain time of the simulation near the first 17,000 units of time.

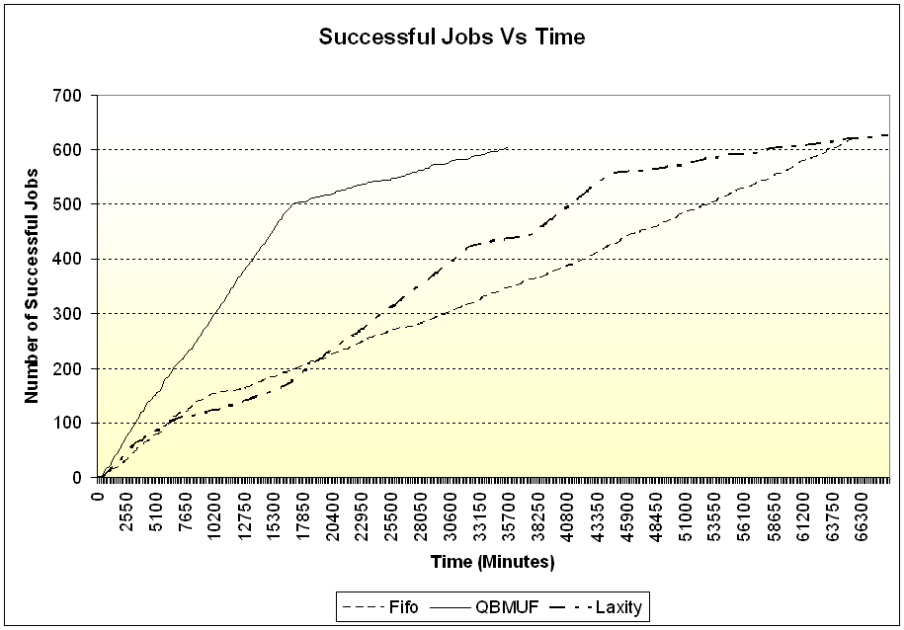

Figure 5–2: Number of successful jobs vs. time in experiment 1

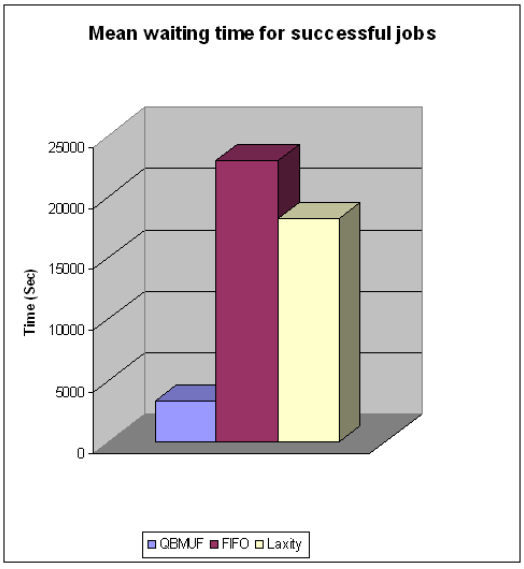

Figure 5–3: Mean waiting time for successful jobs in experiment 1

Moreover, the QB-MUF finish deliver the whole set of successfully jobs just after almost the 51% of the time consumed by the others algorithms. Figure 5–3, shows a better behavior of the QB-MUF algorithm in terms of waiting time for delivered successful jobs.

5.3.2 Experiment 2

In this experiment the number of jobs was reduced from 1000 to 100. The behavior of Figures 5–4 and 5–5 was similar to the behavior in experiment 1.

Table 5–2: Values of the variables considered as inputs in order to run experiment 2

Variable	**Value**
Number of Jobs (NJ)	100
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 15.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Fault threshold (FT)	$FT1 = SJfm = 1000$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 10] & if \quad f_1 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage. The first one with one (1) resource and the second one with two (2) resources.

In this experiment Figure 5–6 shows quality of service (QoS) vs order of execution. Despite that the QB-MUF algorithm continues outperforming the other two algorithms by giving a high priority to those jobs with higher quality of service, Figure 5–6 shows that for this scenario some jobs with low quality of service are served together with the service expected for jobs with high quality of service.

