ISCTE IUL
Instituto Universitário de Lisboa

PhD Program in Information Science and Technology

Department of Information Science and Technology

Task Scheduling for Application Integration
A Strategy for Large Volumes of Data

Daniela Lopes Freire

Thesis specially presented for the fulfillment of the degree of
Doctor in Information Science and Technology

Supervisors:

Dr. Vitor Manuel Basto Fernandes
University Institute of Lisbon

Dr. Rafael Zancan Frantz
Regional University of the Northwest of the State of Rio Grande do Sul

Co-supervisor:

Dr. Fabricia Carneiro Roos Frantz
Regional University of the Northwest of the State of Rio Grande do Sul

February, 2020

First published in February 2020 by
Applied Computing Research Group - GCA
Department of Exact Sciences and Engineering
Rua Lulu Ilgenfritz, 480 - São Geraldo
Ijuí, 98700-000, Brazil.

Support: Scholarship granted by the Brazilian Coordination Board for the Improvement of University Personnel (CAPES). The research developed in this thesis was also supported by the Research Support Foundation of the State of Rio Grande do Sul (FAPERGS) grant 17/2551-0001206-2.

The committee in charge of evaluating the thesis entitled “Task Scheduling for Application Integration: a Strategy for Large Volumes of Data”, presented by Daniela Lopes Freire in partial fulfilment of the requirements for the degree of Doctor of Mathematical and Computational Modelling (by Unijuí University - Brazil) & Information Sciences and Technologies (by University Institute of Lisbon - Portugal), hereby recommends ____________________ this thesis and awards the author the grade of Doctor of Mathematical and Computational Modelling (by Unijuí University - Brazil) & Information Sciences and Technologies (by University Institute of Lisbon - Portugal).

I dedicate this work in memory of José Freire Bento (1943–2018).

List of Figures

List of Tables

List of Algorithms

Index of Abbreviations

ACO - Ant Colony Optimisation

AHP - Analytic Hierarchy Process

API - Application Programming Interface

DAG - Directed Acyclic Graph

EAI - Enterprise Integration Application

EIP - Enterprise Integration Pattern

FCFS - First-Come-First-Serve

FIFO - First-In-First-Out

GA - Genetic Algorithm

GCA - Applied Computing Research Group

HRRN - Highest Response Ratio Next

iPaaS - Integration Platform-as-a-Service

IPS - Integration Process Simulator

IPTG - Integration Pattern Typed Graph

IOTG - Integration Operation Typed Graph

I/O - In/Out

IT - Information Technology

JVM - Java Virtual Machine

LJF - Longest Job First

MqRR - Multi-queue Round Robin

MVLR - Multi-vocal Literature Review

PSO - Particle Swarm Optimisation

P&F - Pipe-and-Filters

qPrior - Queue-Priority

RR - Round Robin

SA - Simulated Annealing

SJF - Shortest Job First

SJN - Shortest Job Next

SPN - Shortest Process Next

SRTF - Shortest Remaining Time First

SSE - Sum of Squared Error

SST - Sum of Squared Total

VM - Virtual Machine

Index of Symbols

ω - inertia weight of the PSO

ϕ_p - acceleration coefficient of the best position of the particle i of the PSO

ϕ_g - acceleration coefficient of the general best position of the PSO

r - random number $\in [0, 1]$ used in the PSO

$\overrightarrow{bpos_i}(t)$ - best position of the particle i of the PSO

$\overrightarrow{gbpos}(t)$ - general best position of the population of the PSO

$\overrightarrow{pos_i}(t)$ - current position of the particle i of the PSO

$\overrightarrow{pos_i}(t+1)$ - position of the particle i in next iteration of PSO algorithm

$\overrightarrow{vel_i}(t)$ - current velocity of the particle i of the PSO

$\overrightarrow{vel_i}(t+1)$ - velocity of the particle i in next iteration of PSO algorithm

R^2 - correlation coefficient

K - number of treatments in Scott & Knott test

T - number of treatments in Scott & Knott test

B - sum of squares in Scott & Knott test

B_o - maximum value of the sum of squares between groups in Scott & Knott test

rp - number of repetitions of an experiment

$\hat{\sigma}_0^2$ - maximum likelihood estimator of $\frac{\hat{\sigma}_0^2}{r}$

λ - asymptotically a χ^2 distributed random variable

ν_0 - degrees of freedom of an experiment

$x = (x_1, x_2, ..., x_n)$ - vector of the independent variables of a linear regression

y - a dependent variable of a linear regression

β vector of unknown parameters of a linear regression

ε - disturbance term of a linear regression

R^2 - correlation coefficient in statistic tests

x_i - coefficient of the hypothesis test of a linear regression

$t-statistic$ - test used in linear regression

$p-Value$ - probability value

$F-statistic$ - test used in linear regression

$F-Value$ - probability value

W - workflow

E - set of edges of a direct acyclic graph

$J = \{j_1, j_2, ..., j_n\}$ - set of jobs that represent the executions of an integration process

$T = \{t_1, t_2, ..., t_n\}$ - set of tasks of an integration process

$R = \{r_1, r_2, ..., r_m\}$ - set of computational resources

$S = \{s_1, s_2, ..., s_n\}$ - set of task segments

$\{t_{ini}, t_{fin}\}$ - time interval

F - set of the function type for tasks of integration process

$\bullet\, t = \{t' \in T | (t' \bullet t) \in E\}$ - set of direct predecessors of a task t

$t \bullet = \{t'' \in T | (t \bullet t'') \in E\}$ - set of direct successors of a task t

ET_{t_i} - Execution time of a task t_i

$TT_{e_{ij}}$ - Transference time of a message between a task t_i and its successor t_j

VM_j - Virtual Machine of type j

T_R - Resource preparation time

T_Q - Queuing time

T_D - Data transfer time

T_O - Overhead system time

TP_{t_i} - Total processing time of a task t_i

$TQ_{t_{ij}}$ Task waiting time of a task t_i in a task queue

ST_{m_1} Start time of the first message that entered in an integration process

ET_{m_np} - End time of the last message that leaves an integration process

TET - Total execution time

$\{C_i'\}$ - Computation and communication duration

$TT(t_i)$ - Transfer time

$SC(t_i)$ - Security services overhead

NaN - Undefined or unrepresentable value

$\overline{pm}$ - average of processed messages

w - workload

x_1 - number of tasks of type «$start$»

x_2 - number of tasks of type «and»

x_3 - number of tasks of type «or»

x_4 - number of tasks of type «xor»

x_5 - number of tasks of type «$join$»

x_6 - number of tasks of type «$message\ processor$»

x_7 - number of tasks of type «$external\ call$»

x_8 - number of tasks of type «end»

x_9 - number of communication channels

x_{10} - number of sequential segments

x_{11} - number of parallel segments

Wilkinson Notation - *According to Table 1*

Wilkinson Notation	**Factors in Standard Notation**
1	Constant (intercept) term
A^k, where k is a positive integer	A, A^2, ..., A^k
A + B	A, B
A*B	A, B, A*B
A:B	A*B only
-B	Do not include B
A*B + C	A, B, C, A*B
A + B + C + A:B	A, B, C, A*B
A*B*C - A:B:C	A, B, C, A*B, A*C, B*C
A*(B + C)	A, B, C, A*B, A*C

Table 1: *Wilkinson Notation.*

Acknowledgements

The way to achieve your own success is to be willing to help somebody else get it first.

Iyanla Vanzant, American lawyer

My family and my friends were my partners in this journey, motivating me and helping me day after day. It is not possible to express in words the sincere gratitude I have for the friendship of the professor Rafael Z. Frantz and professor Fabricia Carneiro Roos Frantz; I became a researcher, thanks to their dedication and patience. I thank professor Vitor Manuel Basto Fernandes, who first bet on me even without knowing me. I thank him for his commitment to co-mentoring by University Institute of Lisbon and all the support given during the learning period in Lisbon, Portugal. I am extremely grateful to be part of the Applied Computing Research Group, a group composed of people that love what they do and make any environment pleasant. I have a special acknowledgement to my friend Eldair Fabrício, which collaborated with the build of the simulation tool.

I thank the collaborations of the researchers Dr. Sandro Sawicki (Unijuí University, Brazil), Dr. Maria do Rosário Laureano and Dr. Sancho M. Oliveira (University Institute of Lisbon, Portugal), and Dr. Inmaculada Hernández Salmerón and Dr. Rafael Corchuelo (University of Seville, Spain). The researchers contributed to the planning of parts of the research, definition of research problems, participation in follow-up meetings and seminars, articles review, feedback's about our inferences.

I am also profoundly thankful for the PhD scholarship granted by the Coordination for the Improvement of Higher Education Personnel (CAPES). As well as of the support received from the Research Support Foundation of the State of Rio Grande do Sul (FAPERGS), without which this project would not have been possible. Last, but not least, I would like to thank UNIJUI and ISCTE professors and employees, especially Mrs. Geni for her care and attention.

Contents

Abstract

Nothing in life is to be feared, it is only to be understood.
Now is the time to understand more, so that we may fear less.

Marie Curie, Polish physicist (1867-1934)

Enterprise Application Integration is the research field, which provides methodologies, techniques and tools for modelling and implementing integration processes. An integration process performs the orchestration of a set of applications to keep them synchronised or to allow the creation of new features. It can be represented by a workflow composed of tasks and communication channels. Integration platforms are tools for the design and execution of integration processes in which, the runtime system is the component responsible for the execution time of the tasks and the allocation of computational resources that perform them. The processing of a large volume of data, corresponding to execution of millions of tasks, can cause situations of overload, characterised by the accumulation of tasks in internal queues awaiting computational resources in the runtime systems, resulting in unacceptable response time for the external applications and users. Our research hypothesis is that the runtime systems of the integration platforms use simplistic heuristics for scheduling tasks, which does not allow them to maintain acceptable levels of performance when there are overload situations. In this research work, we developed *(i)* a representation for integration processes, *(ii)* a characterisation for your task schedules, *(iii)* a heuristic to deal with situations of overload, *(iv)* a mathematical model for a performance metric of the execution of integration processes and *(v)* a simulation tool for task scheduling heuristics. Our research results indicate that, in situations of overload, our heuristic promotes a balanced workload distribution and an increase in the performance of the execution of the integration processes.

Resumo

Nada na vida deve ser temido, é apenas para ser entendido. Agora é a hora de entender mais, para que possamos ter menos medo.

Marie Curie, Física polonêsa (1867-1934)

Integração de Aplicações Empresariais é o campo de pesquisa, que fornece metodologias, técnicas e ferramentas para modelar e implementar processos de integração. Um processo de integração executa a orquestração de um conjunto de aplicações para mantê-las sincronizadas ou para permitir a criação de novas funcionalidades. Ele pode ser representado por um fluxo de trabalho composto por tarefas e canais de comunicação. Plataformas de integração são ferramentas para projetar e executar processos de integração, nas quais o motor de execução é o componente responsável pelo tempo de execução das tarefas e pela alocação de recursos computacionais que as executam. O processamento de um grande volume de dados, correspondendo a execução de milhões de tarefas, pode causar situações de sobrecarga, caracterizadas pelo acúmulo de tarefas em filas internas que aguardam recursos computacionais nos motores de execução, resultando em tempos de resposta inaceitáveis para aplicações e usuários externos. Nossa hipótese de pesquisa é que os motores de execução das plataformas de integração usam heurísticas simplistas para agendar tarefas, o que não lhes permitem manter níveis aceitáveis de desempenho em situações de sobrecarga. Neste trabalho de pesquisa, desenvolvemos *(i)* uma representação para processos de integração, *(ii)* uma caracterização para seus agendamentos de tarefas, *(iii)* uma heurística para lidar com situações de sobrecarga, *(iv)* um modelo matemático para uma métrica de desempenho da execução de processos de integração e *(v)* uma ferramenta de simulação para heurísticas de agendamento de tarefas. Nossos resultados de pesquisa indicam que, em situações de sobrecarga, nossa heurística promove uma distribuição equilibrada da carga de trabalho e um aumento no desempenho da execução dos processos de integração.

Resumen

No hay que temer nada en la vida, solo hay que entenderlo.
Ahora es el momento de entender más, para que podamos temer menos.

Marie Curie, Físico polaco (1867–1934)

Integración de aplicaciones empresariales es el campo de investigación, que proporciona metodologías, técnicas y herramientas para modelar e implementar procesos de integración. Un proceso de integración realiza la orquestación de un conjunto de aplicaciones para mantenerlas sincronizadas o para permitir la creación de nuevas funcionalidades. Se puede representar mediante un flujo de trabajo compuesto por tareas y canales de comunicación. Las plataformas de integración son herramientas para diseñar y ejecutar procesos de integración, en los cuales el motor de ejecución es el componente responsable del tiempo de ejecución de las tareas y de la asignación de recursos computacionales que las ejecutan. El procesamiento de un gran volumen de datos, correspondiente a la ejecución de millones de tareas, puede causar situaciones de sobrecarga, caracterizadas por la acumulación de tareas en colas internas que esperan recursos computacionales en los motores de ejecución, el que resulta en tiempos de respuesta inaceptables para las aplicaciones y los usuarios externo. Nuestra hipótesis de investigación es que los motores de ejecución de las plataformas de integración hacen uso heurística simplista para programar tareas, lo que no les permite mantener niveles aceptables de rendimiento en situaciones de sobrecarga. En este trabajo de investigación, desarrollamos *(i)* una representación para los procesos de integración, *(ii)* una caracterización para sus cronogramas de tareas, *(iii)* una heurística para lidiar con situaciones de sobrecarga, *(iv)* un modelo matemático para una métrica de rendimiento de la ejecución de procesos de integración y *(v)* una herramienta de simulación para la heurística de programación de tareas. Los resultados de nuestra investigación indican que, en situaciones de sobrecarga, nuestra heurística promueve una distribución equilibrada de la carga de trabajo y un aumento en el rendimiento de la ejecución de los procesos de integración.

Part I

Introduction

Chapter 1

Opening Remarks

We cannot change what we are not aware of,
and once we are aware, we cannot help but change.

Sheryl Sandberg, American technology executive

Our goal in this thesis is to report on our research to develop a task scheduling heuristic that improves the performance of the execution of integration processes that are likely to be afflicted by overloads. Task scheduling is classified as a computationally complex problem, which cannot be solved by exact optimisation techniques in reasonable computational time. In this chapter, we introduce the context of our research, motivation, methodology, collaborations, and a summary of our main contributions. Finally, we describe the structure of this thesis.

1.1 Research context

Companies often need to use their software ecosystems [123] to support and improve their business processes. These ecosystems are composed of several applications, usually designed without taking into account their possible integration. Within the area of software engineering, the field of studies known as Enterprise Integration Applications (EAI) [61] seek to provide methodologies, techniques and tools for the design and implementation of integration processes. In general terms, an integration process aims to orchestrate a set of applications to keep them synchronised or provide new features that can be built from those already developed. An integration process is composed of tasks and communication channels, which connect applications of software ecosystems, allowing that these applications work together, exchanging data and functionalities.

Integration platforms are specialised software tools that provide support to design, implement, run, and monitor integration processes. In the last years, several integration platforms have been created by the EAI community. These platforms have been heavily influenced by the catalogue of conceptual integration patterns documented by Hohpe and Woolf [85] and follow the architectural style of Pipes-and-Filters [2]. In an integration process, pipes represent message channels, and filters represent atomic tasks that implement a concrete integration pattern to process encapsulated data in messages. The adoption of this architecture allows uncoupling the tasks that make up the integration process.

There are several open source platforms that can be used to build integration processes such as Mule [49], Camel [91], Spring Integration [58], Synapse [94, 152], Fuse [160], ServiceMix [105], Petals [162], Jitterbit [161], WSO2 ESB [92], and Guaraná [60]. Usually, these integration platforms provide a domain-specific language, development toolkit, a test environment, monitoring tools, and runtime system. The domain-specific language focuses on the elaboration of conceptual models for the integration process, with a level of abstraction close to the domain of the problem. The development toolkit is a set of software tools that supports the implementation of the solution, that is, the transformation of the conceptual model into executable code. The testing environment allows testing individual parts or the entire integration process to identify and eliminate possible defects in the implementation. The monitoring tools are used to monitor, at runtime, the operation of the integration process and detect errors that may occur during message processing. The runtime system provides all the support required to run these integration processes.

Cloud computing [129] is another field of studies that have drawn the attention of the scientific community and represents a new paradigm of development, commercialisation and use of software. This field has been transforming the current software ecosystems and revolutionising the way companies provide computer support to their business processes. Cloud computing enables companies to contract service packages by dramatically reducing their total cost of ownership with the information technology infrastructure, without sacrificing the quality of the IT support provided to their business processes. This reducing is due mainly to the pay-as-you-go charging model that allows billing based on the amount of computing resources consumed by users of the cloud [29]. Along with the pay-as-you-go model, Cloud computing has also brought the elasticity feature, which allows for increasing and decreasing computational resources to meet better the demands of applications running on the cloud infrastructure [40]. The advancement of Cloud computing technologies has led companies to a significant transformation in their software ecosystem, which now includes on-premise applications, migrated applications for virtual machines in the cloud, social media applications and many other software services available in the cloud.

1.2 Motivation

The quality of service that integration processes can achieve in terms of message processing is directly dependent on the runtime system of the integration platform. Figure 1.1 presents a conceptual map in which we abstract the key concepts involved in the research context of this thesis. Typically, to achieve the desired quality of service with an integration process, software engineers have increase computational resources in the server machine on which the integration platform is installed within the enterprise. This approach links the increased performance of an integration process to the increase in financial costs required. Such costs refer to the expansion of the current hardware or the purchase of a new server with higher processing power, to generate the desired performance of the runtime system, increasing the number of messages processed by the integration processes.

Hiring of virtual machines in the cloud to host integration platforms allows a reduction of the total cost of ownership for the realisation of EAI by the companies, as well as by means of the feature of elasticity of the cloud, a greater flexibility for the increment of computational resources [25, 159, 166]. This fact has motivated several integration platform providers to migrate and offer a cloud version of their platforms to run integration processes in the cloud [52]. The migration of integration platforms to virtual machines in the cloud has given rise to a new service model that is being referred to by the EAI community as integration Platform-as-a-Service (iPaaS) [145].

Data from 2015 show that, together, the aggregation of South America, Central America and North America account for 67% of the market for iPaaS integration platforms, followed by Europe, the Middle East and Africa, which together account for 22%, and Asia and the Pacific with 11% of this market, and these values should remain, with little oscillation, by the end of 2019 [175]. The traditional market for integration platforms used on-premise registered a growth of less than 10% in 2016, while the market for iPaaS integration platforms had a 60% expansion over the previous year, representing a global market of 700 million Dollars [77].

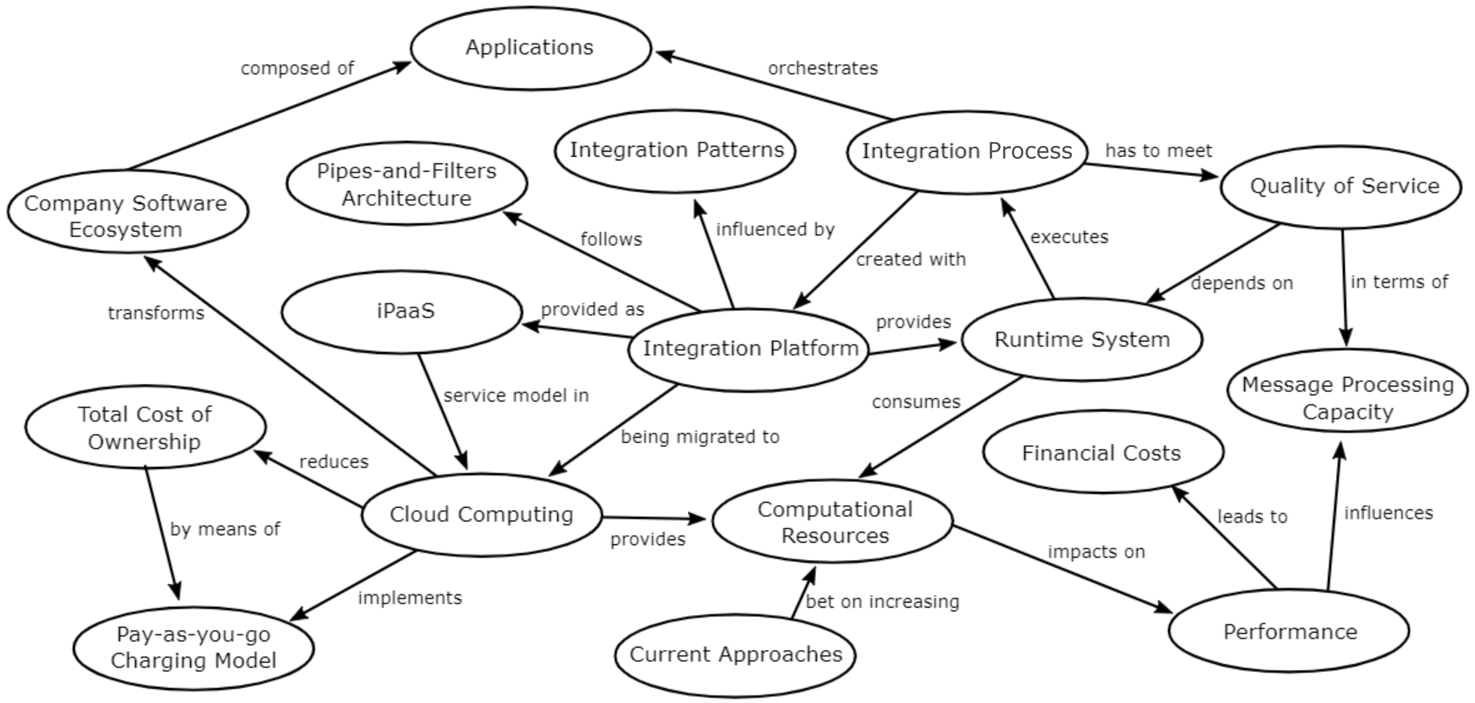


Figure 1.1: *Conceptual map of the key concepts in the research context.*

In 2017, two out of three application integration projects were developed directly with cloud integration platforms [146]. The investment made by companies in iPaaS integration platforms will increase by 40% by 2019 [175], making iPaaS the preferred integration platform by companies and with annual revenue growth higher than the traditional platform market of integration used on-premise [77, 176]. Given the high investments in iPaaS integration platforms, a research effort is needed to study and adapt these platforms to the new paradigm that represents Cloud computing. In this context, the efficiency of runtime systems is fundamental since several computing resources in the cloud follows the pay-as-you-go model, and therefore has a direct impact on the financial cost involved in executing the solutions. The higher the efficiency of runtime systems, the less computational resources will need to be contracted or consumed in order for an integration process to achieve the expected quality of service. The central role of the runtime systems is the task scheduling of the integration processes [76, 83].