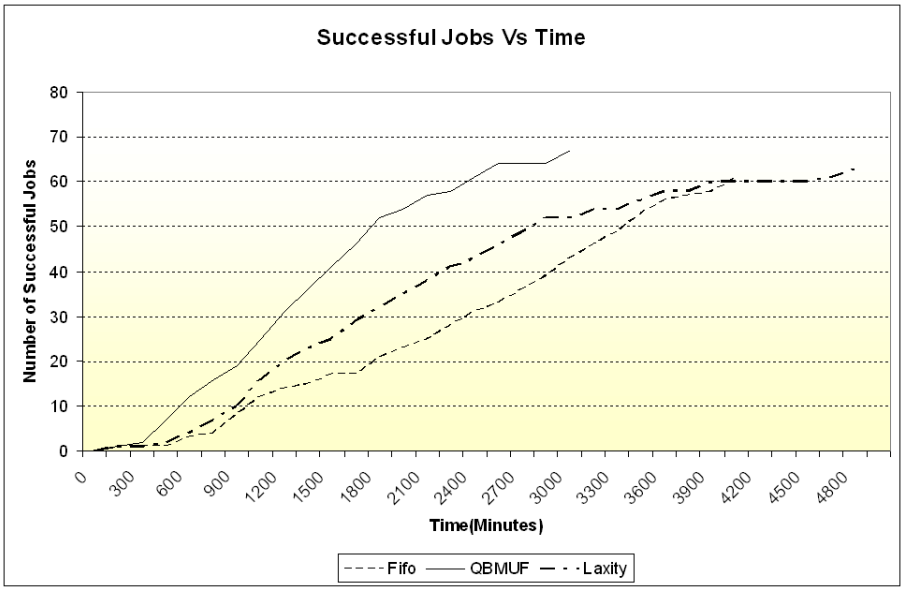

Figure 5–4: Number of successful jobs vs. time in experiment 2

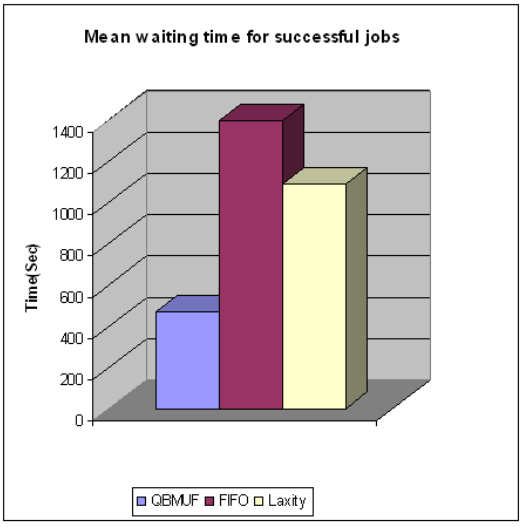

Figure 5–5: Mean waiting time for successful jobs in experiment 2

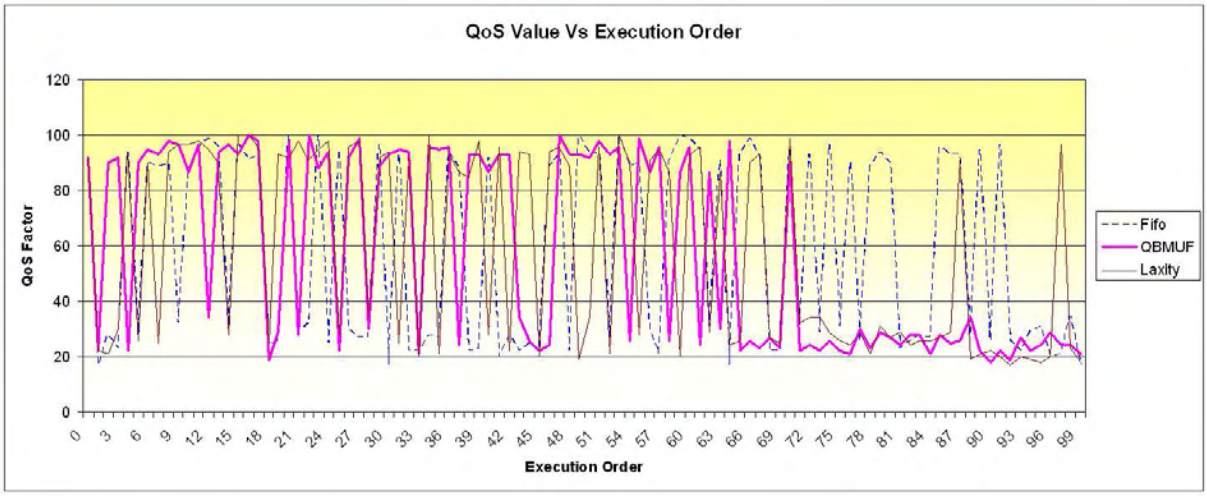

Figure 5–6: Quality of service vs. execution order in experiment 2

5.3.3 Experiment 3

For this experiment, the arrival rate of the jobs was increased in order to generate a large queue in each stage of the framework.

Table 5–3: Values of the variables considered as inputs in order to run the experiment 3

Variable	**Value**
Number of Jobs (NJ)	1000
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 10.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Fault threshold (FT)	$FT1 = SJfm = 1000$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 10] & if \quad f_1 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage. The first one with one (1) resource and the second one with two (2) resources.

As in Experiments of chapter 4, with larger queues the space possible decisions increase for all of the algorithms. The QB-MUF takes advantage of this, which can be noticed by the increase of the gradient of the number of successful jobs delivered in the beginning of the simulation to near the first 13500 units of time. Again, some jobs with low quality of service are served together with jobs with high quality of service with more frequency than in Experiments of chapter 4.

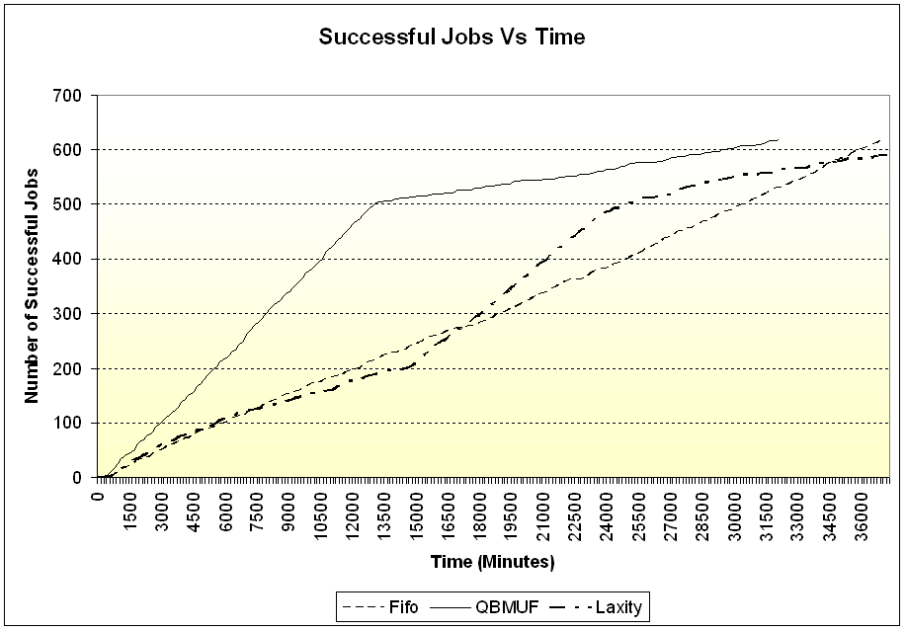

Figure 5–7: Number of successful jobs vs. time in experiment 3

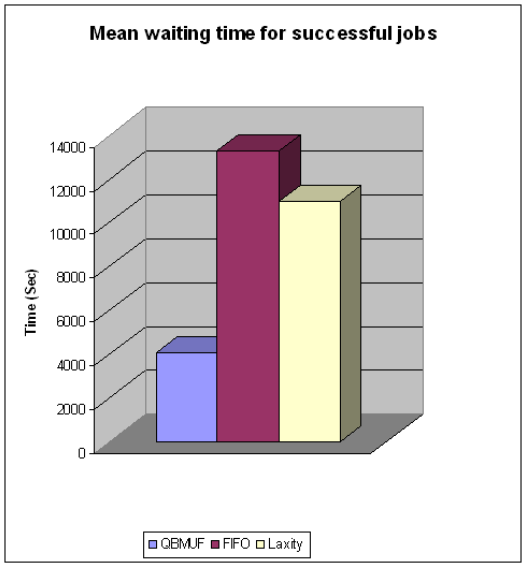

Figure 5–8: Mean waiting time for successful jobs in experiment 3

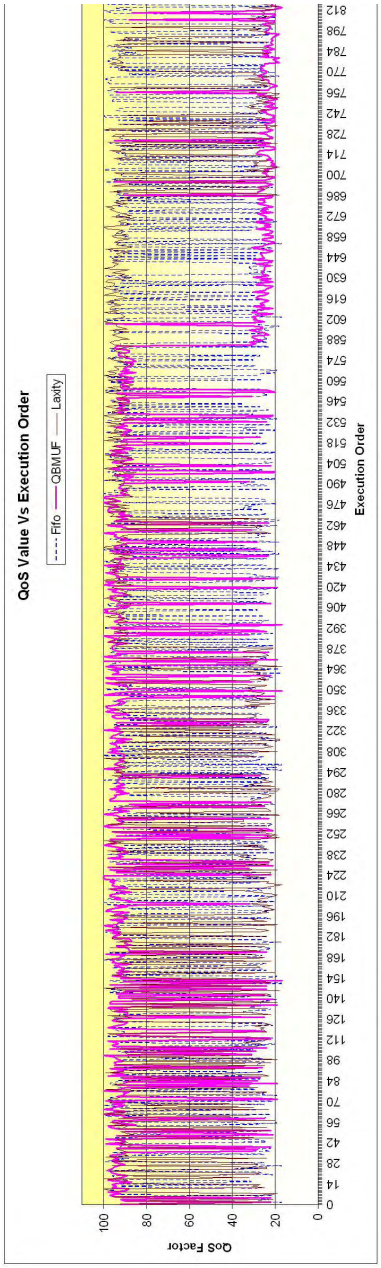

Figure 5–9: Quality of service vs. execution order in experiment 3

5.3.4 Experiment 4

In this experiment the number of jobs was reduced from 1000 to 100, but the arrival rate of the jobs was the same as in experiment 3.

Table 5–4: Values of the variables considered as inputs in order to run the experiment 4

Variable	**Value**
Number of Jobs (NJ)	100
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 10.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Fault threshold (FT)	$FT1 = SJfm = 1000$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 10] & if \quad f_1 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 1 \end{cases}$
Modulator factor (K)	40
Number of Resource-Sets (n)	two (2) per stage. The first one with one (1) resource and the second one with two (2) resources.

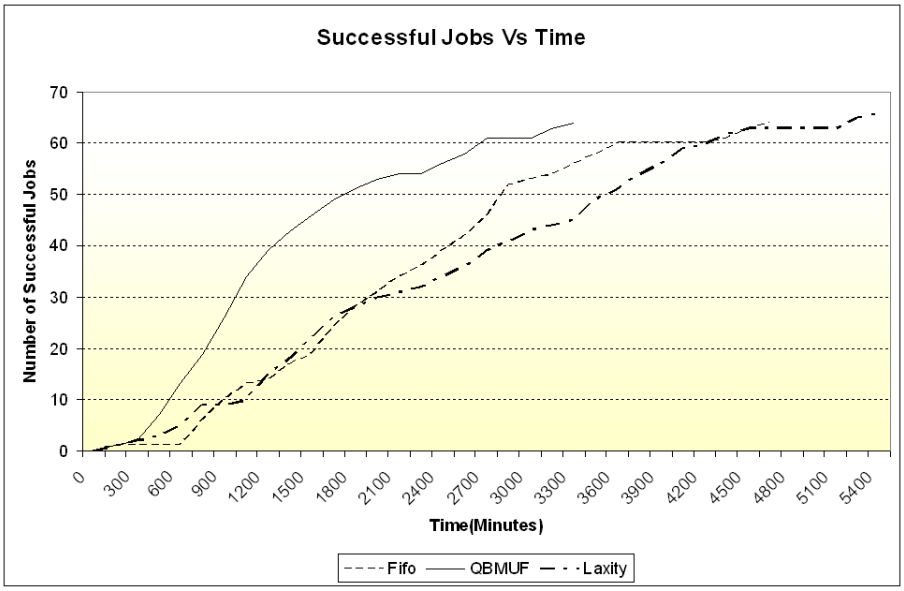

Figure 5–10: Number of successful jobs vs. time in experiment 4

Figure 5–11: Mean waiting time for successful jobs in experiment 4

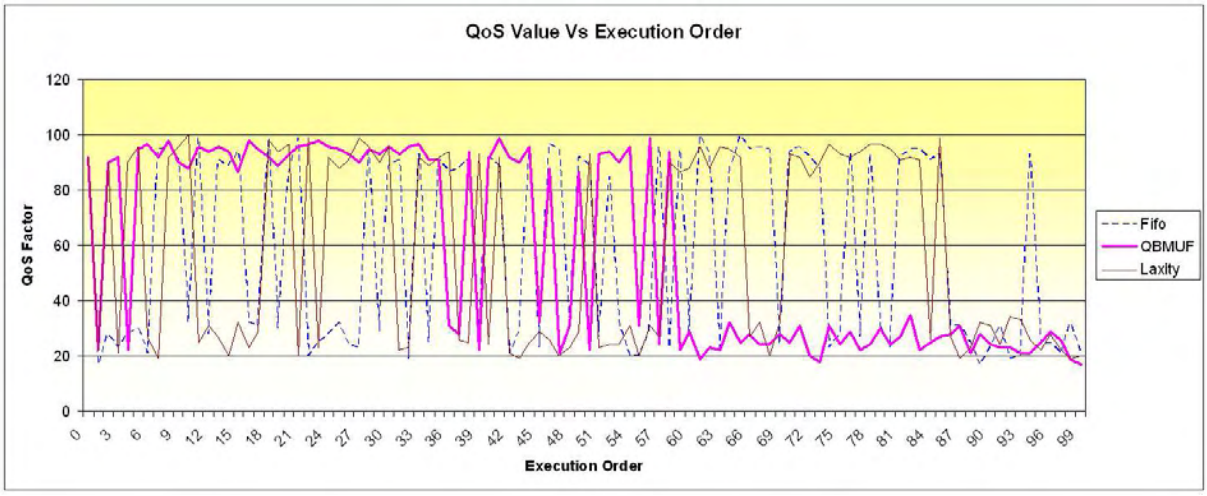

Figure 5–12: Quality of service vs. execution order in experiment 4

5.3.5 Experiment 5

In this Experiment the number of *Resources* in each *ResourceSet* is increased by 1 . The arrival rate of the jobs was the same that in Experiments 3 and 4.

Table 5–5: Values of the variables considered as inputs in order to run Experiment 5

Variable	**Value**
Number of Jobs (NJ)	1000
Arrival rate of Jobs	$AJ = negative\ exponential\ function$ $AJfm = 10.5$ $AJfv = n/a$
Size of the Jobs (SJ)	$SJ = normal\ distribution\ function$ $SJfm = 1000$ $SJfv = 200$
Fault threshold (FT)	$FT1 = SJfm = 1000$
Quality of Service factor (QoSF)	$QoSF = \begin{cases} 100 - [random() * 10] & if \quad f_1 = 0 \\ 80 - [random() * 21] & if \quad f_1 = 1 \end{cases}$
Modulator factor (K)	40
Number of ResourceSets (n)	two (2) per stage. The first one with two (2) resources and the second one with three (3) resources.