1.3 Objectives

We present the objectives that have directed our research in the context of Enterprise Application Integration and that resulted in the contributions of this thesis.

Main

> *Improve the performance of the execution of integration processes in overload situations to endow the runtime systems of integration platforms with features that deal with large volume of data.*

Specifics

- Evaluate current runtime systems with focus in their performance.
- Characterise the task scheduling of integration processes.
- Develop a representation for integration processes that allows the simulation of their execution.
- Build a tool for simulation of the execution of integration processes by runtime systems of integration platforms.
- Implement a new heuristic for task scheduling for integration processes in a programming language.
- Develop a mathematical model for a performance metric of the performance of executions of integration processes.
- Perform statistical tests to evaluate the performance of executions of integration processes with the proposed heuristic.

1.4 Methodology

We classify our research, indicated methods and development process, and list the main parts of research.

Classification

Scientific research varies by genre [45]. We classified our research regarding its nature, goals, technical procedures, and approach the problem, cf. Figure 1.2

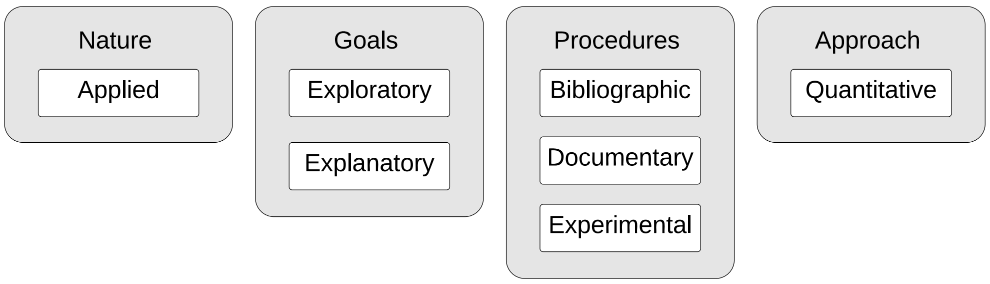


Figure 1.2: *Classification of the research.*

Regarding nature, our research is applied because it produced theoretical and practical knowledge to the advancement of the runtime systems of integration platforms.

Regarding goals, our research is exploratory and explanatory. Exploratory because provided a new representation for integration processes and a classification for the task scheduling. Explanatory because we observed, analysed and interpreted the scheduling of tasks with different heuristics. Additionally, we concerned to investigate the threats to the validity of the observations.

Regarding the technical procedures, our research is bibliographic, documentary, and experimental with case studies. We conducted this part by a multi-vocal literature review [136] that included several sources, such as source-codes of runtime systems, documentation from integration platform, books, and scientific articles. We utilised scientific databases such as IEEE Xplore, ACM Digital Library, and Scopus. Besides, we carried out experiments with four case studies to validate the research proposal, whose results were confirmed by statistical significance tests. The experiments followed a protocol based on Jedlitschka and Pfahl [95], Wohlin et al. [198] and Basili et al. [17].

Regarding of how to approach the problem, our research is quantitative because we formulate the hypotheses and define the relationship between the variables, confirm the results by statistical tests and analyse the threats to their validity.

Methods

The scientific method is based on a set of procedures adopted for the purpose of attaining knowledge [45]. The method of addressing the problem of scientific research characterises the scientific aspect of research [150]. Following, we described the approach and procedures methods of our research, cf. Figure 1.3

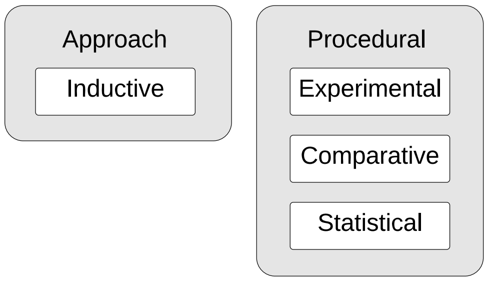


Figure 1.3: *Methods of the research.*

Approach methods offer directives that distinguish scientific and non-scientific objectives. We use the inductive method, which we start from something particular to generalisation. In this method, we begin by observing the scheduling of integration process tasks. Next, we compare the performance of integration process executions with different heuristics for scheduling their tasks, submitted to different scenarios. Finally, we proceeded to the generalisation of the results, validating the conclusions by statistical tests.

Procedural methods refer to the technical procedures to be followed in the research. The procedures to be used in the data collection and analysis depend on the choice of this type of method. We use an experimental, comparative and statistical method. Experimental because we subject our case studies on various scenarios and study the impact of using different heuristics for task scheduling. Comparative because we compare the performance of executions of case studies in each scenario studied. Statistical because we use statistical tests to determine the probability of success and the margin of error of the conclusions obtained.

Development process

The development of this research was iterative and incremental, as shown in Figure 1.4. It was iterative because we planed the work of one iteration to be improved upon in subsequent iterations and it was incremental because, after each iteration, a part of work was completed and was added to our research. Each iteration began with a planning meeting, in which supervisors, together with the doctoral proponent prioritised the items, extracted of our project, that would be carried out, as well as, the activities, and products, which should be performed during that iteration. At the end of each iteration, the proponent presented the progress of the work, by meetings or seminars, attended in-person and by video conference by supervisors, and by members of our research group. The objective was to promote discussions, disseminate knowledge, identify possible inconsistencies and collect new ideas about the work carried out in the iteration.

As the research progresses and incremental content was generated, the proponent participated in workshops and conferences to discuss ideas and have feedback. When these research increments were complete and mature, they were published in journals for dissemination of the results. Besides,

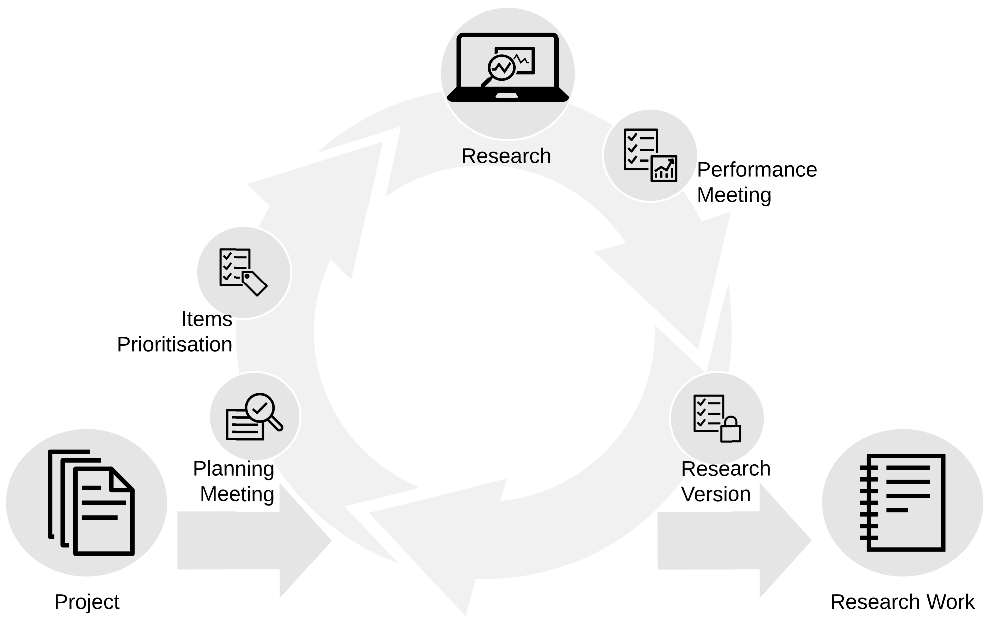


Figure 1.4: *Research development process.*

at the end of the iteration, the proponent and her supervisors held an evaluation meeting to assess the timing, productivity, learning and planning of the next iteration. Then a new one begins, repeating the process until the whole schedule was completed. One of the artefacts generated at the end of the iteration was the research report, which had periodicity defined by the supervisor. It is also important to highlight that the GCA research group is formed by professors of several institutions, whose researches belong to different fields of study within Software Engineering, with emphasis on the Enterprise Application Integration, under which this research is anchored.

Parts of research

The research was divided into seven major parts:

i. Research context review

   In this part, we review topics regarding the research context, such as enterprise application integration, runtime system, Cloud computing, big data, mathematical modelling, optimisation techniques, statistical models and multithread programming.

ii. State-of-the-art review

   In this part, we studied runtime systems from popular integration platforms. Our goal was to find out weaknesses of the runtime systems regarding properties that impact on the performance of the execution of integration processes.

iii. Task scheduling study

   In this part, we mathematically represented the integration processes, characterised the tasks scheduling of integration processes, and to proposed alternative scheduling heuristics.

iv. Simulator development

   In this part, we built a tool to measure the gain in the performance of executions of integration processes with the different heuristics.

v. Mathematical modelling

   In this part, we developed a mathematical model to describe the gain in the performance of executions of integration processes with the proposed heuristic.

vi. Proposal validation

In this part, we performed experiments to compare the proposed heuristic with a popular heuristic, commonly used by open source integration platforms.

vii. Thesis writing

The last part contemplates the closing of the works, evaluation of the obtained results, and writing this thesis.

1.5 Summary of contributions

We enumerated the contributions of this research and the works that were already published or are under revision.

Our general results

- A representation that typifies the tasks of integration processes according to their operation logic.
- A characterisation of the task scheduling of integration processes carried out by runtime systems based on current approaches, classification, problems, and methods used.
- A scheduling heuristic that addresses the execution of integration processes in overload situations.
- A mathematical model and performance metrics definition as a function of integration processes properties.
- A simulation tool that allows the evaluation of heuristics for task scheduling of integration processes carried out by runtime systems.

Our main publications

- In the article entitled "A Survey on the Run-time Systems of Enterprise Application Integration Platforms Focusing on Performance" [68], we did a literature review regarding the performance of runtime systems of integration platforms. We introduced a comparison framework composed of performance properties that allow the analysis and comparison of open source integration platforms. This article was published in the Software Practice and Experience journal. Figures and tables from this article are used similarly in Figures 2.1, 2.2, and Tables 2.1, 2.2, 2.3, 2.4, and 2.4.

- In the article entitled "A Methodology to Rank Enterprise Application Integration Platforms from a Performance Perspective: An Analytic Hierarchy Process-based Approach" [66], we proposed a methodology to support software engineers in the decision-making process for an integration platform when performance is a central requirement. This methodology was based on the analytic hierarchy process (AHP) method [163], but also added objective criteria for performance assessment purposes. This article was published in the Enterprise Information Systems journal.

- In the article entitled "Ranking Enterprise Application Integration Platforms from a Performance Perspective: An Experience Report" [64], we reported our experience in evaluating and comparing four well-known open source integration platforms in the context of in which performance was a central requirement to choose an integration platform. The evaluation was conducted using our decision-making methodology to build a ranking of candidate platforms by means of subjective and objective criteria. This article was published in the Software Practice and Experience journal.

- In the article entitled "Optimisation of the Size of Thread Pool in Runtime Systems to Enterprise Application Integration - A Mathematical Modelling" [67], we proposed a mathematical formulation which characterised the costs associated with adopting the thread-by-request and pool of threads architectures as well as obtained the optimum size of the pool of threads, maximising the expected gain by minimising the execution time of integration processes. Furthermore, we applied our mathematical formulation and analysed the expected gain with the use of the pool of threads architecture. This article was published in the Trends in Applied and Computational Mathematics journal.

- In the article entitled "Task Scheduling Optimisation on Enterprise Application Integration Platforms Based on Meta-heuristic Particle Swarm Optimisation" [169], we proposed an algorithm for task scheduling based on the PSO meta-heuristic, which allocate computational resources to execution of tasks of integration processes, considering the waiting time in the queue of ready tasks and the computational complexity of the tasks in order to optimise the execution of integration processes. This article was published in the Proceedings of the 31st Brazilian Symposium on Software Engineering.

- In the chapter entitled "Experimental Study for Evaluating the Performance of Java Virtual Machines in Application Integration", we evaluated the behaviour of the concurrent execution of tasks in different implementations of Java virtual machines and analysed performance metrics by rigorous statistical techniques. This chapter was published in the Nova Science Publishers book, Inc. NY, USA.

- In the article entitled "Execução de Soluções de Integração de Aplicações Empresariais na Nuvem: Perspectivas e Desafios" [168], we introduced the research context regarding the fair allocation of computational resources to execution of integration processes in integration platforms offered as cloud services and discussed ideas to develop a theoretical model for improvement of runtime systems. This article was presented in the IV Seminário de Formação Científica e Tecnológica, in Ijuí, Brazil.

- In the article entitled "New developments in Round Robin algorithms and their applications: systematic mapping study", we provided a systematic mapping study that identifies the state-of-the-art in the research of the Round Robin algorithms to guide researchers and practitioners in the field of software engineering. The mapping showed that the research regarding the improvement in Round Robin algorithm continues active, indicating that Round Robin remains one of the more efficient scheduling techniques in the fields of research of packets, CPU and virtual machine scheduling. This article is under revision in the Journal of Scheduling.

1.6 Structure of this thesis

This doctoral thesis is organised as follows:

Part I: Introduction. Comprises this introduction, Chapter 2 provides a literature review about the state-of-the-art of open source integration platforms, which produced a comparison framework for runtime systems and identified their weakness regarding performance properties, and Chapter 3 describes the research gaps found in the literature review and indicates one that we approach in this work.

Part II: Background Information. Provides information related to our research context. In Chapter 4, we introduce the well-known enterprise application integration conceptions. In Chapter 5, we introduce the main concepts, classifications, methods and approaches regarding task scheduling. In Chapter 6, we describe some of the scheduling heuristics and meta-heuristics. In Chapter 7, we discuss statistical techniques, namely, ANOVA, Scott & Knott, and Regression analysis.

Part III: Our Proposal. Reports on the core contributions we made with this thesis. This part starts at Chapter 8, we propose and apply a representation for integration processes. Chapter 9, we characterise task scheduling in execution of integration processes. In Chapter 10, we propose the Queue-Priority heuristic to tackle overload situation. Chapter 11, we propose a mathematical model and a performance metric, as a function of the integration processes, the workloads, and the scheduling heuristics

Part IV: Validation. Describes the experiments and statistic tests to validate the application of the proposed heuristic to integration process problems. In Chapter 12, we present a simulation tool for integration processes task scheduling. In Chapter 13, we describe and apply the experimental protocol used to evaluate the performance of the proposed heuristic.

Part V: Final Remarks. Concludes this thesis and highlights a few future research directions in Chapter 14.

Chapter 2

Runtime System State-of-the-art

All sorts of things can happen when you are open to new ideas and playing around with things

Stephanie Kwolek, American chemist (1923–2014)

In this chapter, we present the state-of-the-art of open source integration platforms, addressing the performance properties of the execution of integration processes. To compare the integration platforms, we built a comparison framework to evaluate nine integration platforms regarding performance of their runtime systems.

2.1 Research method

We present the research method we follow in reviewing integration platforms and finding out research gaps in runtime systems performance of integration platforms. We constructed a comparison framework made up of performance properties and applied this framework to nine integration platforms.

The research method for this review is abstracted in Figure 2.1. It has two main activities: framework construction and framework application. In the former, we conducted a feature analysis by means of a qualitative screening research on academic literature [102] and on technical literature [69] to identify performance properties that can be used by academy and industry. In the latter, the comparison framework was applied to nine integration platforms and resulted in the identification of six research gaps. Screening

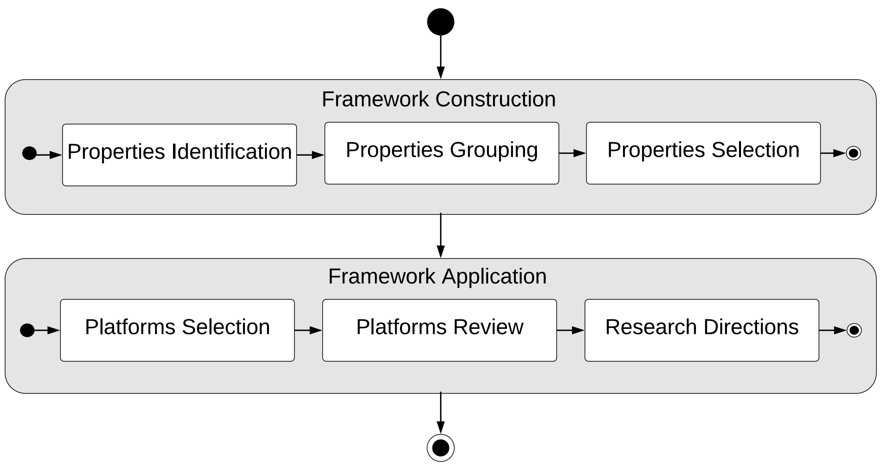


Figure 2.1: *Research method for review of the state-of-the-art.*

research is one of the forms of feature analysis, in which the evaluation is performed by software engineers based on documentation only. Screening is indicated for a more complex evaluation, in which it is possible to reduce a large number of platforms to a short-list that can be deeply evaluated [103].

Feature analysis is constructed from the selected research papers in multi-vocal literature review (MLR). Ogawa and Malen [136] state that "Multi-vocal literature" is comprised of all accessible writings on a common, often contemporary topic. The writings embody the views or voices of diverse sets of authors (academics, practitioners, journalists, policy centres, state offices of education, local school districts, independent research and development firms, and others). The writings appear in a variety of forms. They reflect different purposes, perspectives, and information bases. They address different aspects of the topic and incorporate different research or non-research logics". MLRs recently started to be used in software engineering, thus we conducted and reported our study based in the guidelines documented by Garousi et al. [69] to ensure high quality of our MLR processes and their results.

Framework construction

In this activity, we produced a comparison framework to analyse the runtime systems of the integration platforms. This framework provides properties that contribute to endow current runtime systems of integration platforms with quality attributes to increase their performance and at the same time can lead to efficiency in the execution of integration processes. This activity is based on the coding process [183], by which data is broken down, conceptualised, and synthesised. The main coding procedures are:

- Properties Identification - aims at identifying quality attributes for runtime systems that allow to compare and group similar performance properties. We constructed an initial list of properties that have an impact on the processing time of messages in the integration processes and in the consumption of computational resources of the runtime system.
- Properties Grouping - aims at identifying which problems in the execution of integration processes are solved when the runtime systems are endowed with the identified properties. We classified the properties into dimensions and tabulated the relationships between properties and dimensions. This tabulation allowed us to identify redundancies amongst the dimensions.
- Properties Selection - aims at evaluating the dimensions and properties to select a consistent subset of them. This selection was carried out based on our experience of several years on the development of integration projects in real-world software ecosystems. The subset of dimensions and their properties is the comparison framework used as criteria to evaluate integration platforms, in relation to the performance of their runtime systems.

This activity resulted in a set of ten performance properties grouped in two dimensions: message processing and fairness execution. The former dimension relates to improving the efficiency of processing a message by runtime system, which can also be seen as increasing the average number of messages processed per unit of time. The latter dimension is concerned with the assignment of threads to tasks aiming at a fair execution to help to minimise the average time that a message takes to be processed in the integration process.

Once the coding derived quality attributes, it is required to contextualise the platforms in comparison to each other. In other words, we aim at evaluating the quality performance features of integration platforms that meet three criterion: open source, provide support to the integration patterns [85], and follow the pipes-and-filters architectural style [2]. Thus, we aim at answering the following research questions:

RQ1: What are the relevant features that can help to improve the performance of runtime systems of the integration platforms?

RQ2: What are the state-of-the-art open source integration platforms that provide support to integration patterns and follow the pipes-and-filters architectural style?

RQ3: Are the state-of-the-art open source message-based integration platforms endowed with features that can help to improve the performance of their runtime systems from the perspective of message processing and fairness execution in the context of Cloud computing?

We hypothesise that the runtime systems are not sufficiently endowed with features to improve message processing and to promote a fair execution of tasks in the context of Cloud computing. In this review, we found out research gaps regarding the performance of the runtime systems in this context, that usually have to cope with large volumes of data.

Framework application

In this step, we applied the comparison framework for nine integration platforms. The evaluation of the platforms was carried out based on a deep study of them and in our knowledge of their use in integration projects in real-world software ecosystems. The main procedures are listed below and detailed in next sections:

- Platforms Selection - aims to select integration platforms, in which it is possible to compare their quality attributes regarding performance . We collected 42 platforms and formed an initial list. After, we applied inclusion criteria to filter and homogenise the set of selected platforms, resulting in nine platforms.
- Platforms Review - aims at identifying values for performance properties of the comparison framework. We carried out a multi-vocal literature review on the publicly available source-code and the documentation from their web site, books, and academic chapters in order to study and evaluate integration platforms.
- Research gaps - aims at identifying the lack of features that can increase the performance of runtime and at the same time can lead to efficiency in the execution of integration processes. Based on this review, we indicate research gaps that can be explored in order to improve the integration platforms and at the same time may be useful to adapt them to the context of Cloud computing.

2.2 Evaluation framework

We present a set of properties that guided our analysis of the integration platforms to answer the first research question (RQ1). These properties are grouped into two dimensions and can contribute to current runtime systems by endowing integration platforms with performance features, and, at the same time, lead to an efficient execution of integration processes. The following sections detail the respective properties in each dimension.

Message processing

This dimension addresses the improvement of the efficiency of the runtime system to process a message, which can also be seen as increasing the average number of messages processed per unit of time. The properties of the message processing dimension are related to the capacity of reducing the demanded real-time to completely process a message by an integration process. These properties are described below:

- **Designed for Multi-core**. This property indicates whether the runtime system has been developed to take advantage of multi-core or not. Multi-core programming has to be carried out in order to take full advantage of multiple cores available in the hardware processors. Nowadays, this becomes an increasingly important requirement in the use of the powerful multi-core parallel machines that compose the computing infrastructures [93]. Multiple cores work together to increase the capability of processing multiple tasks or to increase the performance of the system by operating on multiple instructions simultaneously in an efficient manner. This property may take the following values: `yes` or `no`. `yes` indicates that the runtime system was developed to take advantage of multi-core hardware; otherwise, the value is `no`.

- **Thread Pool Configuration**. This property indicates how threads are managed in thread pools. The programming languages, in which the runtime systems are developed, have objects that encapsulate functions to create and manage threads with predefined settings. Furthermore, they also have utility methods that allow a custom configuration to provide more flexibility, such as setting the maximum number of threads allowed in a thread pool, the maximum time a thread remains idle, and the type of queue used for the tasks waiting to be executed. This property may take the following values: `fixed`, `limited`, or `elastic`. `fixed` indicates that a thread pool is composed of a fixed number of threads, which is known at design time; `limited` indicates the number of threads in a thread pool can increase automatically during runtime until a threshold defined at design time is reached; `elastic` indicates that the number of threads in a pool can automatically increase and decrease during runtime within a range of values defined at design time [73].

- **Type of Message Storage in Process**. This property indicates how the runtime system deals with the storing of messages during the execution of an integration process. Storing messages in-memory is faster but can be more expensive [15]. Messages that contain a big amount of data impact the amount of memory required for their processing inside the integration processes. In such cases, rather than storing messages only in-memory, the runtime system can store them on-disk. This property may take the following values: `memory` or `hybrid`. `memory` indicates that the runtime system stores messages only in-memory. `hybrid` indicates that the runtime system adopts different strategies for storing messages, such as combining in-disc and in-memory.

- **Distributed Process Execution**. This property indicates if an integration process can be divided and distributed to different machines to execute a set of correlated messages, thereby promoting scalability [81]. This property allows increasing the number of tasks executed per unit of time in cases in which there is no dependency between the tasks, and the input data of one task is the output data produced by another task. This property may take the following values: `yes` or `no`. `yes` indicates that the runtime system takes advantage of scalability; otherwise, the value is `no`.

- **Thread Pool Creation**. This property indicates the way and the stage at which thread pools can be created. Historical and current execution data can be used by runtime systems to make decisions during runtime [134]. Such data can point to optimised strategies for the creation of threads, empowering runtime systems to deal with different workloads of messages. This property may take the following values: `dynamic` or `static`. `dynamic` indicates that thread pool creation is done at runtime by means of information taken from the runtime system. `static` indicates that the thread pool is created at design time by software engineers.

Fairness execution

This dimension addresses the assignment of threads to execution of tasks, in a balanced way, to minimise the average time that a message takes to be processed in an integration process. The following properties provide means that contribute to a fair execution of tasks:

- **Starvation Detection**. This property indicates if the runtime system is endowed with the capacity to detect tasks that do not execute within an accepted time-frame. Roughly speaking, starvation happens when a thread cannot fetch tasks because other threads have effectively blocked it from doing so [27]. This property may take the following values: `yes` or `no`. `yes` indicates that the runtime system is endowed with intelligence to detect hotspots during the execution of the integration processes; otherwise, the value is `no`.

- **Task Scheduling Strategy**. This property indicates the policy followed by the runtime system to schedule the execution of tasks of an integration process to computational resources. In cloud environments, scheduling of an integration process becomes challenging, because its performance must result in reduced scheduling overhead, minimised cost, and maximise resource utilisation while still meeting the specified deadline [9]. However, cloud environments usually cause computing overheads that negatively impacts on the overall performance and costs of the workflow execution [34]. This property may take the following values: `fifo`, `priority` or `mapping`. `fifo` means that the runtime system follows First-In-First-Out policy, in which threads are assigned to tasks in order that they arrive; `priority` means that the runtime system allows tasks to have priority associated to them, so influencing the scheduling; and, `mapping` means that the scheduling policy follows a mapping based on a mathematical model or optimisation method that allows finding an optimal scheduling policy by previously evaluating the integration process.

- **Task Complexity**. This property indicates if the runtime system takes into consideration the computational complexity of tasks to assign threads. Tasks that perform more complex operations tend to be implemented with more computational instructions, therefore require a longer time to be executed. Task granularity must be considered for reducing the impact of overheads on the execution of an integration process in the cloud environment, since it can lead to an inefficient resource utilisation, resulting in an unfavourable application throughput [131]. This property may take the following values: `yes` or `no`. `yes` indicates that the runtime system recognises the computational complexity of each of the tasks and this can be utilised to decide the order of execution of tasks, so that tasks of less computational complexity can be executed first; if the runtime system does not recognise the computational complexity, the value is `no`.

- **Execution Model**. This property indicates the execution model implemented by the runtime system, which deals with the level of the execution of an integration process. It is possible to classify these models in: process-based and task-based. In the former, the runtime system controls process instances as a whole, i.e., there are no means that it can interact with the internal tasks. In the latter, the runtime system may control both process instances and their internal tasks. The literature shows that the task-based model offers better performance with a steady stream of data input and lower performance when the input rate increases [59], although this model is more complex to provide transaction and fault-tolerance support [60]. This property may take the following values: `process-based`, `task-based` or `hybrid`. `process-based` indicates that the runtime system adopts a process-based model; `task-based` indicates that the runtime system adopts a task-based model. `hybrid` model, indicates that the runtime system will adopt the model which best fits the execution profile regarding predefined parameters, such as message input rate, number of processors, or average message size.

- **Throttling controller.** This property indicates if the runtime system allows to control the rate of incoming messages in an integration process, so that when this rate exceeds a previously determined limit, the runtime system can adopt suitable policies to preserve the execution of the integration process. Such intervention policies may be: (i) refusing new messages or (ii) buffering at input the incoming messages or persisting them in a repository. This property may take the following values: `no` or `yes`. `yes` indicates that the runtime system can control the rate of incoming messages, that is, it has a throttling controller [85]; otherwise, the value is `no`.

2.3 Selection of the integration platforms

We present the research methodology that we applied to select the integration platforms analysed in this chapter. Motivated by the second research question (RQ2), we reviewed the literature to identify the state-of-the-art open source message-based integration platforms and to conduct a study of their runtime systems. The research methodology is abstracted in Figure 2.2 and is composed of three steps: collection of references, collection of integration platforms, and selection of integration platforms.

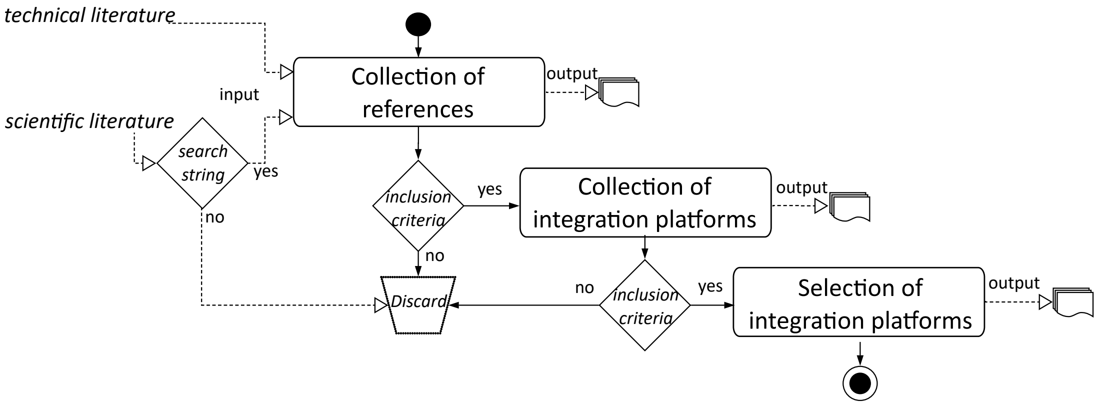


Figure 2.2: *Methodology for selection of the integration platforms.*

In the step for collection of references, we selected scientific chapters and technical reports regarding integration platforms based on predefined search string. In the step for collection of integration platforms, we analysed these documents and extracted a list of integration platforms based on inclusion criteria "cited platforms explicitly". In the step for selection of integration platforms, we selected integration platforms based on inclusion criteria "open source, provide support to the integration patterns [85], and follow the pipes-and-filters architectural style [2]". The methodology and steps are explained in the following sections.

Collection of references

In this step, we performed a search on SCOPUS using the following search string: ("enterprise application integration" or "integration systems" or "business integration") and ("tool" or "platform"). This study searched for published chapters from January 2013 to April 2018 (date that the research was done), written in English, in the subject area of Computer Science. The search returned 158 unique results, which cover a diverse range of journals and conferences.

Next, titles and abstracts of these 158 chapters were carefully reviewed to select those chapters that explicitly make reference to at least one integration platform. At the end, there were 19 chapters. Additionally, we revised the technical literature by means of 3 reports regarding integration platforms: Gartner [77, 78], Forrester [144], and Ovum [176]. Finally, they were grouped by year as shown in Table 2.1.

Literature	Year	Collected	Selected	References
scientific	2013	29	3	[54, 88, 132]
	2014	20	2	[104, 141]
	2015	31	4	[82, 109, 138, 171]
	2016	38	5	[20, 51, 61, 107, 155]
	2017	30	4	[18, 21, 169, 191]
	2018	10	1	[154]
technical	2016	-	1	[144]
	2017	-	2	[77, 176]

Table 2.1: *Review of the state-of-the-art integration platforms.*

Collection of integration platforms

In this step, we collected 42 platforms from the chapters and reports, cf. Table 2.2. In this table, the first column identifies the platform; the second column indicates if the platform is released under an open source licence; the third column indicates if the platform supports the enterprise integration patterns (EIPs) [85]; the fourth column indicates if the platform adopts the pipes-and-filters architectural style [2]; and, the last column provides the references to the selected scientific chapters and technical reports.

Integration Platform	Open source	EIP	P&F	References
Actian	✗	✗	✗	[77]
Adaptris	✗	✗	✗	[51]
Adeptia	✗	✗	✗	[77, 144]

AdroitLogic	✓	✗	✗	[21]
Apache Camel	✓	✓	✓	[54, 61, 104, 154, 155]
Apache Synapse	✓	✗	✓	[21]
Attunity	✗	✗	✗	[51, 77]
Azuqua	✗	✗	✗	[51]
Babelway	✗	✗	✗	[51]
Built.io	✗	✗	✗	[51, 77]
Celigo	✗	✗	✗	[51, 77]
DBSync	✗	✗	✗	[77]
Dell Boomi	✓	✗	✗	[51, 77, 144]
Elastico.io	✗	✗	✗	[51]
Fiorano ESB	✗	✓	✗	[21]
Flowgear	✗	✗	✗	[51, 144]
Fujitsu	✗	✗	✗	[77]
Guaraná	✓	✓	✓	[20, 61, 104, 107, 154, 171]
Fuse	✓	✓	✓	[21, 132]
IBM	✗	✗	✗	[18, 21, 51, 77, 132, 138, 144]
Informatica	✗	✗	✗	[51, 77]
Jitterbit	✓	✓	✓	[51, 77, 144]
Microsoft	✗	✗	✗	[18, 21, 51, 77, 132]
Moskitos	✗	✗	✗	[77]
Mule	✓	✓	✓	[21, 51, 61, 77, 104, 132]
Oracle	✗	✗	✗	[18, 21, 51, 77, 132]
Petals	✓	✓	✓	[132]
SAP	✓	✗	✗	[51, 77, 132]
Scribe Software	✗	✗	✗	[51, 77, 144]
ServiceMix	✓	✓	✓	[132]
Skyvva	✗	✗	✗	[51]
SnapLogic	✗	✗	✗	[51, 77, 144]
Software AG	✗	✗	✗	[51]
Sonic ESB	✗	✓	✗	[21]
Spring Integration	✓	✓	✓	[61, 104]

Talend	✓	✗	✗	[21, 51]
TerraSky	✗	✗	✗	[51, 77]
Tibico	✗	✗	✗	[21, 51, 132]
Vigence	✗	✗	✗	[51]
Workato	✗	✗	✗	[144]
WSO2	✓	✓	✓	[21, 109, 132]
Yourede	✗	✗	✗	[51, 77]

Table 2.2: Integration platforms found in the literature

Selection of integration platforms

In this step, we filtered the 42 integration platforms by considering only those that are released under an open source licence, support the EIPs, and that adopt the pipes-and-filters architectural style. We opted by open source integration platforms because the performance properties that we wanted to investigate required access to their source codes. The option by integration platforms that adopt the integration patterns of [85] and the pipes-and-filters architectural style intended the homogenisation of the set of studied integration platforms. At the end, 9 integration platforms were selected to be analysed, cf. Table 2.3. The chosen platforms were Camel [91], Guaraná [59], Fuse [160], Jitterbit [161], Mule [49], Petal [162], ServiceMix [105], Spring Integration [58], and WSO2 [92]. The analyses of the platform was carried out considering books, online documentation, and source code accessible from their web sites.

2.4 Review of the integration platforms

We analyse the runtime systems of the selected integration platforms to answer the third research question (RQ3). This analysis is guided by the evaluation framework previously introduced in Section 2.2, and aims to infer the values for the properties in the message processing and fairness execution dimensions. Results are summarised in Tables 2.4 and 2.5 and discussed in the following sections. It is important to note that every selected integration platform was developed using Java programming language, which may result in the same values for different platforms.

Integration Platform	References
Apache Camel	[54, 61, 104, 154, 155]
Guaraná	[20, 61, 104, 107, 154, 171]
Fuse	[21, 132]
Jitterbit	[51, 77, 144]
Mule	[21, 51, 61, 77, 104, 132]
Petals	[132]
ServiceMix	[132]
Spring Integration	[61, 104]
WSO2	[21, 109, 132]

Table 2.3: *Selection of the open source integration platforms.*

Message processing

We will now discuss how the runtime systems of the selected integration platforms meet the properties that can improve the efficiency in message processing. None of the platforms have their runtime system endowed with features to take advantage of multi-core design, however the multi-core hardware is currently found in most commodity computers. Mule uses the integration pattern [85] Scatter Gather to execute tasks concurrently and independently. Camel implements integration patterns including Multicast, Splitter, and Aggregator, each one of these patterns providing a custom thread pool. Jitterbit has a feature that splits up the source data for parallel processing; each part is processed in isolation, and it is possible to process several parts in parallel. However, none of the platforms take advantage of multi-core programming to execute integration processes.

Regarding thread pool configuration, every runtime system that we have analysed is endowed with a feature to define a limit for the number of threads to be created during runtime. Petals and Jitterbit do not provide information that allows evaluating them regarding this property, while Guaraná uses a thread pool with a fixed number of threads. Mule allows software engineers to configure of threading profiles in three different ways: configuration, connector, and flow. Threading at configuration sets default threading profiles for all tasks. Threading at connector sets a threading profile for specific tasks, for example, one profile for tasks to receive messages and another to dispatch messages.

	Integration Platforms								
Property	**Mule**	**Camel**	**Spring**	**Fuse**	**Service Mix**	**Petals**	**Jitterbit**	**WSO2**	**Guaraná**
Designed for multicore	*no*	*no*	*no*	*no*	*no*	*N/A*	*no*	*N/A*	*no*
Thread pool configuration	*limited*	*limited*	*limited*	*limited*	*limited*	*N/A*	*N/A*	*limited*	*fixed*
Type of message storage in flow	*hybrid*	*hybrid*	*hybrid*	*hybrid*	*memory*	*memory*	*hybrid*	*memory*	*memory*
Distributed process execution	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*
Thread pool creation	*static*	*static*	*static*	*static*	*static*	*N/A*	*static*	*static*	*static*

N/A = Not Available

Table 2.4: *Runtime systems comparison in message processing dimension.*

Property	Integration Platforms								
	Mule	**Camel**	**Spring**	**Fuse**	**Service Mix**	**Petals**	**Jitterbit**	**WSO2**	**Guaraná**
Starvation detection	*no*	*no*	*no*	*no*	*no*	*no*	*N/A*	*no*	*no*
Task scheduling strategy	*fifo*	*fifo*	*fifo*	*fifo*	*fifo*	*fifo*	*fifo*	*priority*	*fifo*
Task computational complexity	*no*	*no*	*no*	*no*	*no*	*no*	*N/A*	*N/A*	*no*
Execution Model	*process-based*	*process-based*	*process-based*	*process-based*	*process-based*	*process-based*	*process-based*	*process-based*	*task-based*
Throttling controller	*yes*	*no*	*yes*	*no*	*no*	*no*	*yes*	*no*	*no*

N/A = Not Available

Table 2.5: *Runtime systems comparison in fairness execution dimension.*

Threading at flow sets a threading profile for a sequential flow of tasks [130]. Both Camel and ServiceMix offer a fine-grained configuration in which it is possible to tweak individual thread pools and have more coarse-grained configuration with fall back to global settings; furthermore, it should be possible to define a set of rules which matches the thread pool to a given source, i.e., task or group of tasks [10]. Every runtime system that we have analysed have subclasses that extend classes of the Java language, whose methods allow to configure the thread pool. Camel and ServiceMix allow configuring settings of thread pool, whereas WSO2 has a file property that contains parameters for thread pool configuration [200].

Regarding the type of message storage in processes, Mule, Camel, Spring Integration, Fuse, ServiceMix, and Jitterbit can store data in-memory and in-disc. ServiceMix and WSO2 do not provide information that allows them to be evaluated regarding this property, and Guaraná stores messages only in-memory. Mule allows for storing data in-memory or on-disk for eventual retrieval, such as recovered data from processing into tasks like filters, routers, and other ones that need to store stateful messages. In the case of in-memory storage, Mule allows for storing data in a local memory of the runtime system, in which messages are dropped during shut-down of the runtime. In the case of a persistent store, Mule persists data when explicitly configured to do that. In a standalone Mule runtime system, Mule creates a default persistent store in the file system [130]. Mule allows for dealing with streamed data by configuring an initial memory buffer size of 512 KB; if the stream is larger than this, Mule creates a temporary file on-disk to store the contents without overflowing memory; if the stream is larger than 512 KB, the buffer is expanded to a default increment size of 512 KB until it reaches the configured maximum buffer size; when the stream exceeds this limit, the integration process fails. Camel and ServiceMix allow for saving messages in a persistent store that can be a file or a database [91]. Camel supports strategies to deal with streams; it can buffer all messages in an unbounded buffer, or choose to keep only the latest or oldest message and drop all the others. Spring Integration defines a Message Store pattern, which allows components to store messages typically in some type of persistent store, in addition to the capability of buffering messages [127]. Fuse offers a number of different mechanisms for persistence besides the default message store. By default a hybrid system that couples a data logs for message storage and a reference store for quick retrieval is used; another option allows for distributing the messages across multiple message stores or a file-based message store that maintain indexes into log files holding the messages that can be used. Additionally, Fuse supports the use of relational databases as a message store through the Java Database Connectivity feature, in which the persistence adaptor may be either coupled with a high data log or used in standalone mode [151]. To facilitate a rapid access to the content of the log, the message store constructs meta-data to index the data embedded. Jitterbit is also able to persist message on-disk [98].