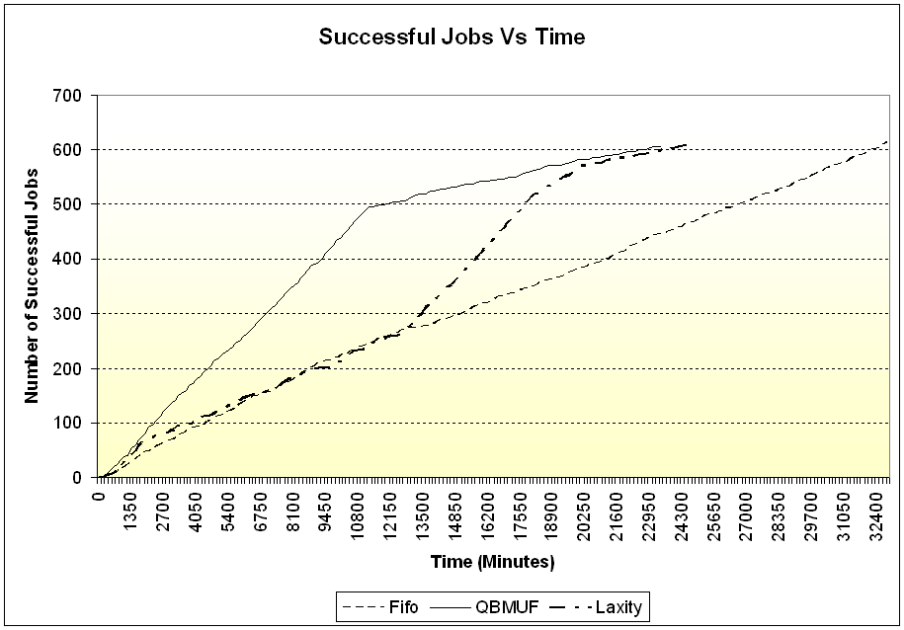

Figure 5–13: Number of successful jobs vs. time in experiment 5

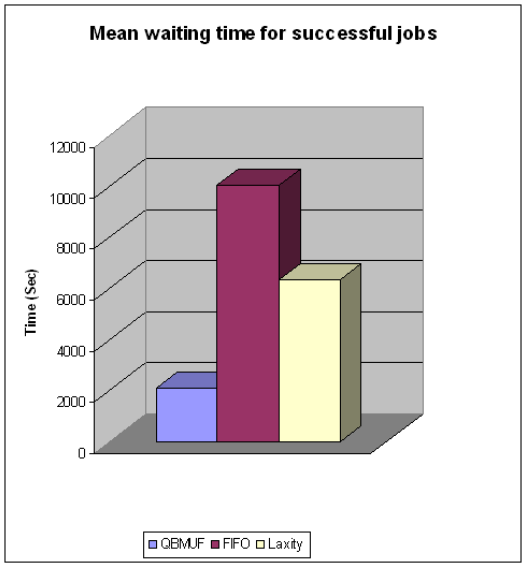

Figure 5–14: Mean waiting time for successful jobs in experiment 5

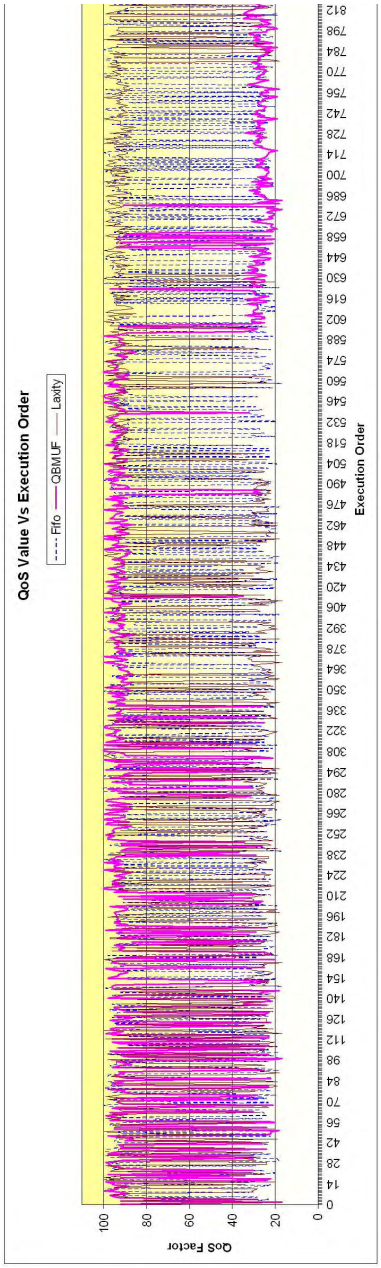

Figure 5–15: Quality of service vs. execution order in experiment 5

5.4 Summary of Results

The same metrics observed in chapter 4, the number of successfully jobs delivered and the mean waiting time of successful delivered jobs, were observed for the experiments in this chapter. Additionally, a new criteria for selecting the best next stage for a job was implemented in the framework. This new criteria take into account the performance of each *ResourceSet* as well as the QoS of queued jobs in each *ResourceSet*.

Similar to the results in chapter 4, simulations under chapter 5 conditions show a reduction of waiting processing time of the QB-MUF over laxity and FIFO approaches, but this time the difference observed was more notorious, up to 4 times the mean waiting time of the QB-MUF algorithm.

After inducing changes in simulation parameters such reducing the number of generated jobs from 1000 to 100, difference between the mean waiting time of the QB-MUF algorithm and the other algorithms presents a significant reduction and the advantage of the QB-MUF algorithm noticed in the graphic of successful jobs on simulation time is less notorious, similarly as occurred in chapter 4 simulations.

In all the simulated scenarios, the graphic of successful jobs on simulation time shows that the whole set of successfully finished jobs where completely served in less time by the QB-MUF algorithm than by the other algorithms. The reason is the additional use of the new next stage selection criteria described above.

The induction of heterogeneous and faster *ResourceSets* in the simulation scenarios, produces a special behavior in the graphics of QoS related to the order of exit of Jobs. Such Figures show that for this scenarios, some jobs with low quality of service are served together with the service expected for jobs with high quality of service. The reason is basically that heterogeneous sets of resources with higher processing rates may produce variable size waiting queues in resources letting sometimes that a *ResoruceSet* serves first a job with low quality of service.

CHAPTER 6
CONCLUSIONS AND FUTURE WORKS

The design and development of a framework that implements a distributed dynamic scheduling methodology based on quality of service, has been discussed in this thesis. Such framework includes mechanisms to take into account contingency and priority fluctuations. It also includes the implementation of the QB-MUF algorithm, inside each local scheduler, which gives high priority to jobs with low probability of failure according to suitable failure probabilities calculated for jobs on particular application environments.

To address the issue of giving high priority to jobs with low probability of failing, a new urgency criteria equation is introduced which accounts for relevance, laxity and probability of failures of incoming jobs. The probability of failure of a job is estimated according to the occurrence of faults in a job, because of the presence of faults or the combination of some of these faults can eventually lead to errors, depending on their severity.

To deal with the priority fluctuation of jobs, the local schedulers uses the new urgency criteria definition and maintains a sorted local waiting job queue, constantly updated to detect priority fluctuations in time.

To deal with resource contingencies (unexpected unavailable resources), each local scheduler manages a list of active and free resources where jobs are sent. This list is constantly maintained by the same resources through communication messages indicating their state.