The execution of integration processes in a distributed way makes the runtime system more suitable for Cloud computing environments [90], by providing greater processing power for tasks that can be processed in parallel across multiple virtual machines. However, it is important that the data transfer time from one machine to another minimises the total message processing time since the total processing time increases, and the distributed processing will be detrimental to the performance of the runtime system. The ability to distribute the execution of tasks amongst several virtual machines is present in every runtime system analysed, except for Guaraná.

Mule has a virtual runtime system composed of multiple nodes, which ensure high system availability to perform distributed processing. It is possible to configure a cluster in Mule for an integration process to maximise performance using a profile of performance. By implementing the performance profile for specific applications within a cluster, it is also possible to maximise the scalability of the deployments while deploying applications with different performance and reliability requirements in the same cluster [130]. Camel, Fuse, and ServiceMix bring different technologies to allow their integration processes to be scalable and to distribute the load amongst different instances, such as load balancing, clustering, and Cloud computing. The load balancing approach allows for distributing the load amongst different *proxys*. Clustering can be achieved by means of one or several instances of the runtime system running on the same machine or distributed in virtual machines in a Cloud computing environment. For Cloud computing, it is possible to consume or push messages to Cassandra NOSQL database [6, 10]. Spring Integration provides a consistent model for intra-process and inter-process messaging implemented using Java Message Service [58].

Petals distributes its processes either statically or dynamically. Statically, no new node can be added to a running Petals cluster. Dynamically, this distribution may be updated regularly, so new nodes can be added to a running Petals cluster [143]. Jitterbit provides high availability and load balancing of integration operations across runtime systems within a group. This platform is automatically scaled within the cloud as necessary, and does not require adding new runtime systems to expand capacity [98]. WSO2 implements a distributed process by means of two models.

The first model consists of two sub-cluster domains as a worker domain and a management domain. These sub-domains take up loads according to a defined load balancing algorithm and auto-scales according to the load on its nodes. The second model consists of a single cluster, in which a selected node works as both a worker and a manager. This worker node requires two load balancers and has to be configured in read/write mode. The others are set up in read-only mode. The management node also should be a well-known member in the non-management worker nodes so that state replication and cluster messaging works [200].

Regarding thread pool creation, none of the analysed runtime systems is able to dynamically create pools of threads to optimise task execution strategies from the analysis of the flow of messages in the integration process. In Camel, it is possible to define a set of rules that matches which thread pool a given source should use. Petals does not provide information that allows it to be evaluated regarding this property.

From the analysis of the properties that may have an impact on message processing, we observe that Mule, Camel, and Jitterbit are advancing in terms of parallel programming, although none of them has actually benefited from multi-core programming. There has been a growing need for a mechanism to runtime system that gives software engineer a simple, yet effective way to make use of multiple processors in a clean, scalable manner. It is need to enable the runtime systems to automatically scale to make use of the number of available processors. The multithreading programming must be optimised for situations in which the runtime system is able to execute portions of an integration process simultaneously, with each part executing on its own CPU. This can be used to significantly speed up the execution of some types of workflows that presented a set of tasks that can be processed in parallel. Parallel programming should be thought of as a possible improvement in integration platforms because it offers a way to significantly improve integration process performance. The majority of them already manages threads but are not endowed with the elasticity feature for configuration of thread pools, in which the size of these computational resources changes proportionally to the demand of tasks at runtime. Most of them are equipped with the ability to deal with large volumes of data, including features to store data either in-memory or on-disk. Also, they advanced the issue of exploiting the benefits of distributed processing; except for Guaraná, every runtime system analysed is endowed with the feature for distributing the execution of tasks amongst several virtual machines. None of them is equipped with the ability to create thread pools dynamically, demanding the expertise of software engineers to create thread pools at design time.

Fairness execution

In the following, we discuss how the runtime systems of the selected integration platforms are meeting the properties that can improve a fair execution of tasks of an integration process.

Regarding starvation detection, none of the platforms is endowed with the ability to detect tasks that are not executed within an accepted time frame, except for Jitterbit, which does not provide information that allows to evaluate it regarding this property. The absence of support to this feature can lead to a risk of tasks to remain waiting forever for a thread to be assigned to it. Every runtime system that we have analysed can monitor the execution of integration processes, so that to detect bottlenecks that may appear during execution. Mule can use the management console to monitor the health of the runtime system, i.e., see which flows are running or stopped, and determine memory usage, which can be a clue for bottleneck detection. In this platform, it is also possible to view detailed information about the integration platform, including alerts, memory usage, threads, pools, files, operating system resources, and runtime system settings [130]. Camel has extensive support for Java Management Extensions to allow monitoring and controlling executions of integration processes [10]. Spring Integration allows to monitor message sources, enable metrics, to detect bottlenecks. Such metrics can count and measure the number of failed sent message, the mean message sent rate, the number of messages in queue, and the number of active threads [127]. Fuse and WSO2 use Java Management Extensions to monitor and manage resources that may themselves turn into bottlenecks, such as memory allocation, thread utilisation, data input and output operations, CPU consumption, and request processing time [11, 151]. Jitterbit provides an interface that allows to monitor every integration process to catch errors [98]. Petals provides metrics through a control development kit, such as current, maximum and minimum number of active threads, response time of tasks, and the execution time of a task [143].

Regarding task scheduling strategy, the heuristics adopted in every runtime system, which we have analysed, is FIFO, except for WSO2 that adopts a priority-based policy. The priority of tasks is an alternative implementation that allows tasks to be ordered within the channel based upon a priority. To prioritise the execution of tasks, WSO2 uses a Java class, which executes sequences of tasks with a given priority. This approach allows for software engineers to control the resources allocated to execute sequences and prevent high priority tasks from getting delayed and dropped [200]. This class is backed by a custom implementation which has multiple internal queues for handling separate priorities. The scheduling policy impacts on the total processing time of a message, since it is possible to give priority to tasks that need to be executed first or more frequently, avoiding them to wait in a queue for a time larger than necessary before being assigned to threads.

Regarding task computational complexity, none of the runtime systems is endowed with a feature to take into account the computational complexity of tasks in the assignment of threads, except for Jitterbit and WSO2, which do not provide information that allows to evaluate them regarding this property [131, 184].

Regarding the execution model, except Guaraná, the majority of the runtime systems adopt a process-based execution model. In spite of most of platforms adopt this model, they can process message synchronously or asynchronously. The synchronous approach is used to process messages in the same thread that initially has received the message. After the integration process receives a message, all processing, including the processing of the response, is done by the same thread. The asynchronous approach uses a queue to keep tasks that wait an available thread. In this case, a thread checks the task queue, then catch a task to execute. Thus, in the processing of a message, different threads can execute the tasks of the integration process [10, 92, 130, 139, 151].

Regarding throttling controller, Mule, Camel, Spring Integration, Jitterbit, and WSO2 allow to control the rate of incoming messages at input ports; this type of control is absent in Guaraná; Fuse, ServiceMix, and Petals do not provide information that allows to evaluate them regarding this property. Camel and Fuse support the control of inbound messages by implementing an integration pattern that allows the configuration of the size of the queue that connects the message producer and the polling consumer; or to block any message producer if the internal queue is full. In Mule, it is possible to arrange for tasks to actively consume messages at regular intervals of time [130]. Spring Integration and Jitterbit use a Java class for the same purpose [98, 127]. In WSO2, a polling inbound pattern allows for polling a given message source at regular intervals of time. The polling inbound pattern periodically checks for new messages in the message source, and if there are messages available, they are loaded into the integration process [92].

From the analysis of the properties that may have an impact on the fairness execution of the runtime systems, we observe that the integration platforms are similar. None of them is endowed with features that allow to detect starvation or consider task computational complexity. Most of them follow the FIFO policy for scheduling of tasks; adopt the process-based execution model; and have the ability of inbound control of messages. In this dimension, WSO2 differs from others platforms because it is the single platform endowed with the ability to prioritise to task execution.

2.5 Summary

In this chapter, we presented a literature and technical review on the runtime systems of integration platforms. We analysed properties which can have an impact on their performance when executing integration processes and evaluated the runtime systems of nine different state-of-the-art open source message-based integration platforms. It was possible to identify that runtime systems have advanced the configuration of thread pools regarding efficiency of message processing, since most of them allow the number of threads in a thread pool to be increased automatically during runtime until a threshold defined at design time is reached; most of them store messages in-memory and on-disk; all of them are able to distribute the processing. On the other hand, none of them has been designed to take advantage of multi-core and none of them is able to dynamically create thread pools. The platforms, however, have few features that allow the fair execution of tasks. None is able to detect tasks that have been waiting for a long time to be executed, i.e., starvation detection; most of them use basic heuristics as scheduling policy to tasks; none of them recognises the computational complexity of these tasks; most of them adopt the process-based execution model; and, only a small part of them has some kind of throttling controller to limit the incoming message rate.

Chapter 3

Research Problem

I was taught that the way of progress
was neither swift nor easy.

Marie Curie, Polish physicist (1867–1934)

From our literature review, we identify research gaps that have to be explored by researchers to improve the performance of the runtime systems of integration platforms and at the same time may be useful to adapt them to the context of Cloud computing. In this chapter, we describe these research gaps, the research problem approach, our proposal and objectives.

3.1 Research gaps

Performance has become a critical issue in the Cloud computing environment [9] and enterprises have been concerned about integration application to improve their business. We discuss the research gaps identified in our literature review. There are quality attributes that can endow integration platforms of features to enhance the performance of runtime systems. As a starting point, it is possible to look at other research fields and get inspiration from them to solve similar problems, so that these ideas can be studied and adapted to enterprise application integration in the context of Cloud computing. Table 3.1 summarises the properties that indicate the research gaps found; for each them, we discuss incipient studies identified in the literature, which permeate similar issues in different areas of knowledge, reiterating the pertinence of the focus on performance and the interest of the scientific community in addressing them.

Dimension	**Property**
Message Processing	Designed for multi-core
	Thread pool configuration
	Thread pool creation
Fairness Execution	Starvation detection
	Task scheduling strategy
	Task computational complexity

Table 3.1: *Research gaps.*

Gaps for message processing

Software engineers seek to develop algorithms that can take full advantage of multi-core in order to achieve a high level of parallelism and an overall high performance. The way algorithms are written strongly impact the success of multi-core technology [172]. Multi-core design is a property which should be present in runtime systems bearing in mind that the technological advances already offer resources to extend parallel programming. The Graphics Processing Unit (GPU) technology is an example of large scale processing resources, allowing for tens of thousands of concurrent threads, to be explored by software applications [205]. GPUs hold massive parallel computing capabilities with the potential of accelerating computationally intensive algorithms [188]. Measurements of thread pool throughput and measurements of thread utilisation in combination with analysis of prior thread pool re-sizing actions can be used to determine the increase or decrease of the number of threads from a thread pool in a current re-sizing action [134]. An exponential moving average scheme that adjusts the idle time-out period and thread pool size to adapt the system to the changing environment can be used to predict the number of threads and thread pool management, ensuring better response time and CPU usage [112]. Throughput degradation can be minimised by means of the creation of threads, based on the estimation of the range of threads needed, found via task arrival times and task processing times [137]. Measures of request frequencies can be used to dynamically optimise thread pool size, using non-blocking synchronisation primitives offering advantages of scalability and liveliness [13]. Research in methods of thread pool configuration and creation is needed, to support dynamic and optimal thread pool sizing in multi-threaded processing environments, in reaction to changes in workload and processing resources availability.

The previous studies motivate us to go deeper into the research that explores the advantages of multi-core in the context of integration platforms, as well as the elastic configuration of the size of the thread pools and the dynamic creation of these pools by the runtime system during runtime. These improvements would increase message processing and, consequently, allow for better performance on the integration platforms in the context of Cloud computing.

Gaps for fairness execution

The implementation of multiple thread pools based on a distribution of service times can avoid starvation and achieve concurrent processing, decreasing the response time and reducing the waiting time in the execution of tasks [174]. Studies indicate that task scheduling and resource allocation can be optimised by means of algorithms such as differential evolution algorithms based on the proposed cost and time models in Cloud computing [192]. Algorithms combining different policies, like shortest-job-first and round-robin schedulers [53], or other meta-heuristics techniques based on swarm intelligence and bio-inspired techniques could help to choose a suitable approach for better schemes for scheduling according to the application [181].

It is essential to balance the execution time of all tasks in a workflow running in parallel in order to achieve the best possible use of computing resources. One of the possible solutions to deal with the heterogeneity of the underlying platform is to use a model that to consider different execution times for loop iterations of the program. Thus, these execution times are taken into account to select the processing units according to its performance characteristics, as well as, to determine the number of processing units that are used simultaneously [7]. An efficient and economical way of using computational resources is to adopt policies and approaches for deciding the task granularity at runtime based on computational resource utilisation constraints, quality of service requirements, and the average task deployment metrics [131]. The prediction of the quantity of computational resources that are needed for a re-configurable architecture to suit task granularity can be used to make compatible resources and tasks [184]. Computational complexity should also be considered in the scheduling process to achieve better allocation of computational resources.

This incipient research reflects the concern of the scientific community regarding performance. Such ideas still have to be developed, applied, and experimented on the runtime systems of integration platforms in the context of Cloud computing. Thus, these and other studies point to solutions that can endow platforms with features that provide an efficient message processing and fairness execution of tasks, and, consequently, the runtime systems will achieve better performance in the execution of integration processes in Cloud computing.

3.2 Research problem approach

The current integration platforms present design challenges for providing near-real-time performance. These platforms manage petabyte-scale data and distributing integration processes across contemporaneous environments ranging from traditional on-premise servers to cloud systems and mobile devices [108, 202]. Enterprises also face the challenge to suit and integrate their applications, together with the optimisation of resource usage to save costs [71, 76, 79]. Amongst the research gaps identified in our literature review, the task scheduling strategy regarding the scheduling of tasks of integration processes and the resource allocation of the runtime systems to promote a fair execution in integration platforms, desearve special attention. In this research , we focus in this gap because it is the central role of the runtime systems [76, 83].

Task scheduling concerns with the time of task execution and with the allocation of computational resources that perform the tasks. A runtime system has threads, usually grouped into a thread pool, that represent the computational resources available to execute the integration process. The need for efficient scheduling has increased, to minimise costs when executing an integration process in an integration platform deployed on the cloud, due to the pay-as-you-go charging model [68]. Besides, it is necessary to optimise the use of computational resources because the processing of a large amount of data requires more allocation of resources [56, 179]. In overload situations due to high and continuous input data rate, the scheduling of tasks of the workflow tends to concentrate on tasks in the beginning of the workflow. This concentration causes the threads to execute more frequently tasks in the beginning, in contrast to the other tasks. This behaviour impacts negatively on the performance of the integration process.

It is possible to observe degradation in the performance as the workload increases when the task scheduling of the integration process is performed using the FIFO heuristic, c.f. shown in Figure 3.1. This evaluation was carried out by simulations, which consisted of running an integration process [84] with a fixed number of messages per experiment distributed in a time-frame of 60 seconds to simulate heavily-loaded scenarios. The total number of inbound messages in each experiment was 1,000, 10,000, 100,00, 500,000,

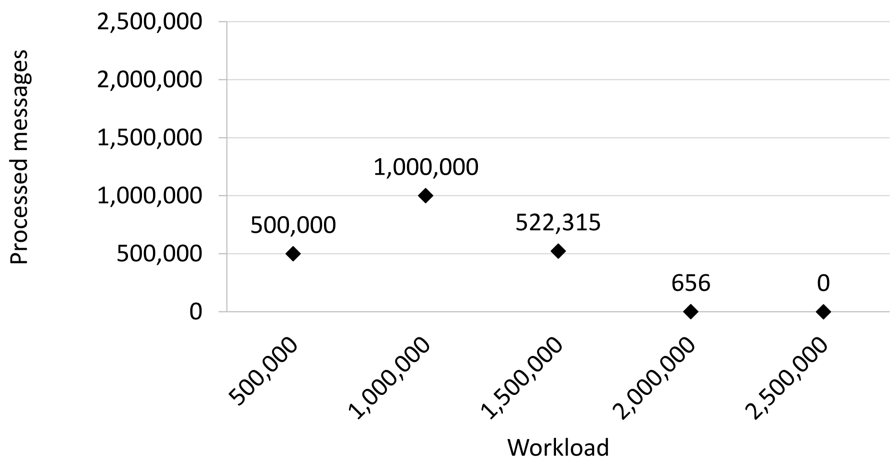


Figure 3.1: *Number of processed messages with FIFO heuristic.*

1,000,000, 1,500,000, 2,000,000, 2,500,000 msg. In total, we ran 25 repetitions for each experiment to draw our conclusions.

FIFO heuristic is more commonly used in the open source integration platforms. With a workload of 1,000, 10,000, 100,000, 500,000 and 1,000,000 inbound messages, the integration process produced 1,000, 10,000, 100,000, 500,000 and 1,000,000 outbound messages, respectively; with 1,500,000 inbound messages produced 522,315 outbound messages; with 2,000,000 inbound messages produced only 656 outbound messages; and, with a workload of 2,500,000 inbound messages, this integration process did not produce outbound messages. We consider that an integration process is in overload situation when the number of remained message is greater than the number of processed message.

There are several proposals of task scheduling heuristics, but as far as we are concerned, none of them were studied in the specific context of integration processes scheduling, taking into account integration process characteristics such as unknown message arrival rate, variable task processing time, unpredictable path of the workflow traced by messages and elastic resources provisioning. Building fair scheduling while increasing performance is a significant concern of enterprises, which submit an integration process for executing concurrently in different resources. Without the reengineering of the integration platforms, it is not possible to address these fundamental technical challenges. Besides, it is not possible to ensure that enterprises take advantage of the scalability provided by Cloud computing, optimise computational resources usage or can reduce their costs [117].

3.3 Our proposal

The task scheduling heuristic used by most open source integration platforms that follow the integration patterns documented by Hohpe and Woolf [85] and Pipes-and-Filters architecture [2] present satisfactory performance in the execution of integration processes for low workloads. However, it is increasingly common enterprises cope to large (of the order of millions) and continuous data volumes due to the fast-emerging of cloud and mobile applications, which are now part of their daily business. This large and continuous data volumes can cause an overload in integration processes and, hence, a degradation in the execution of these integration processes.

We propose a heuristic that addresses a new task scheduling policy for the execution of integration processes in the execution of integration processes in overload situation. A high arrival rate of large messages becoming, corresponding to millions of tasks to process, can become the execution of integration processes overloaded, characterised by the accumulation of tasks in internal queues awaiting computational resources in the runtime systems.

As demonstrated in subsequent sections, the innovative heuristic that we suggest is based on scheduling policies, rather than on FIFO like in most existing platforms. Then to this innovation, it responds more rapidly than using FIFO regardless of the incoming workload. Besides, it keeps the production of messages although with high workloads. The behaviour of the proposed heuristic is shown in Figure 3.2.

We verified that the increase of inbound messages rate did not harm under the number of outbound messages. FIFO heuristic is more commonly used in the open source integration platforms. With a workload of 1,000, 10,000, 100,000, 500,000, 1,000,000 and 1,500,000 inbound messages, the integration process produced 1,000, 10,000, 100,000, 500,000, 1,000,000 and 1,500,000 outbound messages, respectively; with a workload of 2,000,000 inbound messages produced 1,911,304 outbound messages; and, with a workload of 2,500,000 inbound messages produced 1,876,828 outbound messages. We validated our proposal by performing several experiments and confirmed its results with statistical tests.

Our research is a breakthrough towards dynamic scheduling for the execution of integration processes. Our goal is to contribute for a better understanding of scheduling of tasks of integration processes, so software engineers can upgrade the current integration platforms to avail Cloud computing better. To the best of our knowledge, this is the first work that deals with the characterisation of scheduling of tasks of integration processes.

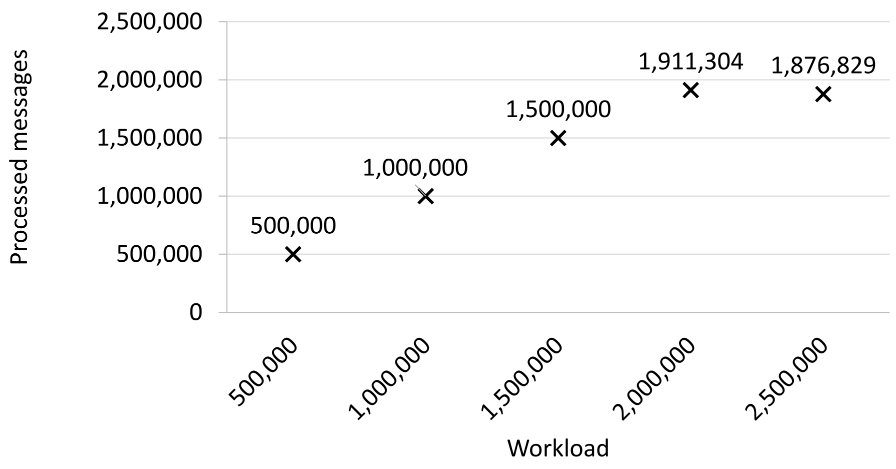


Figure 3.2: *Number of processed messages with qPrior heuristic.*

3.4 Summary

In this chapter, we summarised the why of the runtime systems of the current integration platforms need to be adapted. Indeed, runtime systems have to be endowed with a proper task scheduling heuristic to deal with overload situations brought of large and continuous volumes of data present in the contemporaneous environments. We have presented some reasons that we believe may be behind the inefficiency of the runtime systems in these situations. We also highlighted, experimentally, the behavioural differences of the commonly adopted heuristic and the heuristic proposed in this thesis.

Part II
Background Information

Chapter 4

Integration Process

For me, education was power.

Michelle Obama, American lawyer & writer (1867-1934)

Application integration is the conjunction of data and functionalities from amongst applications to improve the efficiency of the enterprises becoming their businesses more productive through faster processes, better information sharing, and smarter workflows. In this chapter, we concept the main terms of the application integration, explain the elements involved in task scheduling of integration processes and describe the task-based execution model for runtime systems of integration platforms. Task-based model is the approach followed in this thesis.

4.1 Integration process

An integration process is a computational program that supports the exchange of data and functionalities amongst applications to perform a «job». A job is a service request. The accomplishment of a job consists of receiving input data regarding service request, processing these data, and then, producing output data.

The input data of an integration process comes from one or more sources data. Whereas the output data of an integration process are send to one or more deliver data sinks. Sources and sinks of data can be applications, databases, sensors, amongst others. Data flow through the integration process wrapped in «messages», which are compose of two parts: header and a body. The header of the message can contain custom properties, such as identification or priority. The body has the payload data. A message can be split into one or more messages in the integration process, as well as two or more messages, can be merged in a unique message.

A conceptual model of an integration process is a workflow is a set of «segments», which in turn are composed of «tasks» uncoupled and connected by «communication channels». An example of an integration process is depicted in Figure 4.1, in which, the larger rectangle represents the integration process. Small rectangles inside it represent tasks and arrows connecting tasks represent communication channels. Rectangles outside the integration process represent applications that are being integrated. The highlighted segment represents a possible path for a message in the workflow.

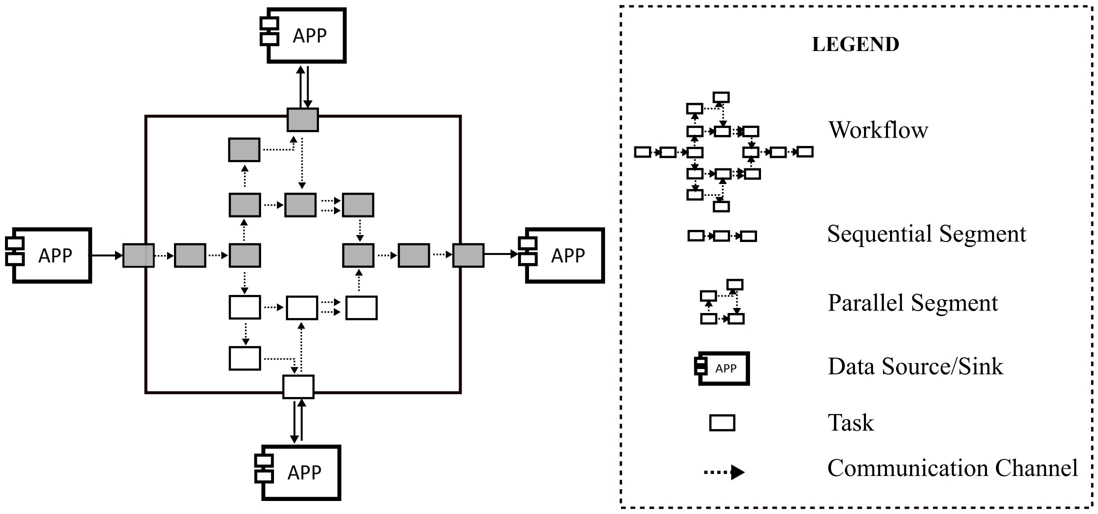


Figure 4.1: *Integration process example.*

We use the following terminology:

Task is a computational code that implements an atomic operation.

Communication channel is the means whereby messages pass from a task to another.

Workflow is a set of atomic chained tasks by communication channels inside an integration process.

Segment is a piece of a workflow that can be composed of tasks arranged sequentially, in parallel, or both.

Path refers to a specific set of tasks, by which a message is entirely processed in an integration process.

A task can have one or more inputs, and one or more outputs, depending on the implemented operation. This operation can be to transform, filter, split, join, or route messages. The tasks have an order of dependence in which they must be executed, such that a task can only process a message after every predecessor tasks have processed this message. After a message be processed by a task, it is written to the communication channel that connects this task with the next successor task in the path. There may be parts of the integration process that contain tasks that can be executed in parallel.

The accomplishment of a job corresponds to receiving of one or more messages from one or more source applications, the processing of these messages by tasks of a path of an integration process, up to sending of one or more messages to one or more deliver applications. Generally, several jobs are processed at a particular point in time; that is, several service requests are fulfilled at a particular point in time. The execution model of runtime systems establishes how they must execute tasks of an integration process and allocate threads during the processing of messages [65].

4.2 Task-based execution model

There are two main execution models for runtime systems: process-based and task-based [4, 23, 24, 60]. In the former model, a thread is assigned to an instance of the integration process, so that the thread is used to execute every task that composes the workflow over an inbound message to make this message flow throughout the process. After every task in the workflow has been executed, the thread is released. In the latter model, a thread is assigned to an instance of a task, so that this thread is used to execute the task over the inbound message that reaches the task. When the task finishes, an outbound message is written to the channel that connects the current task to the next task in the workflow and the thread is released. The execution of the message in the next task now depends on a new assignment of an available thread to this task. In this research, we address the task-based execution model.

A task is ready to be executed if there are messages in all communication channels that are sources to this task. Ready tasks depend on an available thread to execute them. Meanwhile, their executions are annotated in a waiting queue. So, the tasks are instantiated and executed following a FIFO policy, in which the task that was annotated first to enqueue is scheduled first to execute. Usually, threads are grouped in pools that avoid the creation of consecutive threads and allow quickly handle requests of tasks [97].

All activities required for the accomplishment of the message processing are orchestrated by the «scheduler» that is the central element of the runtime system. The scheduler also manages the computational resources responsible for the execution of the tasks. These resources are «threads» and are grouped in «pool of threads». A thread is the smallest sequence of a computational program that can be managed by the runtime system. The execution threads are abstractions of pieces of the physical threads, also called, CPU cores, which are physical and independent processing units. In this research, we refer to execution threads as threads and physical threads as «cores». The number of tasks that can be executed in parallel is limited to the number of available physical threads, and the executions always obey the order of dependence in an integration process.

The scheduler creates, manages, and releases threads and can configure the pool by determining parameters, such as initial thread number, the maximum number of thread, and the maximum lifetime of an idle thread. The scheduler assigns threads to execute instances of tasks, and after an instance of the task is executed, the thread is released back to the pool. The processing of a message in the next task now depends on a new assignment of an available thread from the pool to this task. A message is processed under the order of dependence on the tasks of the path, which is composed of several segments. Tasks in sequential segments are executed sequentially, whereas tasks in parallel segments can be executed in parallel because there is no dependency between them.

The operations required to create and start the threads are shown in Figure 4.2. Threads are started asynchronously by invoking operation `start`. The business logic that a thread executes by run operation, corresponds the execution of a task. The `run` operation implements a loop that enables the threads to poll the task queue as long as the scheduler is running. A thread polls an annotation of task execution and execute this respective task. So, available threads to keep working as long as there is a task ready to be executed.

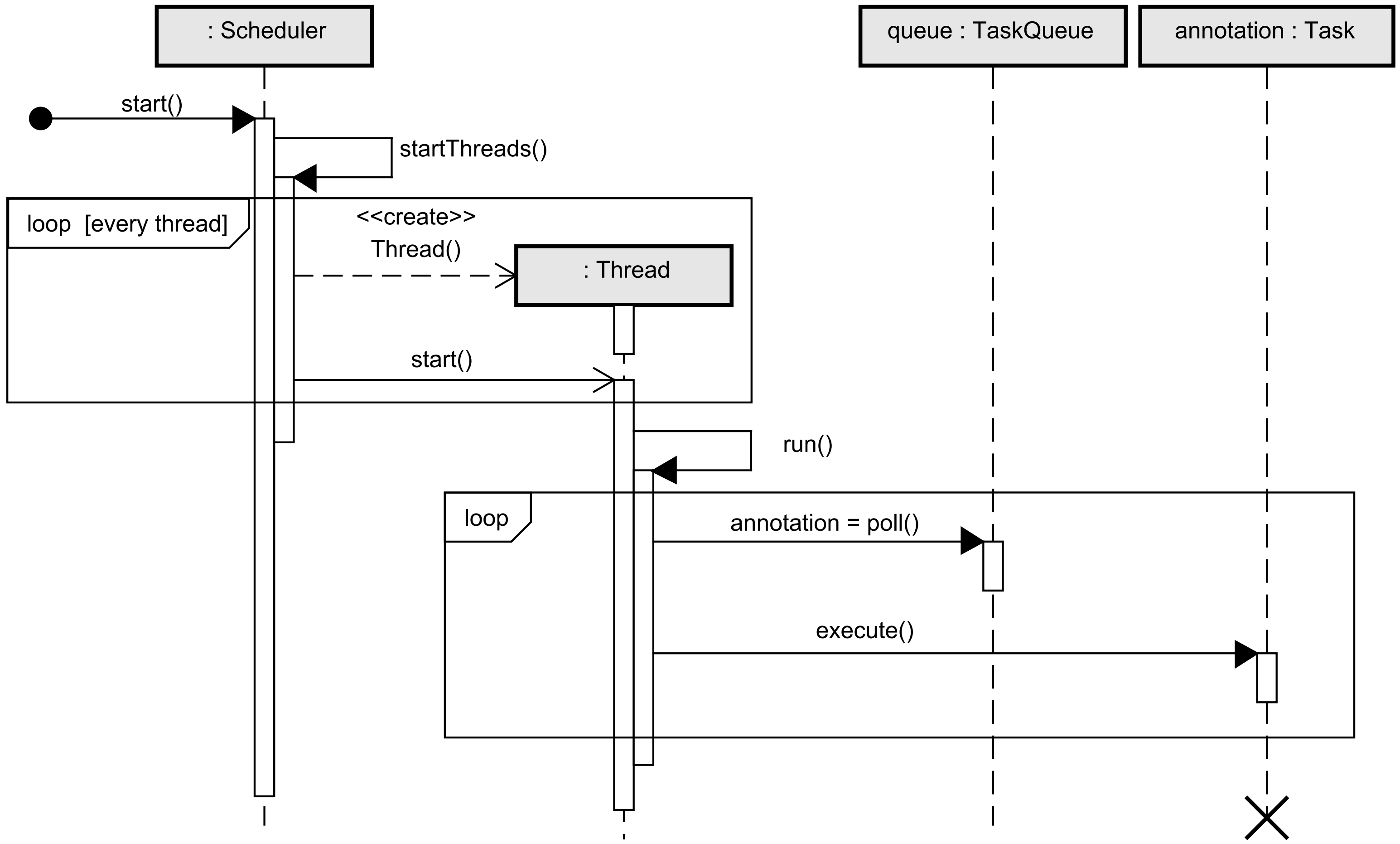


Figure 4.2: *Thread creation and initialisation (based on Frantz [63]).*

The execution of a task is depict in Figure 4.3. This execution requires invoking operation `execute` on the associated task, which first packages the input messages and then invokes operation `doOperation`, which corresponds the operation of the task, such as to transform, filter, split, join, or to route messages. Then, the annotation of task writes its output messages to the appropriate communication channel, which in turn notifies the tasks that read from them. These tasks then determine if they become ready for execution or not; in the former case, the tasks notify the scheduler. For every task notification that the scheduler receives, it creates a new annotation and appends it to the task queue.

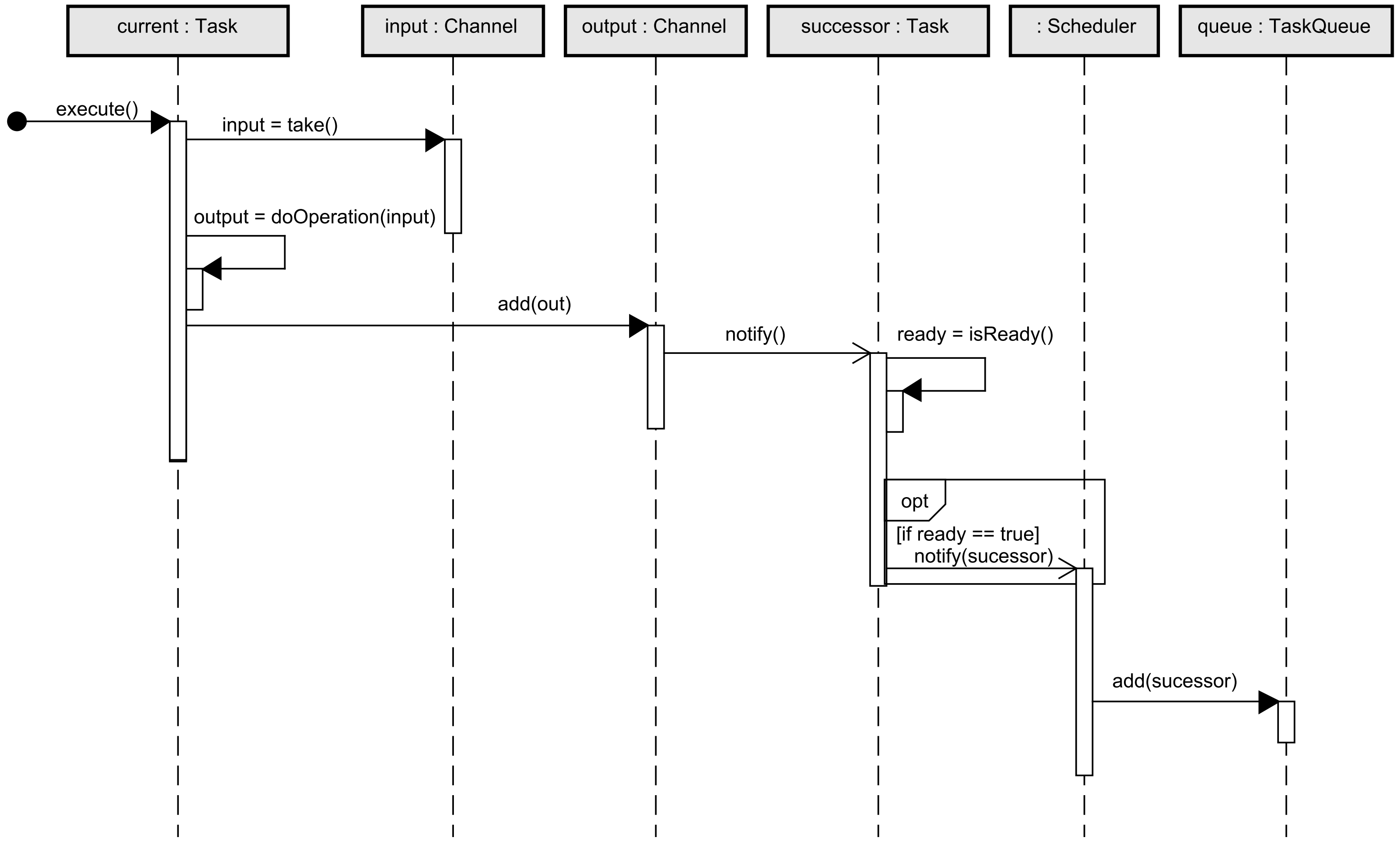


Figure 4.3: *Execution of tasks (based on Frantz [63]).*

4.3 Summary

In this chapter, we presented the main concepts of the research field of Enterprise Application Integration. We defined the key terms used in this research and described how is the task scheduling for integration processes of runtime system that use the task-based execution model.

Chapter 5
Task Scheduling

Courage is like — it's a habitus, a habit, a virtue: you get it by courageous acts. It's like you learn to swim by swimming. You learn courage by couraging.

Marie M. Daly, American biochemist (1921–2003)

Task scheduling is the proceeding of partitioning an integration process into tasks and assigning each task to a thread based on a policy or heuristic aiming to optimise the performance of processing of messages. It concerns the allocation of computation resources (threads) and the sequencing and timing for execution of the tasks that compose the integration processes. In this chapter, we define the concept of scheduling and outline the main types of scheduling problems, highlighting the dynamic scheduling, which is usually found in real-world problems.

5.1 Scheduling Definition

Scheduling is the decision-making process, in which it is necessary to answer «what», «when», and «where» to perform a job [140]. «What» refers to the decision about the set of tasks that must be carried out to complete a job. «When» refers to the decision about the set of time intervals associated with tasks that must be carried out. «Where» refers to the decision about the set of resources used to perform the tasks.

Scheduling covers two problems: resource allocation and task sequencing and timing. The former precedes the latter. Resource allocation addresses the decision about which resources are used for performing which tasks, and according to Pinedo [147] and Blazewicz et al. [22]; it can be described as follows:

> *If* $T = \{t_1, t_2, ..., t_n\}$ *is a set of tasks, where* $t \in T$*, and* $R = \{r_1, r_2, ..., r_m\}$ *is a set of computational resources that executes tasks, where* $r \in R$*, how the resources of* R *must be allocated to perform the tasks of* T *in order to maximise performance?*

Task sequencing and timing addresses the decision about how to distribute the execution of the tasks over time. Sequencing addresses the order in which tasks must be executed, whereas timing addresses the initial time that each task must be executed. According to Baker and Trietsch [14], task sequencing and timing can be described as follows:

> *If* $T = \{t_1, t_2, ..., t_n\}$ *is a set of tasks, where* $t \in T$*;* $S = \{s_1, s_2, ..., s_n\}$ *is a set of task segments, where* $s \in S$*; and* $[t_{ini}, t_{fin}]$ *is a time interval, how the tasks of* T *must be sequenced in* S *and distributed in* $[t_{ini}, t_{fin}]$ *in order to maximise performance?*

It is possible to find optimal solutions to resource allocation problems through mathematical models; however, the combinatorial nature of task sequencing and timing problems makes them complex, and, consequently, they are not efficiently solved through mathematical models [14, 22, 147].

5.2 Scheduling Classification

The main scheduling problem classification divides the problems into two categories: static and dynamic [189], cf. Figure 5.1. The scheduling is static when all the tasks and their characteristics are previously known, and none task is added until the conclusion of the entire job; otherwise, the scheduling is dynamic.

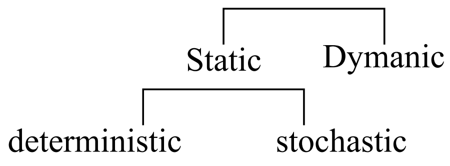


Figure 5.1: *Scheduling classification.*

A static scheduling problem can be deterministic or stochastic. When deterministic, it is assumed that the conditions of the problem are known and the parameters of the problem are described as constants. Usually, this approach is used to simplify real problems that always embed a certain degree of randomness. When stochastic, it is assumed that the conditions of the problem are not known.

In the static and deterministic approach, scheduling seeks to the initial times of a given set of tasks that must be executed on one or more resources. It involves a finite set of tasks with known characteristics and a finite processing time interval. The parameters of the problem are described as constants. In this case, there is one possible scenario to schedule at a particular point in time, only valid for this scenario. In real problems, the static and deterministic approach copes with short-term decisions. Such decisions based on a particular status of the environment, on its combinatorial structure, and realised uncertainty.

In the static and stochastic approach, scheduling seeks to the order in which a finite set of tasks with unknown characteristics should be executed. The parameters of the problem are described as approximations of static distributions. In this case, there are several possible scenarios, and the schedule must deal with the combinatorial and stochastic nature of the problem. The static and stochastic approach copes with short-term decisions based on a particular status of the environment, corresponding to a particular performance of the job arrival process at a specific point in time, considering real problems.

Dynamic scheduling deals with any job performance and jobs of different characteristics within a long time horizon. Then, dynamic scheduling is always stochastic. In this case, it seeks an optimal schedule for the current job performance, such that it is near to the schedule in which the uncertainties had been previously revealed. There is still the community scheduling that is a generalisation of the dynamic scheduling, which copes with a dynamic problem as a collection of related static problems; thus, methods developed for static scheduling problems can be applied to the dynamic ones. Several literature reviews discuss the limitations of the static scheduling approach in the presence of real-time information [195].