SimJava was used to implement the first version of the proposed dynamic scheduling framework. The components of the framework architecture together with their behavior were mapped into entities of the *SimJava* class `Sim_entity`. Also, a main program which encases the logic to build the representation of a whole scheduling environment was implemented. This framework supports different scheduling algorithms including FIFO and a laxity based algorithm besides the proposed QB-MUF. Another feature of the implemented framework is the possibility of implementing other scheduling methodologies by implementing the abstract class `LocalScheduler` but restricted to use the job characteristics pre-defined on the framework.

To our knowledge, this is the first time that the mapping of job meta-data properties into quality of service metrics is considered in the decision making process of a dynamic scheduling strategy.

6.1 Conclusions

- The simulation results indicate that a dynamic scheduling algorithm may be sensitive to different parameters including internal job properties indicating its quality.
- The definition of QoS metrics based on intrinsic properties of jobs opens an opportunity to job scheduling strategies that improves the number of successful finished jobs in a workflow.
- It has been demonstrated that the inclusion of a second dynamic factor, based on QoS, in the existing MUF, and the combination of this QoS with a laxity factor, provide a good mechanism to generate a balanced scheduling for quality and deadline fulfillment.

- The implementation of the QB-MUF algorithm and the definition of quality of service metrics provides indeed a framework to enable dynamic scheduling.

6.2 Future Work

Along with the research process, interesting new questions and visions of improvement for the framework were exposed. These visions and questions can be drawn as future research work as follows:

- One focus of this research was the quality of service of a job from the perspective of its internal properties. An interesting question is how this methodology interacts with other different quality of service methodologies which are based on external requirements of jobs.
- Most dynamic scheduling strategies for distributed environments include a global or central scheduler. Another interesting exercise may be to divide the global scheduler into small autonomous entities and then distribute them in the whole space of the environment.
- To maintain a fair scenario for experimentation, some functions required for the proposed strategy, such as the penalty, were used for and considered as side effects on the evaluation of the other algorithms. A good exercise would consist on focusing on those penalization functions, and observe and isolate the effects of generating and scheduling with only penalty functions based on quality of service as considered for this research.
- An interesting proposal includes the generation of a tool module to analyze and predict deadlocks generated by potential failures on a workflow.
- Actually, the values for constants such *Modulator factor* (K) and *Multiply factor* (MF) were defined empirically and tuned with base on the observation of the

simulations behavior. A good supplementary component will consist on develop an add-on component capable to suggest possible initial values for the K and MF factors, according to particular parameters and expected behavior of a scheduling problem.

- Last but not least is the improvement of a graphic interface for the framework to facilitate the realization of friendly simulations.

APPENDICES

APPENDIX A
GUIDELINES FOR THE FRAMEWORK SOFTWARE

A.1 Introduction

The Quality of service based scheduling framework was defined over *SimJava*. *SimJava* is a discrete event, process oriented simulation package for *Java*.

The approach to simulating systems adopted by our framework is inherited from *SimJava*. Each system is considered to be a set of interacting entities as they are referred to in SimJava. These entities communicate with each other by passing events. The simulation time progresses on the basis of these events.

A.2 Getting started

A.2.1 A first example

Throughout this guideline a small example will serve to demonstrate in practice the functionality of the framework. The system presented in the example will be a simplified model of a digital printing workflow.

The actual model consists of a *Source* of jobs, a *Global Scheduler*, a *Router*, a stage of *Artifact recognition* process, a second *Router*, a stage of *Riping* process and a *End* Stage, see Figure A–1. The following is a brief description of the functionality of each of the components mentioned above:

***Source*:** It is in charge of generating new Jobs with the characterization required for each specific problem. Such characterization includes the size of the job, the

elapsed time between the arrival of new jobs and the faults contained in each new Job.

***Global Scheduler*:** The Global Scheduler work as a whole central entity executing tasks as resource manager also. Playing as Global Scheduler uses the information gathered while playing as Resource Manager to select the optimal next set of resources (*ResourceSet*) available for any job that requires the execution of a process. Playing as Resource Manager, maintains on line the information of the best *ResourceSet* during any time of the simulation and in any stage of The Framework.

***Router*:** Its function consist of transferring a Job between the *Global Scheduler* and the first *Stage*, between two consecutive *Stages* of the Framework or between the last *Stage* of the Framework and the *End Stage* of the Framework. To perform the redirection of the job toward the correct next *Stage*, the *Router* uses the name of the port which was previously stamped in the job either by the previous stage or by the *Global Scheduler*.

***ResourceSet*:** It works as a whole entity capable of receiving a Job, processing and sending it to the next step through a Router. Furthermore, the *ResourceSet* must inform the *Global Scheduler* about changes in the work load caused by the processing of each Job, as well as to receive information of the best next stage to set it up in the dispatcher of the current *ResourceSet*. A *ResourceSet* is composed of one *Local Scheduler*, one *Dispatcher* and one or more homogeneous *Resources*.

***End Stage*:** It does not perform any process on the Job The *End Stage* receives each Job and makes a report with the information of the process of the Job in the whole Framework.

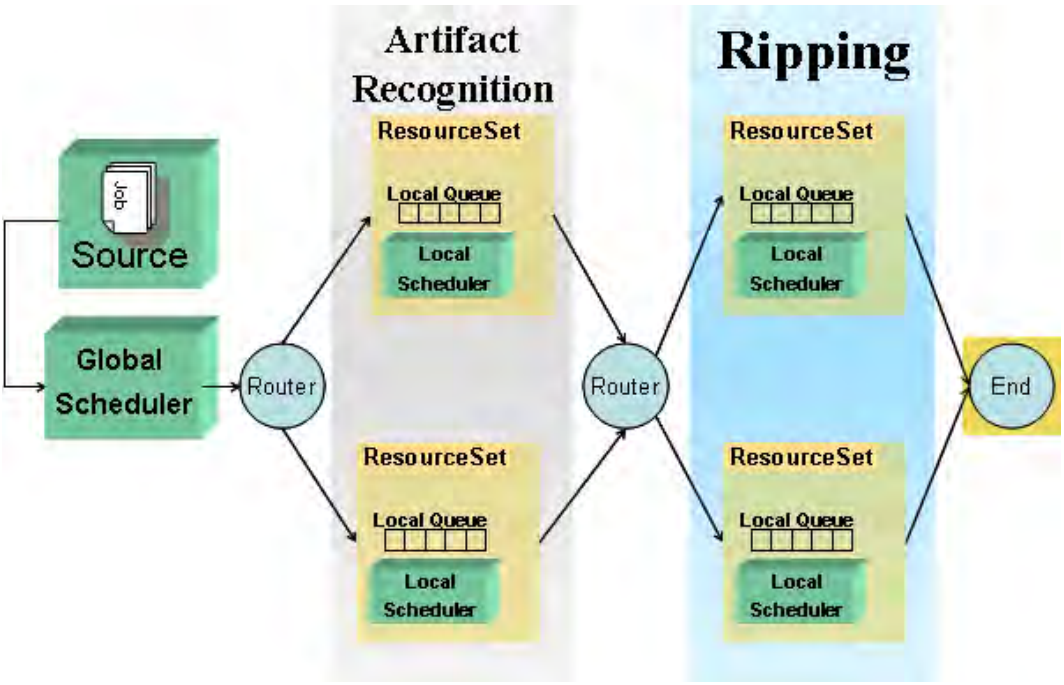

Figure A–1: The simplified digital printing model

A.3 Specifying components' behavior

To define a framework component, it was necessary subclass the *SimJava* class Sim_entity. The subclass implements the component's desired behavior. This behavior is provided by means of the body() method which must be overridden in the subclass.