5.3 Summary

In this chapter, we defined resource allocation and task sequencing and timing. The former addresses the decision about which the resources perform tasks. The latter discusses the decision about how to distribute the execution of the tasks over time. After, we explained about static and dynamic scheduling. The static scheduling refers to the scheduling with know characteristics and dynamic scheduling refers to the scheduling with unknown characteristics.

Chapter 6
Heuristics

A scientist in his laboratory is not a mere technician; he is also a child confronting natural phenomena that impress him as though they were fairy tales.

Marie Curie, Polish physicist (1867-1934)

The literature classifies heuristic methods into constructive heuristics, local search, and meta-heuristics. In this chapter, we explain this classification and address seven popular scheduling heuristics: First-In-First-Out, Shortest Job First, Shortest Remaining Time First, Round Robin, and Priority Scheduling. Besides, we approach some meta-heuristic scheduling algorithms as Particle Swarm Optimisation, Genetic Algorithm, Simulated Annealing, and Ant Colony Optimisation, highlighting the Particle Swarm Optimisation meta-heuristic, which was used in our proposal.

6.1 Heuristic Methods

An optimisation problem may be solved by an exact method or an approximate method, depending on the complexity of the. Exact methods find optimal solutions and guarantee their optimality. For NP-complete problems, exact algorithms are non-polynomial time algorithms. Heuristics find high-quality solutions in a feasible time for practical use, but there is no guarantee of finding a global optimal solution [186]. Task scheduling is classified as a computationally complex problem, which cannot be solved by exact optimisation techniques in reasonable computational time. Usually, it is addressed by approximation methods that will achieve satisfactory performance in short processing times [14, 22]. We highlight three heuristic methods: «Constructive Heuristics», «Local Search», and «Meta-heuristics».

Constructive Heuristics are heuristic optimisation methods that build a solution of a problem, starting from an empty solution to which elements are being added. In this case, the solutions are constructed by a sequence of sub-problem solutions. They are called myopic techniques because in building a solution, they do not consider the impact of decisions at later stages [22, 186].

Local Search is heuristic optimisation methods that search part of the problem-solution space. In this case, they handle a problem solution to find new solutions with better performance. Usually, they start from random solutions, and then they move towards the best solutions from the neighbourhood. However, they run the risk of remaining in the local optimal space [22, 186].

Meta-heuristics are usually inspired by phenomena observed in nature. They are endowed with stochastic mechanisms that allow greater diversity in search of the solution space and surpass the optimal local. Therefore, they are one of the main optimisation methods for problems complex.

6.2 Heuristic Scheduling Algorithms

In this thesis, we address task scheduling that tackles with the problem of selecting a task instance from a waiting queue to be executed by the thread. Some traditional scheduling algorithms will be described follow as.

First-In-First-Out (FIFO), also known as First-Come-First-Serve (FCFS), is the simplest and the most common scheduling algorithm. In this algorithm, the task that requests the thread first is one that allocated the thread first. Then, this will be the task that waits in the queue for the longest time. The FIFO algorithm is non-preemptive, i.e. when the thread has been allocated to a task, that task keeps the thread up to either terminating or requesting external call (I/O). The weakness of the FIFO algorithm is that each task does not get to share the thread at the regular intervals due to its non-preemptive nature[180].

Shortest Job First (SJF), also known as Shortest Job Next (SJN) or Shortest Process Next (SPN), is a non-preemptive scheduling policy that selects the task with the shortest execution time from a waiting queue to be executed by an available thread[187]. SJF minimises the average amount of time each task has to wait until its execution is complete [121]. However, the disadvantage of SJF is that if short tasks are continually added and the tasks that require a long time to complete can become a long time to execute or do not executed, starvation state.

Shortest Remaining Time First (SRTF) is a preemptive algorithm in which the task with the minimum execution time remaining until its completion is selected to execute. A task is executed until its complete execution or until a new task with a shorter execution time arrived in queue. In this case, firstly short tasks are executed very quickly, and the comparison between the execution time of currently executing the task and the execution time of the new process, ignoring all other tasks. It also has the potential for task starvation [187].

Round Robin a (RR) scheduling is a method of time-sharing, in which each task his executed per a given time slice (time quantum). When a task is not finished by the end of the time slice, the execution of the task is interrupted, and the next task is executed [182]. If the time slice is too short, then too much task switching occurs, if the slice time is too long, then the system may become unresponsive. Many varied versions of the Round Robin algorithm are available which provide better results [19, 124, 164, 173, 209].

Priority Scheduling is a heuristic in which a priority is associated with each task, and an available thread is allotted to the highest priority task. Priority scheduling can be either pre-emptive or non-preemptive [118]. The former interrupted the execution if the priority of the newly arrived task is higher than the priority of the currently running task. The latter put the new task at the head of the waiting queue. Tasks of low priority can keep in a starvation state.

Highest Response Ratio Next (HRRN) scheduling [182] is a non-preemptive algorithm, in which the priority of each task is dependent on its estimated execution time and also the of waiting time.

Longest Job First (LJF) is a non-preemptive algorithm that selects the task with the largest execution time from the waiting queue to execute first. In this case, makespan is minimized, but the response time can increase [12, 201].

6.3 Meta-heuristic Scheduling Algorithms

Some of the main scheduling meta-heuristics Particle Swarm Optimisation, Genetic Algorithm, Simulated Annealing, and Ant Colony Optimisation.

Particle Swarm Optimisation (PSO) is a population based stochastic optimisation algorithm, in which a random swarm of particles is created initially, and particle position represent a solution for an optimisation problem. Some scheduling algorithms use the basic PSO algorithm, while others improve the PSO [194]. This meta-heuristic is described in detail in the next section because it composes our proposal.

Genetic Algorithm (GA) was developed by Holland [86] in the University of Michigan, USA to understand the adaptive processes of natural systems. GA was applied to optimisation and machine learning [74, 99] A GA usually applies a crossover operator to two solutions and a mutation operator that randomly modifies the individual contents to promote diversity. GAs use a probabilistic selection that is initially the proportional selection. The replacement is generational, in which the parents are replaced systematically by the off-springs [87].

Simulated Annealing (SA) applied to optimisation problems arises from the works of Kirkpatrick et al. [101] and Černỳ [32]. This meta-heuristic is popular in the field of heuristic search for its simplicity and efficiency in solving combinatorial optimisation problems as well as in dealing with continuous optimisation problems [44, 120]. SA bases on the principles of statistical mechanics, whereby the annealing process requires heating. Then slowly cooling a substance to obtain a stable crystalline structure. The SA algorithm simulates the exchange of energy in a system under a cooling process up to it converges to an equilibrium state [186].

Ant Colony Optimisation (ACO) was proposed by Dorigo [47] and bases on the cooperative behaviour of real ants to solve optimisation problems. Usually, ACO has been applied to combinatorial optimisation problems, and they are widely used to solve different issues, such as scheduling [48]. The decentralised and asynchronous nature of ants is favourable to solve distributed problems where there is no global view of the objective function and decisions are taken under a local view of the problem [186].

6.4 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is a population based on stochastic optimisation algorithm, in which a random swarm of particles is created initially, and particle positions represent possible solutions for an optimisation problem. Some scheduling algorithms use the basic PSO algorithm, while others improve the PSO [194]. PSO meta-heuristic was proposed by Eberhart and Kennedy [50] based on the behaviour of animal flocks, such as insects, fish and birds. Such animals locate right regions with food sources through iterative adjustment of their positions in search space, taking into account their best individual positions and the best general position of the group. The algorithm that implements the PSO is widely used because it presents better computational performance for high dimensional nonlinear optimisation problems with continuous variables and has fewer parameters to adjust than other algorithms, so, facilitating their implementation [5, 26]. The pseudo-code for the PSO algorithm is shown in Algorithm 6.1.

The PSO algorithm is a stochastic optimisation technique in which a particle represents an individual and is characterised by its position on search space and its velocity in a given time. The particle position represents a candidate solution to the optimisation problem, and the velocity is a function of the best position of the particle and of the general best position of any of the particles in a given time interval. An objective function is a function used to measure how good is the position of a particle and a position is considered the best position if it is the closest to the goal.

The algorithm iteratively tunes the particle position and velocity towards the record of best particle positions $\overrightarrow{bpos}$ and the general best position $\overrightarrow{gbpos}$. A random term weights how much the particle moves towards $\overrightarrow{bpos}$ and $\overrightarrow{gbpos}$, with different random numbers for acceleration towards each one of them. Equation 6.1 updates the particle position, where t and $t+1$ indicate two successive iterations of the algorithm. Equation 6.2 updates the particle velocity $\overrightarrow{vel_i}$ to the next iteration of the algorithm, where $\overrightarrow{vel_i}$ is the vector collecting the velocity-components of the ith particle along the D dimensions [126, 133]. The algorithm is interrupted when a stopping criterion is reached. Usually, the stopping criterion is a maximum iteration number. However, it also is possible to define as stopping criterion a near-optimal value considered to be good enough.

$$\overrightarrow{pos_i}(t+1) = \overrightarrow{pos_i}(t) + \overrightarrow{vel_i}(t+1) \tag{6.1}$$

$$\begin{aligned}\overrightarrow{vel_i}(t+1) = \omega \bullet \overrightarrow{vel_i}(t) + \phi_p \bullet r_p \bullet (\overrightarrow{bpos_i}(t) - \overrightarrow{pos_i}(t)) \\ +\phi_g \bullet r_g \bullet (\overrightarrow{gbpos}(t) - \overrightarrow{pos_i}(t))\end{aligned} \tag{6.2}$$

where:
ω = inertia weight
ϕ_p = acceleration coefficient of the best position of the particle i
ϕ_g = acceleration coefficient of the general best position
r_p and r_g = random numbers $\in [0, 1]$
$\overrightarrow{bpos_i}(t)$ = best position of the particle i
$\overrightarrow{gbpos}(t)$ = general best position of the population
$\overrightarrow{pos_i}(t)$ = current position of the particle i

The velocity vector $\overrightarrow{vel}$ is the sum vector of the cognitive term and social term with the particle inertia, cf. Equation 6.2. The cognitive term, which is based on individual experience, is given by $\phi_p \bullet r_p \bullet (\overrightarrow{bpos} - \overrightarrow{pos})$; whereas the social term, which is based on interaction amongst the particles is given

```
Input: stopping criterion
     $d \leftarrow n$
     $pos[i] \leftarrow pos_{random}$
     $vel[i] \leftarrow vel_{random}$
    while stopping criterion = false
      for i=1 to n

        if $pos_{current} \leftrightarrow bpos$

           $pos[i] \leftarrow pos_{current}$
           $bpos \leftarrow pos_{current}$
        enf if
        if $pos_{current} \leftrightarrow gbpos$
           $pos[i] \leftarrow pos_{current}$
           $gbpos \leftarrow pos_{current}$
        end if

         $pos[i] \leftarrow pos_{calculated}$

         $vel[i] \leftarrow vel_{calculated}$
      end for
    end while
```

Program 6.1: *PSO algorithm.*

by $\phi_g \bullet r_g \bullet (\overrightarrow{gbpos} - \overrightarrow{pos})$. The inertia weight ($\omega$) is the parameter that defines how much previous velocities affect the current velocity and defines a balance between the cognitive term and social term. If inertia is large, then the velocity increases, favouring the social term that corresponds to the global exploration. If inertia is small, then the velocity decreases favouring the cognitive term, i.e., the particles decelerate and hence favour local exploration. So, the proper value for inertia is one that balances global and local search diminishing the iteration number needed to the algorithm to converge [55]. Despite ϕ_p and ϕ_g do not directly impact the convergence of PSO, the proper tuning these parameters contributes for the convergence, as well as, helps to avoid that the algorithm stops in local minimal. In addition to the parameters present in Equation 6.2, the PSO algorithm receives as inputs: the number of particles, the dimension of particles, the range, and the maximum particle velocity. The number of particles depends on the complexity of the optimisation problem, but the standard is a value between 20 and 40 particles. The higher the number of particles, the greater the chance of finding the global optimum. dimension of particles and range define the space where the particles move and depend on the optimisation problem. The maximum velocity determines the maximum transition of a particle at each iteration, and usually, it is defined to the half of the position range of the particle [157, 170, 177].

6.5 Summary

In this chapter, we reported some heuristic and meta-heuristic scheduling algorithms. First-In-First-Out is the heuristic scheduling algorithm used in current integration platforms. The heuristic scheduling algorithms that were reported can be adapted to task scheduling of integration processes because it is necessary to obey the precedence order of the tasks. One adaptation of Round Robin was made to compare with the FIFO and with our proposal in our experiments. Amongst the meta-heuristic cited, we highlighted the Particle Swarm Optimisation, which was used in our proposal to the optimisation of an important parameter of our algorithm.

Chapter 7

Statistical Techniques

We have a hunger of the mind which asks for knowledge of all around us, and the more we gain, the more is our desire; the more we see, the more we are capable of seeing.

Maria Mitchell, American astronomer (1818–1889)

The used methodology for evaluating and analysing the performance of a runtime system should consider random sources to avoid distorted or even wrong results. Statistic analysis must be used to confirm findings of simulations of computational systems for performance evaluation, because these simulations cope with non-determinism factors present in such systems. In this chapter, we review statistical techniques that allow the validation of the findings of simulations of executions of integration processes by runtime systems related to performance evaluation.

7.1 ANOVA test

ANOVA test is a statistical technique that allows for differentiating amongst the measurements of an experiment, that are derived from random factors, and which are due to real differences between the alternatives being analysed. For the application of the analysis of variance, it is assumed that the observations are independent, the groups compared have the same variance and the errors are independent and come from a normal distribution with average zero and constant variance. The errors in an experiment can be of two types: thematic and random. Thematic errors are those caused by incorrect procedures during measurements. It is up to the researcher to control and eliminate such errors since they can invalidate the results, even in the cases in which the statistical analysis was performed. Random errors are unpredictable and non-deterministic and may come from external sources unrelated to what one wants to measure. Although it is not possible to predict random errors, it is possible to develop a statistical model to describe its effect on experimental results.

Therefore, the ANOVA test separates the total variation in two types: the first is the variation observed within each alternative studied, which is presumed to be a result of non-deterministic factors, and the second is the difference found between the alternatives considered. If the variation between the alternatives is greater than the variation within each alternative, it is concluded that there is a statistically significant difference between the alternatives [114].

The two graphs, in Figure 7.1, show the distribution of the averages of any three groups, called: G1, G2, G3. Graph (a) shows the average of distributions, in which there is no statistically significant difference between the three groups and the graph (b) shows the average of distributions, in which there is a difference between the groups.

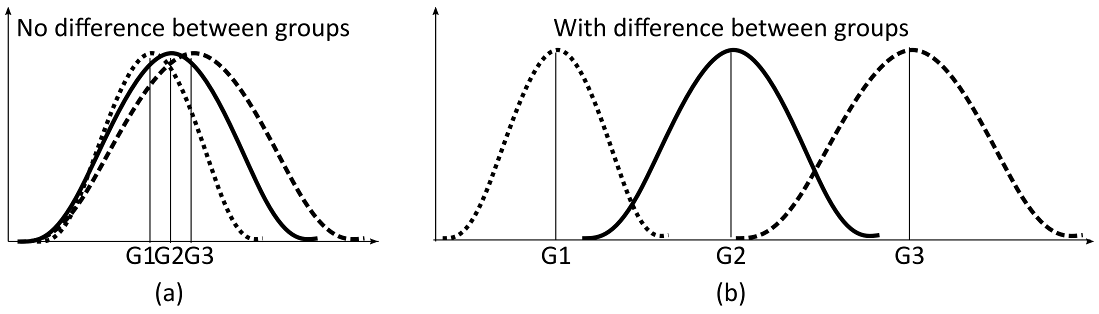


Figure 7.1: *Analysis of variance.*

7.2 Scott & Knott test

Scott & Knott is a test adopted in experiments for performance evaluation since it is a simple approach [70]. Multiple comparison techniques are used to distinguish of similar and of different treatments when there are several treatments in an experiment. There are numerous procedures of multiple comparisons in the literature. An efficient alternative is the Scott & Knott test [167], mainly when a large number of treatments is evaluated. The Scott

& Knott is considered a more rigorous test because it takes account significant differences between treatments, whereas tests least rigorous take account little differences between treatments.

The original procedure of Scott & Knott test starts by partitioning the groups to maximise the sum of squares between groups. Let «K» be the number of treatments. Firstly, the procedure of Scott & Knott test will either find two distinct groups dividing the treatment averages or will declare those «K» treatment average as homogeneous, belonging to just one group. Thus, $(2^{(K-1)} - 1)$ possible partitions of the «K» averages are verified into two non-empty groups. The process is facilitated when the averages are ordered since it is enough to verify at the $(K - 1)$ partitions formed by ordering the treatment averages and dividing them between two successive ones. Let T_1 and T_2 be the totals of two of those groups with K_1 and K_2 treatments respectively, so that $K_1 + K_2 = K$. The sum of squares is defined as «B», according to Equation 7.1. The maximum value of the sum of squares between groups is defined as «B_o», taken over the $(K - 1)$ partitions of the «K» treatments into two groups.

$$B = \frac{T_1^2}{K_1} + \frac{T_2^2}{K_2} - \frac{(T_1 + T_2)^2}{K_1 + K_2} \tag{7.1}$$

Then, the procedure of Scott & Knott test performs likelihood ratio test for the null hypothesis of equality of all averages against the alternative, in which the averages belong to the two groups aforementioned. If the null hypothesis is rejected, then the two groups are kept. Otherwise, the group of «K» treatment averages is considered homogeneous. This procedure is repeated for each group separated until all groups formed are homogeneous [96]. The statistics used for the likelihood ratio test is express by Equation 7.2. In this equation, $\hat{\sigma}_0^2$ is the maximum likelihood estimator of $\frac{\hat{\sigma}_0^2}{rp}$, where «rp» is number of repetitions of the experiment. λ is asymptotically a χ^2 distributed random variable with $\nu_0 = \frac{k}{\pi - 2}$ degrees of freedom and is used as the cutoff point for a given α value at each test. α is a Scott & Knott parameter for the significance level. Thus, if $\lambda < \chi^2(\alpha, \nu_0)$, all averages are considered homogeneous and, further partitioning is, therefore, unnecessary; otherwise, the two groups are considered statistically different and then should be tested separately for new possible divisions.

$$\lambda = \frac{\pi}{2 \bullet (\pi - 2)} \bullet \frac{B_o}{\hat{\sigma}_0^2} \tag{7.2}$$

7.3 Regression analysis

Regression analysis is a method to estimate the relation between the dependent and the independent variables. Let $x = (x_1, x_2, ..., x_n)$ be a vector of the independent variables and y a dependent variable. The mathematics function, which relates y and x, can be expressed by the regression model, c.f. Equation 7.3, where β is a vector of unknown parameters, and ε is a disturbance term [203].

$$y = f(x \mid \beta) + \varepsilon \tag{7.3}$$

In regression analysis, the square of Pearson product-moment correlation coefficient is an important parameter to determine the degree of linear correlation of variables. This coefficient is known as the correlation coefficient or, simply, R^2. R^2 is defined by Equation 7.4, where SSE is the sum of squared error and SST is the sum of squared total [100]. Thus, R^2 tends to 1 when $SSE \ll SST$, i.e., the sum of squared error is too small compared to the sum of the squared total.

$$R^2 = 1 - \frac{SSE}{SST} \tag{7.4}$$

The $t\!-\!statistic$ is a test used for making inferences about the regression coefficients. The hypothesis test on coefficient x_i checks the null hypothesis is equal to zero against the alternative hypothesis regarding the hypothesis of the coefficient to be different from zero. The null hypothesis that equal to zero means that the corresponding term to the coefficient x_i is not significant. The probability value $p\!-\!Value$ is another common metric used to determine the significance of the model results when applying hypothesis testing. According to McCarthy et al. [128] the result is considered statistically significant when $p\!-\!Value$ is small, i.e., $p\!-\!Value \leq 0.05$.

Stepwise regression is an iterative method for adding and removing terms from a multi-linear model based on their statistical significance in a regression. The method starts with an initial model and then compares the explanatory power of incrementally larger and smaller models. At each iteration, the $p\!-\!Value$ of a $F\!-\!statistic$ is calculated to test models with and without a potential term. Similarly to $t\!-\!statistic$, $F\!-\!statistic$ tests if the averages between the two populations are significantly different. The $t\!-\!test$

checks if a single variable is statistically significant whereas $\mathsf{F-test}$ checks if a group of variables are jointly significant. The $\mathsf{F}-value$ in regression is the result of a test in which the null hypothesis is that all of the regression coefficients are equal to zero. The method finishes when no single step improves the model.

7.4 Summary

In this chapter, we described three statistical techniques: ANOVA, Scott & Knott, and Regression analysis. ANOVA test allows differentiating amongst the results of an experiment, what was due to random factors and what was due to treatment applied. Scott & Knott test identifies which results are similar and which are different. Regression analysis is a method to mathematically model the dependent variables as a function of the independent variables.

Part III

Our Approach

Chapter 8

Representation

I believe that fortitude is key. More than anything, be consistent.
Go at it. Go at it. Go at it. When you succeed, don't forget the responsibility
of making someone else succeed with you.

Antonia Novello, American physician

Our hypothesis in this chapter is that it is possible to obtain information about conceptual models and of the logic of integration processes, which help to categorise them. We have developed a model for representation of integration processes as a directed acyclic graph, called «integration operation typed graph» or IOTG, which focus on the functions and in the logic operations of the tasks of integration processes. We describe the environment of four integration processes and represent their conceptual models by IOTG. The case studies are based on research work of Frantz [63].

8.1 Integration operation typed graph

Information can be extracted from the logic semantics of the task; cf. we found in works of Belusso et al. [20], Kraisig et al. [107], Roos-Frantz et al. [158]. We also consider these information for the building of our representation. The conceptual models of integration processes describe in this chapter were designed in a single integration platform to standardise the implementation of the integration patterns. Initially, the models were created using the Guaraná domain-specific language [62]. We describe the terminology and then formulate a representation for task scheduling of integration processes.

The terminology of the task scheduling problem in integration processes is based on the classic real-time scheduling theory and general-purpose parallel systems.

An integration process can be represented as a job, J that consists of the entirely inbound message processing. For the entirely processing to take place, the inbound message has to be processed by every task of a path of an integration process. The task scheduling of integration processes can be represented as a set of jobs, $J = \{j_1, j_2, \cdots, j_n\}$ on computational resources of the same capability, consisting of «m» threads. A path has segments of tasks, which can be sequential, parallel, or both. The tasks in sequential segments are executed, obeying their order of dependence. The tasks of parallel segments can be simultaneously executed on different cores.

The Directed Acyclic Graph (DAG) represents task models for real-time scheduling, allowing the description of constraints on tasks execution [165]. In the DAG model, an integration process is described as a workflow «*W*» composed of «k» tasks, being an extension of the DAGs with weighted vertices (E_i, T_i), where $T_i = \{t_{i,1}, t_{i,2}, \cdots, t_{i,k}\}$ is the set of vertices and «E» is the set of edges. Every vertex in the graph represents a task of the process, and each edge represents a communication channel between tasks, as well as indicates precedence constraints between tasks. An edge between (t_i, t_j) represents a dependence between t_i and t_j, in which t_i is the predecessor node of t_j and t_j is the successor node of t_i. Every edge has a weight, which represents the waiting time of the task in the queue.

A starting task is a task that has no predecessor tasks in the order of dependence. An ending task is a task that has no successor tasks. An integration process can have one or more starting tasks and one or more ending task. Besides, an integration process can have tasks that exchange messages with applications during runtime and intermediate tasks that fork, join or route messages to segments that compose the path whereby the message must flow. There are several possible paths for the processing of a message, which are defined at design time. However, the path through which a message will flow depends on the integration process logic.

Ritter et al. [154] represented the integration process as a directed graph, called Integration Pattern Typed Graph (IPTG). IPTG was defined as a set of nodes T and a set of edges $E \subseteq T \times T$ add to the function *type* $: T \to F$, where $F = \{start, end, message\ processor, fork, join, condition, merge, external\ call\}$. For a node $t \in T$, $\bullet\, t = \{t' \in T | (t' \bullet t) \in E\}$ for the set of direct predecessors of t, and $t\, \bullet = \{t'' \in T | (t \bullet t'') \in E\}$ for the set of direct successors of t.

The function *type* records what type of task each node represents. An IPTG model is correct if it observes three correctness conditions: the first claims that an integration pattern has at least one input and one output; the second condition indicates the cardinality of the involved tasks, i.e., the in-degrees and out-degrees of a node; and the last condition states, «the graph (T, E) is connected and acyclic», indicates that a graph represents only a task and its relation with its predecessor and successor tasks and that messages do not loop back to previous tasks. IPTG representation adopt as the condition of verification, the classification by task cardinality, and some terminologies, such as type «start», «end», «join», «message processor», and «external call». However, because we have taken into consideration the logic operation of the task, then we add «and», «or», and «xor» function *types*. If the logic operation equals «and», the task generates outbound messages to every one of its outputs. If the logic operation equals «xor», the task generates outbound messages to only one of its outputs. The logic operation «or» acts as both «and» and «xor», i.e., the task can generate an outbound message to one or more outputs. We named our representation as integration operation typed graph. The function *types* of tasks are summarised in Table 8.1.

Type	**Description**
«start»	Task that has no predecessor task.
«end»	Task that has no successor task.
«join»	Tasks that has more of one predecessor task.
«message processor»	Tasks that has one predecessor task and one successor task.
«external call»	Task that communicates with external application.
«and»	Tasks that has one predecessor task and more of one successor task. This task generates outbound messages to every one of its outputs.
«xor»	Tasks that has one predecessor task and more of one successor task. This task generates outbound messages to only one of its outputs.
«or»	Tasks that has one predecessor task and more of one successor task. This task can generate an outbound message to one or more outputs.

Table 8.1: *Function* types *of tasks.*

An IOTG (T, E, *type*) is *correct* if it satisfies the following conditions:

- $\exists\, t_1, t_2 \in T$ with *type* $(t_1) = \mathtt{start}$ and *type* $(t_2) = \mathtt{end}$;
- if *type* $(t) \in \{\mathtt{and}\}$ then $|\bullet\, t| = 1$ and $|t\, \bullet| = n$ and must produce messages to all n outputs;
- if *type* $(t) \in \{\mathtt{or}\}$ then $|\bullet\, t| = 1$ and $|t\, \bullet| = n$ and produce message in at least one of its outputs;
- if *type* $(t) \in \{\mathtt{xor}\}$ then $|\bullet\, t| = 1$ and $|t\, \bullet| = n$ and produce message in by only one of its outputs;
- if *type* $(t) \in \{\mathtt{join}\}$ then $|\bullet\, t| = n$ and $|t\, \bullet| = 1$;
- if *type* $(t) \in \{\mathtt{message\ processor}\}$ then $|\bullet\, t| = 1$ and $|t\, \bullet| = 1$;
- if *type* $(t) \in \{\mathtt{external\ call}\}$ then $|\bullet\, t| = 1$ and $|t\, \bullet| = 1$;
- the graph (T, E) is connected and acyclic.

Integration processes that have more than one task of the type «start» benefit from the adoption of the task-based execution model because it is not necessary to have messages in all the inputs of the process to begin its execution. Similarly, processes, which have several tasks of the type «external call» type, also benefit from the task-based model because while a thread can be blocked waiting for a response of an external application, the other threads can execute the tasks of the integration process. Integration processes that have more than one task of the type «and» would benefit from the use of parallel processing. Integration processes that have several tasks of the type «join» may have a delay, mainly when such tasks are involved in the correlation of messages. Integration processes with several communication channels demand more memory to store messages. Usually, the message arrival rate is random and can be batch, periodic, or real-time. The size and format of a message can also vary, from bytes to petabytes, and data can be structured or unstructured. The value of these parameters determine the execution time of tasks.

8.2 Coffee shop integration process

The case study 1 (`CS1`) is the Coffee Shop problem, a benchmark of an integration process introduced by Hohpe [84]. Its conceptual model is depicted in Figure 8.1. In the `CS1`, the integrated applications are: «Orders», «Barista Cold Drinks», «Barista Hot Drinks», and «Waiter». «Orders» represents the source application that delivers the data of the customer orders to the integration process. The data of the customer orders are wrapped inside messages. An order may include either hot, cold drinks or both. Different baristas prepare cold and hot drinks, which represent two applications that exchange messages with the integration process: «Barista Cold Drinks» and «Barista Hot Drinks». The orders are delivered to the «Waiter» when all drinks corresponding to the same order have been prepared. The «Waiter» application represents a final data sink. The processing of one customer order corresponds to one job instance. It is possible to process one or more orders. One or more instances of the job can be being processed simultaneously. The number of instances of the job corresponds to the number of customer orders that are being processed. If there are several instances of the job at a given time, then there are several instances of the same task and each task instance is associated with a job instance.

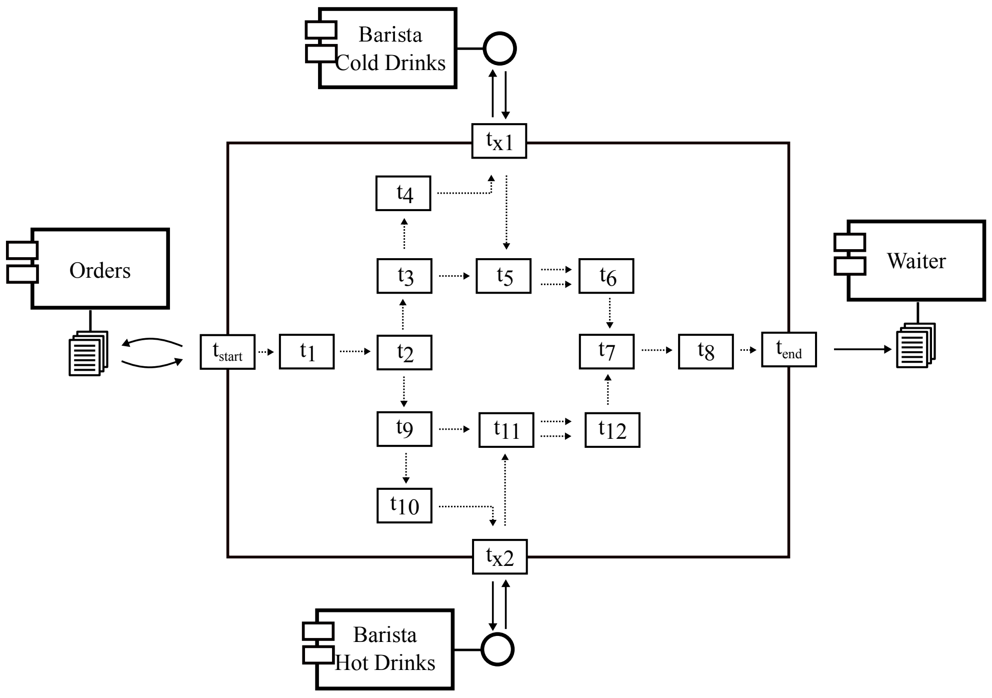


Figure 8.1: *Conceptual model of case study 1.*

The `CS1` is an integration process with three different paths for messages, in which there are tasks of types: «start», «and», «or», «join», «message processor», «external call» and «end». There is one input task represented by t_{start}, and one output task represented by t_{end}. Tasks that exchange messages with applications during runtime are represented by t_{x1} and t_{x2}. Intermediary tasks are represented by t_i, where i ranges from 1 to 12.

In the integration logic of this conceptual model, there may be customer orders containing only cold drinks, hot drinks, or orders containing both cold and hot drinks. For each one of these types of customer orders, there is a path through which messages are processed. The possible paths were defined during design time by the CS1 model. However, it is during runtime that the exact path for a given message is known, according to the type of customer order. There is a path for customer order containing only cold drinks; another path for only hot drinks; and another path for both cold and hot drinks. Examples of tasks that can be executed in parallel in the Coffee Shop integration process are $[t_3, t_9]$, $[t_4, t_{10}]$, $[t_{x1}, t_{x2}]$, $[t_5, t_{11}]$, $[t_6, t_{12}]$. The Coffee Shop integration process is represented by a DAG in Figure 8.2.

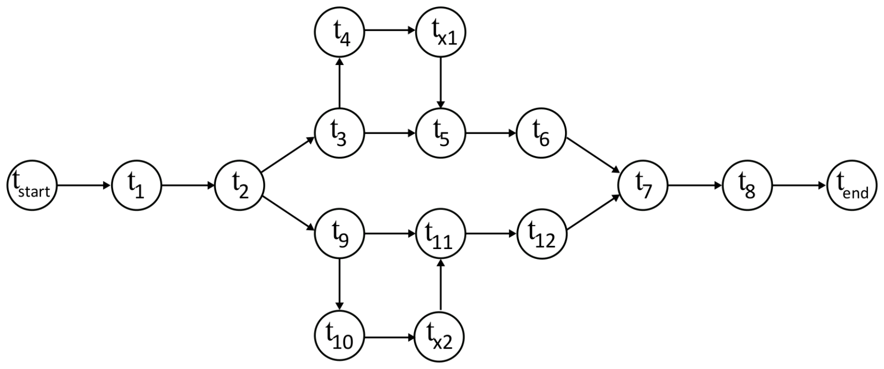


Figure 8.2: *Integration operation typed graph for case study 1.*

In CS1 integration operation typed graph, there are 16 nodes, which represent the tasks, and there are 19 edges, which represent the channels. Node t_{start} is a starting node, that represents a task of the type «start». The nodes t_3 and t_9 represent tasks of the type «and». The t_2 represents a task of the type «or». The nodes t_5, t_7, and t_{11} represent tasks of the type join; Nodes t_1, t_4, t_6, t_8, t_{10}, and t_{12} represent tasks of the type «message processor»; The nodes t_{x1} and t_{x2} represent tasks that send and receive information to/from applications that are tasks of the type external call; and, node t_{end} is a ending node that represent a task of the type «end». The possible paths of the conceptual model CS1 and their segments of tasks are shown in Table 8.2.

Path	**Segment**	
	Sequential	Parallel
$\{t_{start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_7, t_8, t_{end}\}$	$\{t_{start}, t_1, t_2\}$; $\{t_6, t_7, t_8, t_{end}\}$	$\{t_3, t_4, t_{x1}, t_5\}$
$\{t_{start}, t_1, t_2, t_9, t_{10}, t_{x2}, t_{11}, t_{12}, t_7, t_8, t_{end}\}$	$\{t_{start}, t_1, t_2\}$; $\{t_{12}, t_7, t_8, t_{end}\}$	$\{t_9, t_{10}, t_{x2}, t_{11}\}$
$\{t_{start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{x2}, t_{11}, t_{12}, t_{end}\}$	$\{t_{start}, t_1\}$; $\{t_8, t_{end}\}$	$p1 = \{t_3, t_4, t_{x1}, t_5\}$; $p2=\{t_9, t_{10}, t_{x2}, t_{11}\}$; $\{t_2, p1 , t_6, t_7, t_{12}, p2\}$

Table 8.2: *Representation of case study 1 by integration logic.*

8.3 Processing order integration process

The case study 2 (CS2) is the Processing Order problem, another benchmark of an integration process introduced by Hohpe [84]. Its conceptual model is depicted in Figure 8.3. In the CS2, the integrated applications are: «Ordering System», «Widget Inventory», «Gadget Inventory», «Invalid Items Log», and «Inventory System». «Ordering System» represents the source application that delivers the data of the new orders to the integration process. The data of the orders are wrapped inside messages. Every message with a new order is split into individual messages, each of which must contain only one item. A message is routed to «Widget Inventory» or «Gadget Inventory» depending on their contents. Messages with items that do not belong to any of these inventories are routed to «Invalid Items Log». The «Inventory System» application represents a final data sink that responds regarding the availability of items. The processing of one order corresponds to one job instance.

CS2 is an integration process with three different paths for messages, in which there are tasks of types: «start», «and», «xor», «join», «message processor», «external call» and «end». The CS2 has a different integration logic differs from CS1, besides this, in CS2 there are two ending tasks and a task of type «xor». There is one input task represented by t_{start}, and two output tasks represented by t_{1end} and t_{2end}. The tasks that exchange messages with applications during runtime are represented by t_{x1} and t_{x2}. The intermediary tasks are represented by t_i, where i ranges from 1 to 12.

In the integration logic of this conceptual model, an order contains several items. An order is split into unitary items, which can belong exclusively to

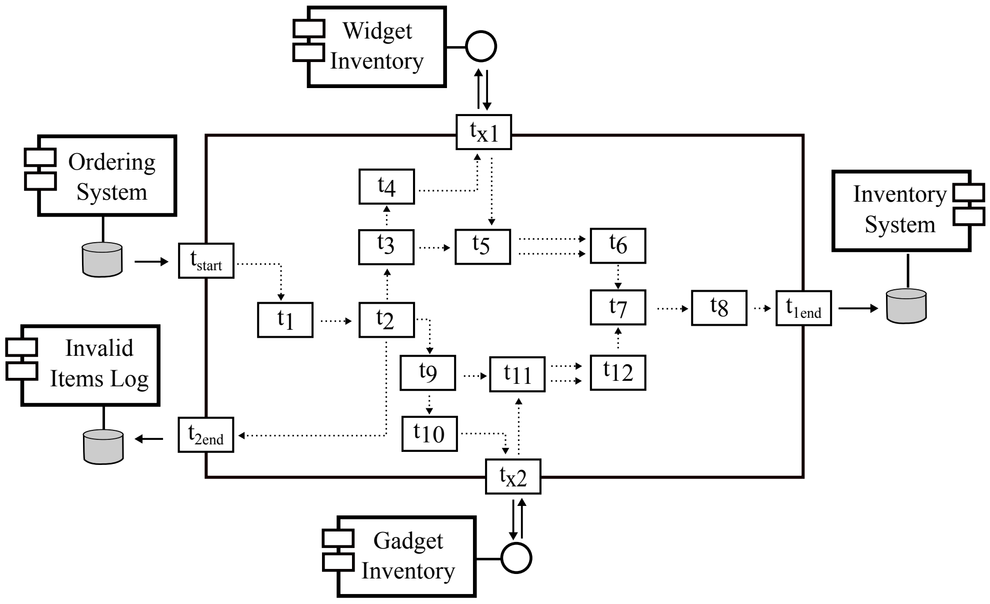


Figure 8.3: *Conceptual model of case study 2.*

one of the inventories, «Widget Inventory» and «Gadget Inventory», or to none. There is a path for a unitary item that belongs to «Widget Inventory»; another path for a unitary item that belongs to «Gadget Inventory»; and another path for a unitary item that does not belong to any inventory. Examples of tasks that can be executed in parallel in the CS2 integration process are $[t_3, t_9]$, $[t_4, t_{10}]$, $[t_{x1}, t_{x2}]$, $[t_5, t_{11}]$, $[t_6, t_{12}]$. The CS2 integration process is represented by a DAG in Figure 8.4.

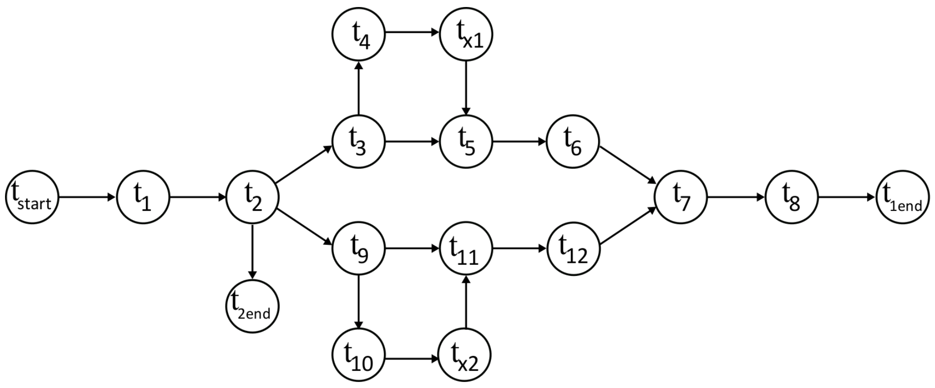


Figure 8.4: *Integration operation typed graph for case study 2.*

In CS2 integration operation typed graph, there are 17 nodes, which represent the tasks, and there are 19 edges, which represent the channels. Node

t_{start} is a starting node, that represents a task of the type «$\mathtt{start}$». The nodes t_3 and t_9 represent tasks of the type «$\mathtt{and}$». The t_2 represents a task of the type «$\mathtt{xor}$». The nodes t_5, t_7, and t_{11} represent tasks of the type «$\mathtt{join}$»; Nodes t_1, t_4, t_6, t_8, t_{10}, and t_{12} represent tasks of the type «$\mathtt{message\ processor}$»; The nodes t_{x1} and t_{x2} represent tasks that send and receive information to/from applications that are tasks of the type $\mathtt{external\ call}$; and, nodes t_{1end} and t_{2end} are ending nodes that represent tasks of the type «$\mathtt{end}$». The possible paths of the conceptual model `CS2` and their segments of tasks are shown in Table 8.3.