Now that the framework components for the simulation are already known, if it is desired to make changes in the scheduling algorithm or in the properties of jobs, it is necessary to perform some of the following steps:

- If a change in the scheduling algorithm is desired, the it is necessary to subclass the framework's class LocalScheduler, overwriting the body of the class, the quality of service function and the weighting factors, W1, W2 and W3.

- If a change in the properties in the Jobs generated by the source is desired, it is necessary to overwrite the functions that defines the presence of a fault in a job in the class *Resource*.

A.4 Setting up the simulation

After perform the required changes in the source of the framework, it is possible to proceed to setup the simulation itself. To define the simulation's main() method it is necessary to create one further class. The name given to this class should be representative of the system being simulated. In general, a simple *SimJava* simulation requires four steps:

1. Initialize Sim_system.
2. Make an instance for each entity.
3. Link the entities.
4. Run the simulation.

The following figure presents an example of the code of the main class of the simulation created for the simple digital printing model:

```
public static void main(String[] args) {
int intUniqueResourceIdTemp = 0;
Sim_system.initialise();                    // Initialise Sim_system
//Primero Se crea el source (solo uno)
source = new Source("Source", 10.5, NUM_JOBS);   // Add the source

//se crea el stage final
objFinalStage = new EndStage(strNombreStageFinal);

//Luego se crea el Global Scheduler (solo uno)
objGlobalScheduler = new GlobalScheduler("GlobalScheduler");

//Se enlazan los puertos del source y del Global Scheduler (son fijos)
Sim_system.link_ports("Source", "Out", "GlobalScheduler", "GlobalIn");
```

```
//Se se definen ordenadamente los stages y un router antecesor para
//c/uno de ellos.
//Se define el primer Stage
intStageNumber =1;

//Se agrega una entrada a la lista de stages (Stage list) del
//Global Scheduler
objGlobalScheduler.addHashMapStage();

//Se define el primer router
strNombreRouter = "Router"+intStageNumber;
objRouterStage1 = new RouterQBMUF(strNombreRouter, 0);

//Se conecta el Global scheduler con el router 1
Sim_system.link_ports("GlobalScheduler","GlobalOut",
        strNombreRouter,strNombrePuertoEntradaRouter);

//************************************
//****Set de Recursos 1 - Stage 1*******
//Se define el primer Set de Recursos (ResourceSet) para dicho stage
intResourceNumber = 1;
strNombreResourceSet = "ResourceSet"+intStageNumber+"_"+intResourceNumber;
objResourceSet1_1 = new ResourceSet(strNombreResourceSet,
intResourceNumber, intStageNumber);

//Se crean dos recursos dentro del Resource Set
intUniqueResourceIdTemp = objResourceSet1_1.addNewResource();
intUniqueResourceIdTemp = objResourceSet1_1.addNewResource();

//Se aade un nuevo puerto al router
strNombrePuerto = objRouterStage1.addNewPort();

//se conecta el router a travs del puerto creado con el ResourceSet creado
Sim_system.link_ports(strNombreRouter, strNombrePuerto,
strNombreResourceSet,"ResourceSetIn");

//Se crea un ResourceSetInfo con la informacin del ResourceSet creado
intUniqueIDRSNumber = objResourceSet1_1.get_id(); objRSInfo = new
ResourceSetInfo(intStageNumber,intResourceNumber,
 strNombrePuerto, intUniqueIDRSNumber);

//Se agrega una entrada al hashtable del stage 1 con el ResourceSetinfo
objGlobalScheduler.addResourceSetInfo(objRSInfo);
objRSInfo = null;
```

```
//****************************************
//******Set de Recursos 2 - Stage 1 ********
//Se define el segundo Set de Recursos (ResourceSet) para dicho stage
intResourceNumber = 2;
strNombreResourceSet = "ResourceSet"+intStageNumber+"_"+intResourceNumber;
objResourceSet1_2 = new ResourceSet(strNombreResourceSet,
intResourceNumber, intStageNumber);

//Se crean tres recursos dentro del Resource Set 2
intUniqueResourceIdTemp = objResourceSet1_2.addNewResource();
intUniqueResourceIdTemp = objResourceSet1_2.addNewResource();
intUniqueResourceIdTemp = objResourceSet1_2.addNewResource();

//Se aade un nuevo puerto al router
strNombrePuerto = objRouterStage1.addNewPort();

//se conecta el router a travs del puerto creado con el ResourceSet creado
Sim_system.link_ports(strNombreRouter, strNombrePuerto,
strNombreResourceSet,"ResourceSetIn");

//Se crea un ResourceSetInfo con la informacin del ResourceSet creado
intUniqueIDRSNumber = objResourceSet1_2.get_id();
objRSInfo = new ResourceSetInfo(intStageNumber,
intResourceNumber,strNombrePuerto, intUniqueIDRSNumber);

//Se agrega una entrada al hashtable del stage 1 con el ResourceSetinfo
objGlobalScheduler.addResourceSetInfo(objRSInfo);
objRSInfo = null;

//*******************************Stage2 ****************
//Se define el Segundo Stage
intStageNumber =2;

//Se agrega una entrada a la lista de stages (Stage list) del Global Scheduler
objGlobalScheduler.addHashMapStage();

//Se define el segundo router
strNombreRouter = "Router"+intStageNumber;
objRouterStage2 = new RouterQBMUF(strNombreRouter, 0);

//*********************************************************************
//**Se conecta la salida de los stages anteriores con el nuevo router
//*********************************************************************
objResourceSet1_1.connectToRouter(strNombreRouter,
strNombrePuertoEntradaRouter);
```

```
objResourceSet1_2.connectToRouter(strNombreRouter,
\strNombrePuertoEntradaRouter);

//************************************
//****Set de Recursos 1 - Stage 2*******
//Se define el primer Set de Recursos (ResourceSet) para dicho stage
intResourceNumber = 1;
strNombreResourceSet = "ResourceSet"+intStageNumber+"_"+intResourceNumber;
objResourceSet2_1 = new ResourceSet(strNombreResourceSet,
intResourceNumber, intStageNumber);

//Se crean 3 recursos dentro del Resource Set
intUniqueResourceIdTemp = objResourceSet2_1.addNewResource();
intUniqueResourceIdTemp = objResourceSet2_1.addNewResource();
intUniqueResourceIdTemp = objResourceSet2_1.addNewResource();

//Se aade un nuevo puerto al router
strNombrePuerto = objRouterStage2.addNewPort();

//se conecta el router a travs del puerto creado con el ResourceSet creado
Sim_system.link_ports(strNombreRouter, strNombrePuerto,
strNombreResourceSet,"ResourceSetIn");

//Se asigna este puerto como mejor puerto inicial en los
//resourceSets del stage anterior
//*****esta parte debe ser automatizada al unirlo con la interfaz grfica
objResourceSet1_1.setNextportNameonDisp(strNombrePuerto);
objResourceSet1_2.setNextportNameonDisp(strNombrePuerto);

//Se crea un ResourceSetInfo con la informacin del ResourceSet creado
intUniqueIDRSNumber = objResourceSet2_1.get_id();
objRSInfo = new ResourceSetInfo(intStageNumber,intResourceNumber,
 strNombrePuerto, intUniqueIDRSNumber);

//Se agrega una entrada al hashtable del stage 1 con el ResourceSetinfo
objGlobalScheduler.addResourceSetInfo(objRSInfo);
objRSInfo = null;

//****************************************
//******Set de Recursos 2 - Stage 2 ********
//Se define el primer Set de Recursos (ResourceSet) para dicho stage
intResourceNumber = 2;
strNombreResourceSet = "ResourceSet"+intStageNumber+"_"+intResourceNumber;
objResourceSet2_2 = new ResourceSet(strNombreResourceSet,
intResourceNumber, intStageNumber);
```

```
//Se crean dos recursos dentro del Resource Set
intUniqueResourceIdTemp = objResourceSet2_2.addNewResource();
intUniqueResourceIdTemp = objResourceSet2_2.addNewResource();

//Se aade un nuevo puerto al router
strNombrePuerto = objRouterStage2.addNewPort();

//se conecta el router a travs del puerto creado con el ResourceSet creado
Sim_system.link_ports(strNombreRouter, strNombrePuerto,
strNombreResourceSet,"ResourceSetIn");

//Se crea un ResourceSetInfo con la informacin del ResourceSet creado
intUniqueIDRSNumber = objResourceSet2_2.get_id();
objRSInfo = new ResourceSetInfo(intStageNumber,intResourceNumber,
strNombrePuerto, intUniqueIDRSNumber);

//Se agrega una entrada al hashtable del stage 1 con el ResourceSetinfo
objGlobalScheduler.addResourceSetInfo(objRSInfo);
objRSInfo = null;

//**********************************
//*******Se supone que es el stage final asi que se manda a conectar cada
//*******set de recursos con el stage final
//**********************************
objResourceSet2_1.connectToRouter(strNombreStageFinal,
strNombrePuertoEntradaStageFinal);
objResourceSet2_2.connectToRouter(strNombreStageFinal,
strNombrePuertoEntradaStageFinal);

//Genero la lista de puertos iniciales en el Global Scheduler
objGlobalScheduler.setBestNextStages();

Sim_system.set_trace_detail(true, false, false);
Sim_system.set_termination_condition(Sim_system.EVENTS_COMPLETED,
        strNombreStageFinal, QB_MUFTags.IND_REDIRECT, NUM_JOBS, false);

Sim_system.run();
}
```

REFERENCE LIST

[1] M. Pinedo. *Scheduling - Theory, Algorithms and Systems*. Prentice Hall, 2002.

[2] U. Al-Turki, C. Fedjki, and A. Andijani. Tabu search for a class of single-machine scheduling problems. *Computers & Operations Research*, pages 1223–1230, 2001.

[3] A. Agnetis, A. Alfieri, and G. Nicosia. A heuristic approach to batching and scheduling a single machine to minimize setup costs. *Computers & Industrial Engineering*, pages 793–802, 2004.

[4] H. Tamaki, T. Komori, and S. Abe. A heuristic approach to parallel machine scheduling with earliness and tardiness penalties. *7th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1367–1370, 1999.

[5] Z.L. Chen and W.L. Powell. Solving parallel machine scheduling problems by column generation. *INFORMS Journal on computing*, pages 78–94, 1999.

[6] D.B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal of Computing*, pages 617–632, 1994.

[7] L. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk. Better approximation guarantees for job shop scheduling. *8th ACM-SIAM Symposium on Discrete Algorithms(SODA)*, pages 599–608, 1997.

[8] S. Sevastianov and G. Woeginger. Makespan minimization in open shops : A polynomial time approximation scheme. *Mathematical Programming*, pages 82(191–198), 1998.

[9] M. Sviridenko, K. Jansen, and R. Solis-Oba. Makespan minimization in job shops: A polynomial time approximation scheme. *31st Annual ACM Symposium on Theory of Computing*, pages 394–399, 1999.

[10] S. Yang and D. Wang. Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *IEEE Transactions on Neural Networks*, pages 474–486, 2000.

[11] S.E. Elmaghraby. On the optimal release time of jobs with random processing times, with extensions to other criteria. *International Journal of Production Economics*, pages 103–113, 2001.

[12] D. Golenko-Ginzburg and A. Gonik. Optimal job-shop scheduling with random operations and cost objectives. *International Journal of Production Economics*, pages 147–157, 2002.

[13] S. Nakasuka and T. Yoshida. New framework for dynamic scheduling of production systems. *International Workshop on Industrial Applications of Machine Intelligence and Vision*, pages 253–258, 1989.

[14] D.L. Levine, C.D. Gill, and D.C. Schmidt. Dynamic scheduling strategies for avionics mission computing. *Proceedings The 17th AIAA/IEEE/SAE Digital Avionics Systems Conference*, 1:C15/1–C15/8, 1998.

[15] D.B. Stewart and P.K. Khosla. Real-time scheduling of sensor-based control systems. *Proceedings 8th IEEE Workshop on Real-Time Operating Systems*, pages 144–150, 1991.

[16] A. Sulistio, G. Poduvaly, R. Buyya, and CH. Tham. Constructing a grid simulation with differentiated network service using gridsim. *Proceedings of The 6th International Conference on Internet Computing (ICOMP'05)*, 2005.

[17] F. Howell and R. McNab. simjava: a discrete event simulation package for java with applications in computer systems modelling. *Proceedings of the First International Conference on Web-based Modelling and Simulation*, Jan 1998.

[18] E.M. Goldratt. *The Goal*. The North River Press, MA 1984.

[19] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.

[20] F. Berman, G. Fox, and T. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[21] F. Berman and R. Wolski. Scheduling from perspective of the application. *Proceedings of the Symposium on High Performance Distributed Computing*, 1996.

[22] J.M. Schopf. Ten actions when superscheduling. Technical Report WD8.5, Scheduling Working Group, 2001.

[23] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerical libraries and the grid. *Proceedings of Supercomputing 01*, 2001.

[24] M. Baker, R. Buyya, and D. Laforenza. Grids and grid technologies for wide-area distributed computing. *Intl. Journal of Software: Practice and Experience*, 2002.

[25] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using apples. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 14(4):369–382, 2003.

[26] R. Buyya, D. Abramson, and J. Giddy. A computational economy for grid and its implementation in the nimrod/g resource broker. *Future Generation Computer Systems, Elsevier Science*, 2002.

[27] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The apples parameter sweep template: Userlevel middleware for the grid. *Proceedings of Supercomputing 00*, 2000.

[28] G. Shao, R. Wolski, and F. Berman. Master/slave computing on the grid. *Proceedings of Heterogeneous computing Workshop*, 2000.

[29] J.B. Weissman. Prophet: Automated scheduling of spmd programs in workstation networks. *Comcurrency: Practice and Experience*, 11(6), 1999.

[30] J.B. Weissman. Gallop: The benefits of wide-area computing for parallel processing. *Journal of Parallel and Distributed Computing*, 54(2):183–205, 1998.

[31] M. Sirbu and D. Marinescu. A scheduling expert advisor for heterogeneous environments. *In Proc. Heterogeneous Computing Workshop*, pages pp. 74–87, 1997.

[32] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valiente, I. Ra, D. Kim, Y. Kim, X. Bing, and B. Ye. The software architecture of a virtual distributed computing environment. *Proceedings of the High-Performance Distributed Computing Conf.*, pages pp.40–49, 1997.

[33] J. Gehring and A. Reinefeld. Mars: A framework for minimizing the job execution time in a metacomputing environment. *Future Generation Computer Systems*, 12(1):87–99, 1996.

[34] I. Foster, Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the i-way metacomputing experiment. *Concurrency: Practice and Experience*, 1998.

[35] J. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey, and P. Stephan. Dome: Parallel programming in a heterogeneous multi-user environment. Technical Report CMU-CS-95-137, Carnegie Mellon University, Pittsburg, 1995.

[36] P. Au, J. darlington, M. Ghanem, Y. Guo, H. To, and J. Yang. Coordinating heterogeneous parallel computation. *Proceedings of the 1996 Euro-Par Conf.*, pages pp. 601–614, 1996.

[37] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the maui scheduler. *Lecture Notes in Computer Science*, 2221:87–102, 2001.

[38] P. Keyani, N. Sample, and G. Wiederhold. Scheduling under uncertainty: Planning for the ubiquitous grid. Technical report, Stanford Database Group, 2002.

[39] F. Berman. The grads project: Software support for high-level grid application development. *International Journal of high Performance Computing Applications*, 15(4):327–344, 2001.

[40] A. Kohkhar, V. Prasanna, M. Shaaban, and C. Wang. Heterogeneous computing: Challenges and opportunities. *IEEE Computer*, 26(6), 1993.

[41] KB. Hamidzadeh, D. Lilja, and Y. Arif. Dynamic scheduling techniques for heterogeneous computing systems. *Concurrency: Practice and Experience*, 7(7):633–652, 1995.

[42] H. Siegel, J. Antonio, R. Metzger, M. Tan, and Y.A. Li. *Heterogeneous computing. Parallel and Distributed Computing Handbook.* New York: McGraw-Hill, 1996.

[43] D.B. Stewart, D.E. Schmitz, and P.K. Khosla. Implementing real-time robotic systems using CHIMERA II. *Proceedings of The IEEE International Conference on Robotics and Automation*, pages 598–603, 1990.

[44] D.B. Stewart and P.K. Khosla. Real-time scheduling of dynamically reconfigurable systems. *IEEE International Conference on Systems Engineering*, pages 139–142, 1991.

[45] V. Kalogeraki, P.M. Melliar-Smith, and L.E. Moser. Dynamic scheduling of distributed method invocations. *Proceedings of The 21st IEEE Real-Time Systems Symposium*, pages 57–66, 2000.

[46] L. He, S.A. Jarvis, D.P. Spooner, and G.R. Nudd. Dynamic scheduling of parallel real-time jobs by modeling spare capabilities in heterogeneous clusters. *Proceedings of The IEEE International Conference on Cluster Computing*, 2003.

[47] B. Zolfaghari. A dynamic scheduling algorithm with minimum context switches for spacecraft avionics systems. *Proceedings of IEEE Aerospace Conference*, pages 2618 – 2624, 2004.

[48] C. Hartmann and R. Vilzmann. Urgency based scheduling for user-individual qos in cellular mimo-systems. *ITG Workshop on Smart Antennas*, pages 257–260, 2004.

[49] Yiping Yuan., Tao Yu., Feng Xiong., and Minlun Fang. Qos-based dynamic scheduling for manufacturing grid workflow. *Ninth International Conference on Computer Supported Cooperative Work in Design*, pages 1123– 1128, 2005.

[50] A. Mittal, G. Manimaran, and C.S.R. Murthy. Integrated dynamic scheduling of hard and qos degradable real-time tasks in multiprocessor systems. *Fifth International Conference on Real-Time Computing Systems and Applications*, pages 127–136, 1998.

[51] H. J. Santos-Villalobos. Style-dependent artifact recognition for digital variable data printing. Master's thesis, University of Puerto Rico, Mayaguez Campus, July 2005.

[52] G. A. Chaparro-Baquero. Statistical correlation between job creators and print failures. Technical report, HP Labs, 2004.

[53] L. Bautista and D. Rodríguez. Web-based data processing for hydro-ecological applications. *Fourth LACCEI International Latin American and Caribbean, Conference for Engineering and Technology*, June 2006.

BIOGRAPHICAL SKETCH

WILSON E. LOZANO-ROLÓN

Was born on February 18th, 1977, in Cúcuta, Norte de Santander, Colombia. Wilson is the son of David Lozano and Marydilia Rolón. In November 2000, he received his B.S. degree in computer engineering from the Universidad Industrial de Santander, Bucaramanga Campus. Prior to beginning his Masters program in computer engineering, he worked in various areas related to information technology. He acted as web developer, support engineer for technical applications, web system administrator, and part time professor.

In 2004 he left Colombia and starts the M.S. studies in computer engineering at the University of Puerto Rico, Mayagüez campus. He worked as research assistant in the PDC laboratory under the supervision of Dr. Wilson Rivera. Wilson Lozano did his research in dynamic scheduling applied to digital publishing and grid computing.

A FRAMEWORK FOR DYNAMIC SCHEDULING BASED ON QUALITY OF SERVICE METRICS

Wilson Ernesto Lozano Rolón
(787) 464-1864
Department of Electrical and Computer Engineering
Chair: Wilson Rivera
Degree: Master of Science
Graduation Date: July 10 2006

This is the general Audience Abstract.

Use the file: `GeneralAudienceAbstract.tex`