Path	**Segment**	
	Sequential	Parallel
$\{t_{start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_7, t_8, t_{1end}\}$	$\{t_{start}, t_1, t_2\}$; $\{t_6, t_7, t_8, t_{1end}\}$	$\{t_3, t_4, t_{x1}, t_5\}$
$\{t_{start}, t_1, t_2, t_9, t_{10}, t_{x2}, t_{11}, t_{12}, t_7, t_8, t_{1end}\}$	$\{t_{start}, t_1, t_2\}$; $\{t_{12}, t_7, t_8, t_{1end}\}$	$\{t_9, t_{10}, t_{x2}, t_{11}\}$
$\{t_{start}, t_1, t_2, t_{2end}\}$	$\{t_{start}, t_1, t-2, t_{2end}\}$	

Table 8.3: *Representation of case study 2 by integration logic.*

8.4 Huelva's county council integration process

The case study 3 (CS3) is the Huelva's County Council problem, a real-world integration process [61] that consists of the automatisation of the user registration into a central repository. Its conceptual model is depicted in Figure 8.5. In the CS3, the integrated applications are: «Local Users», «Portal Users», «LDAP», «Human Resources System», «Digital Certificate Platform», and «Mail Server». The «Local Users» represents one of the source applications that manage the data of the users of county council information systems. The «Portal Users» represents the web portal used to manage the users and is another source application. The «LDAP» is a unique repository for users that provides authentication access control for several other applications inside the software ecosystem. The «Human Resources System» represents the application that provides personal information about the employees and from it, information like name and e-mail are required to compose notification e-mails. The «Digital Certificate Platform» represents the application that manages digital certificates. Finally, the «Mail Server» represents the application that runs the e-mail service and is used exclusively for notification purposes.

CS3 is an integration process with four different paths for messages, in which there are tasks of types: «start», «and», «xor», «join», «message processor», «external call» and «end». The CS3 has a different integration logic from CS1 and from CS2, besides this, in CS3 there are two starting tasks and four possible paths. There are two input tasks represented by t_{1start} and t_{2start}, and two output tasks represented by t_{1end} and and t_{2end}. The tasks that exchange messages with applications during runtime are represented by t_{x1} and t_{x2}. The intermediary tasks are represented by t_i, where i ranges from 1 to 13.

In the integration logic of the CS3, the data of users that arrive from t_{1start} and t_{2start} are replicated and one copy flows towards «Human Resources System» for information about the employee who owns a user record. Further, on t_6, the message is replicated, and one copy flows towards «LDAP», and another flows towards «Digital Certificate Platform». «Digital Certificate Platform» is queried by a message that includes an e-mail address. The sending of the certificate and its notification to the employee about the inclusion in the «LDA» is done by «Mail Server». There is a path for a local user that has e-mail address; another path for a local user that has not an e-mail address; another path for a web user that has an e-mail address;

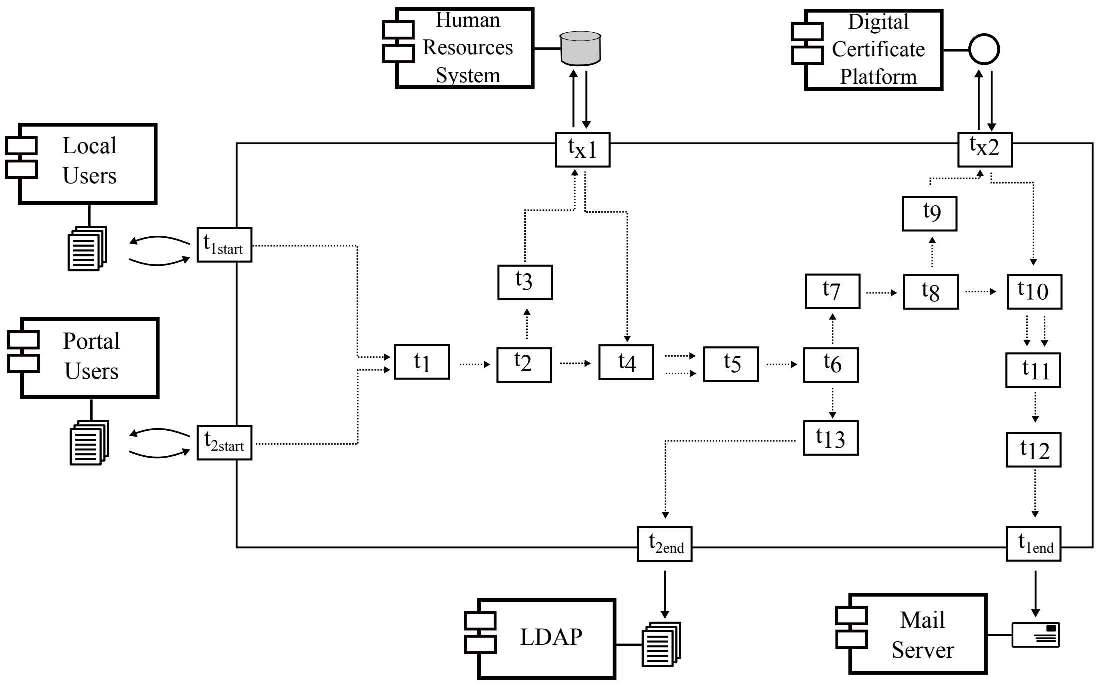


Figure 8.5: *Conceptual model of case study 3.*

and another path for a web user that has not an e-mail address. Examples of tasks that can be executed in parallel in the CS3 integration process are $[t_3, t_7]$, $[t_9, t_{10}]$, $[t_7, t_{18}]$. The CS3 is represented by a DAG in Figure 8.6.

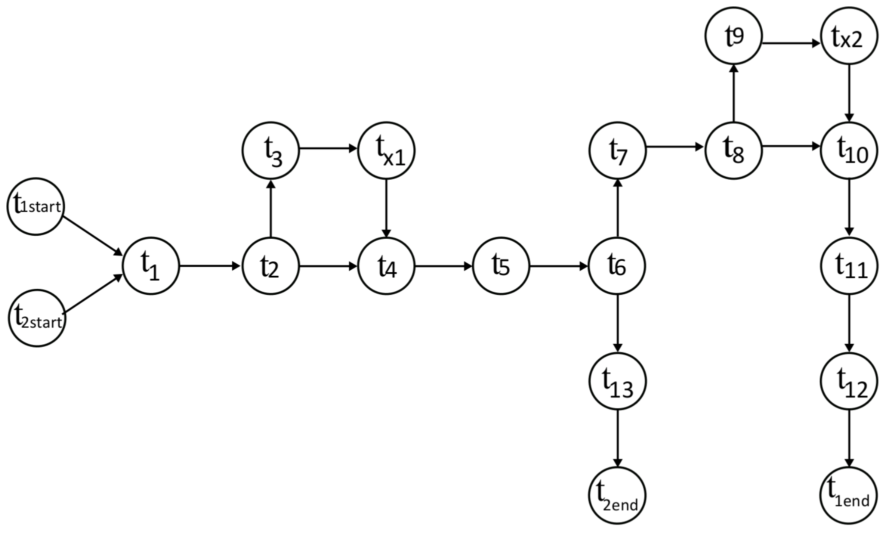


Figure 8.6: *Integration operation typed graph for case study 3.*

In CS3 integration operation typed graph, there are 19 nodes, which represent the tasks, and there are 20 edges, which represent the channels. Nodes t_{1start} and t_{2start} are starting nodes, that represent tasks of the type «start». The nodes t_2 and t_8 represent tasks of the type «and». The t_6 represents a task of the type «xor». The nodes t_1, t_4, and t_{10} represent tasks of the type join; Nodes t_3, t_5, t_7, t_9, t_{11}, t_{12}, and t_{13} represent tasks of the type «message processor»; The nodes t_{x1} and t_{x2} represent tasks that send and receive information to/from applications that are tasks of the type «external call»; and, nodes t_{1end} and t_{2end} are ending nodes that represent tasks of the type «end». The possible paths of the conceptual model CS3 and their segments of tasks are shown in Table 8.4.

Path	**Segment**	
	Sequential	Parallel
$\{t_{1start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{1end}\}$	$\{t_{1start}, t_1\}$; $\{t_5, t_6, t_7\}$; $\{t_{11}, t_{12}, t_{1end}\}$	$\{t_2, t_3, t_{x1}, t_4\}$; $\{t_8, t_9, t_{x2}, t_{10}\}$
$\{t_{1start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_{13}, t_{2end}\}$	$\{t_{1start}, t_1\}$; $\{t_5, t_6, t_{13}, t_{2end}\}$	$\{t_2, t_3, t_{x1}, t_4\}$
$\{t_{2start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{1end}\}$	$\{t_{2start}, t_1$; $t_5, t_6, t_7\}$; $\{t_{11}, t_{12}, t_{1end}\}$	$\{t_2, t_3, t_{x1}, t_4\}$; $\{t_8, t_9, t_{x2}, t_{10}\}$
$\{t_{2start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_{13}, t_{2end}\}$	$\{t_{2start}, t_1\}$; $\{t_5, t_6, t_{13}, t_{2end}\}$	$\{t_2, t_3, t_{x1}, t_4\}$

Table 8.4: *Representation of case study 3 by integration logic.*

8.5 Phone call integration process

The case study 4 (CS4) is the Unijuí University problem, a real-world integration process [61] that consists of the automatisation of the charge of personal phone calls using phones of the university. Its conceptual model is depicted in Figure 8.7. In the CS4, the integrated applications are: «Call Centre», «Human Resources», «Payroll System», «Mail Server», and «SMS Notifier». The «Call Centre» records every call every employee makes from a university belonged phone. The code is also used to correlate phone calls with the information in the «Human Resources» and the «Payroll System». The «Human Resources» supplies personal data concerning employees, and the «Payroll System» computes their wages. The «Mail Server» and the «SMS Notifier» notify employees about their charges. The former provides e-mail service and the later offers short message system services.

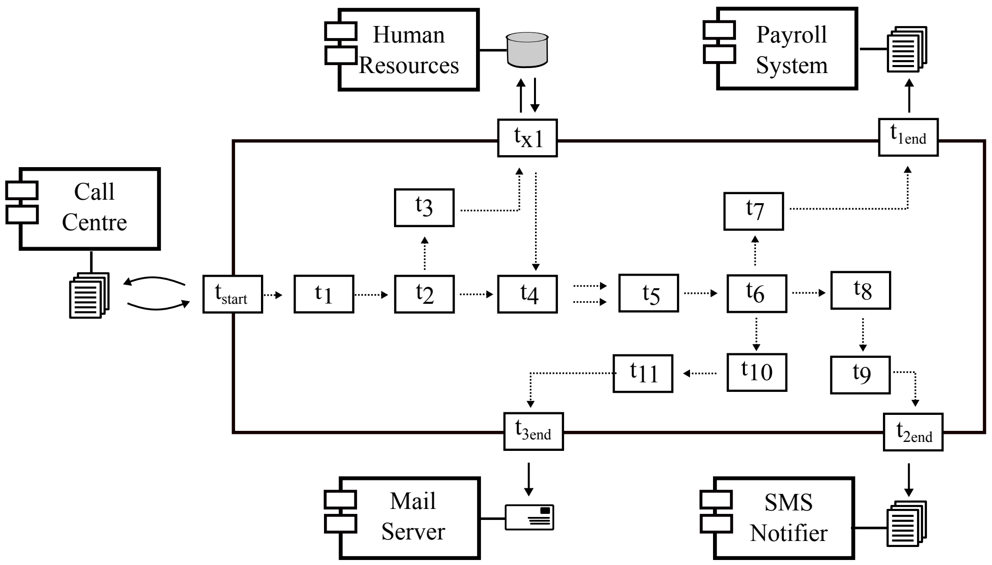


Figure 8.7: *Conceptual model of case study 4.*

CS4 is an integration process with single path for messages, in which there are tasks of types: «start», «and», «join», «message processor», «external call» and «end». The CS4 has a different integration logic from the previous cases, besides this, in CS4, there are three ending tasks and a single possible path. There is an input task represented by t_{start}, and three output tasks represented by t_{1end}, t_{2end}, and and t_{3end}. The task that exchange messages with another application during runtime is represented by t_{x1}. The intermediary tasks are represented by t_i, where i ranges from 1 to 11.

In the integration logic of the CS4, the data of users that arrive from t_{start} is replicated and one copy flows towards «Human Resources System» for information about the employee. Further on t_6, the message is replicate and one copy flows towards «Payroll System», another flows towards «Mail Server», and another, towards «SMS Notifier». The sending of the notifications to the employees about the charge is done by «Mail Server» and «SMS Notifier». Examples of tasks that can be executed in parallel in the CS3 integration process are $[t_6, t_6]$, $[t_9, t_{11}, t_{14}]$, $[t_{10}, t_{12} t_{15}]$, and $[t_{13}, t_{16}]$. The Unijuí University integration process is represented by a DAG in Figure 8.8.

In CS4 integration operation typed graph, there are 16 nodes, which represent the tasks, and there are 20 edges, which represent the channels. Node

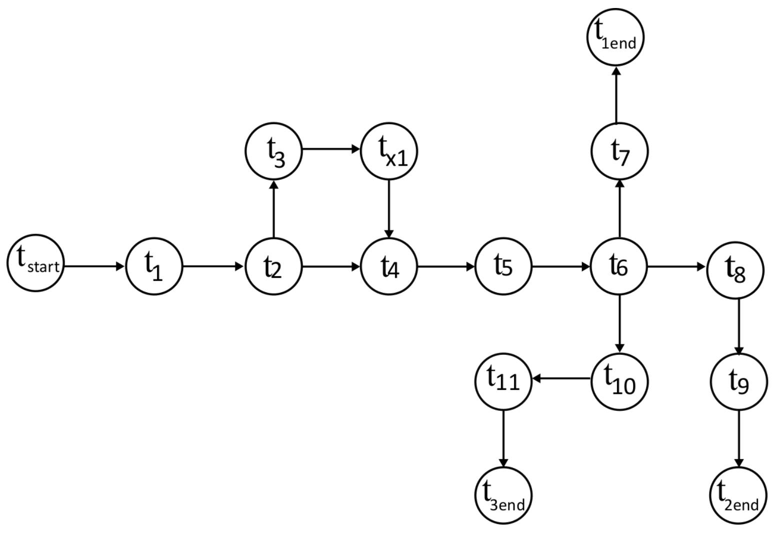


Figure 8.8: *Integration operation typed graph for case study 4.*

t_{start} is a starting node, that represent a task of the type «start». The nodes t_2 and t_6 represent tasks of the type «and»; The node t_4 represents a task of the type join; Nodes t_1, t_3, t_5, t_7, t_8, t_9, t_{10} and, t_{11} represent tasks of the type «message processor»; The node t_{x_1} represents a task that sends and receives information to/from applications that is a task of the type «external call»; and, nodes t_{1end}, t_{2end}, and t_{3end} are ending nodes that represent tasks of the type «end». The possible paths of the conceptual model `CS4` and their segments of tasks are shown in Table 8.5.

Path	**Segment**	
	Sequential	Parallel
$\{t_{1start}, t_1, t_2, t_3, t_{x1}, t_4, t_5, t_6, t_7, t_{1end}, t_8, t_9, t_{2end}\ t_{10}, t_{11}, t_{3end}\}$	$\{t_{start}, t_1\}$; $\{t_5, t_6, t_7, t_{1end}\}$; $\{t_8, t_9, t_{2end}\}$; $\{t_{10}, t_{11}, t_{3end}\}$	$\{t_2, t_3, t_{x1}, t_4\}$

Table 8.5: *Representation of case study 4 by integration logic.*

8.6 Categorisation by IOTG

A categorisation for the four case studies is summarised in Table 8.6. The first column identifies the case study; from second to eighth column are expressed the number of each type of tasks, respectively: «start», «and», «or», «xor», «join», «message processor», «external call», and «end». The ninth column expresses the number of communication channels. The tenth column shows the number of sequential segments, and the eleventh column represents the number of parallel segments.

ID	**Nº of Tasks - per function type**								**Nº of Channels**	**Nº of Segments**[†]	
	start	and	or	xor	join	message processor	external call	end		Sequential	Parallel
CS1	1	2	1	0	3	6	2	1	18	2	3
CS2	1	2	0	1	3	6	2	2	19	2	1
CS3	2	2	0	1	3	7	2	2	20	3	2
CS4	1	2	0	0	1	8	1	3	16	4	1

[†] *Considering the longest path.*

Table 8.6: *Model conceptual representation.*

8.7 Summary

In this chapter, we created a representation of integration processes according to their type of logic operations and described four case studies: Coffee Shop, Processing Order, Huelva's county council, and Phone call. The two first are benchmarks introduced by Hohpe [84], and the other two are real-world integration processes. We applied the IOTG, classifying the integration processes as to types of tasks, the number of communication channels, number sequential task segments and parallel task segments.

Chapter 9

Characterisation

A woman with a voice is, by definition, a strong woman.

Melina Gates, American philanthropist

The task scheduling of integration processes has specific characteristics that are discussed in this chapter. We have characterised the task scheduling of the execution of integration processes, analysing the primary methods, approaches, challenges, and guidelines concerning the dynamic environment involved in application integration.

9.1 Environment

In realistic problems, several events disturb the environment and degrade the performance of the execution of integration processes. We grouped the most common types of disturbance into five groups of events related to: the integration models, the messages, the computational resources, the tasks, and the queues, cf. Table 9.1. These events are also divided into internal and external. The former refers to events caused by internal elements that can be managed or monitored by the runtime system. The latter are caused by external elements, such as application, user requirements, or data source.

Model-related events are caused by factors related to the design of integration processes. Some internal events are bottlenecks and dynamic routing. Regarding bottlenecks, there are task arrangement patterns in integration process models, which, subject to certain conditions, indicate obstruction or delay the message processing [158]. The dynamic routing refers to the unpredictability of the path in which a message is processed. Usually, the integration process includes tasks that filter or route some specific messages;

Group	**Event**	
	Internal	External
Model-related	bottleneck dynamic routing	I/O delay constraint change
Message-related	-	priority change workload peak arrival rate random
Resource-related	unavailability idleness deadlock	constraint change
Task-related	starvation processing time random	-
Queue-related	overload	constraint change

Table 9.1: *Common disturbances in the integration environment.*

thus, the tasks in which a message will be processed are uncertain. Also, there are external events such as I/O delay and constraint change. I/O delay refers to the interruptions or delays caused by applications, database or any other component in which messages are processed. Constraint change refers to adjustments to meet new business requirements, such as a change in the message processing maximum time or message processing minimum rate.

Message-related events are caused by unexpected external events, such as change of priorities, workload peak, and the random arrival rate. Priority change refers to a change in the order of message processing or task execution. Scheduling based on the priority of task specifies a decision criterion to select the task to be executed first, for example, the earliest finish time [37, 208]. Peak load refers to the simultaneous arrival of a large number of messages. Random arrival rate refers to the unpredictability of the arriving message number per unit of time.

Resource-related events are caused by factors related to threads managed by the runtime system. Some internal events are unavailability, idleness, and deadlock. Unavailability refers to the lack of the number of threads for task execution, causing inefficient message processing. Idleness refers to the oversizing number of threads, leading to financial costs for the enterprise that has under-utilised computational resources. Deadlock is directly linked to bottlenecks and occurs when a thread long-executes a task and cannot be released. In this case, constraint change refers to events, such as adjustments in computational infrastructure that reduce the execution capability and machine-fault.

Task-related are events related integration tasks that compose an integration process. Some internal events are starvation, random processing time and high processing time. Starvation refers to tasks that wait for a long time to be executed; for example, when a task has low priority and threads are always busy with high priority tasks. Random processing time refers to environments that deal with Big Data or Internet of Things [149], in which data varies in volume, variety, velocity, and variability causing variation in the processing time of the tasks, which becomes unpredictable.

Queue-related are events related to queues in which tasks look forward to available threads to execute them. An internal event is an overload that occurs when there is a high-accumulation of tasks due to high message arrival rate or to unavailability of threads. Constraint change refers to events, such as adjustments in computational infrastructure that reduce the capability of storing tasks.

9.2 Approaches for scheduling

In most real-world environments, the schedule needs to cope with the presence of a variety of unexpected factors, such as workload oscillations, task delays, and resource overload. The use of static scheduling-based approaches becomes unfeasible and the near-optimal scheduling-based on approximated data, antiquate for practical use [193].

Chaari et al. [33] classify dynamic scheduling into «proactive», «reactive», and «hybrid», cf. Figure 9.1. These scheduling approaches are more appropriated to deal with environmental uncertainties [3, 42, 125].

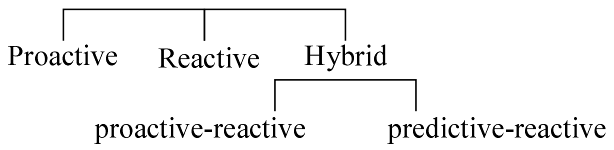


Figure 9.1: *Dynamic scheduling approaches.*

Proactive scheduling deals with uncertainties in design time. It produces one or more scheduling endowed with flexibility that becomes responsive to uncertainties. Thus, this approach is also called robust scheduling [116]. Reactive scheduling is used in highly disturbed environments in which uncertainties are both frequent and abundant. Thus, decisions are quickly made during runtime. Hybrid scheduling can be proactive-reactive or predictive-reactive. The proactive-reactive approach produces a set of static schedules and adopts one of them during runtime. So, the decision-making is not made during runtime, but the scheduling can change to a previously defined scheduling capable of managing the current disturbance. The predictive-reactive approach has a deterministic schedule and adaptive scheduling. While the former produces schedules in design time, the latter makes decisions during runtime, so reacting to disturbances.

9.3 Methods for scheduling

We describe the primary methods for resource allocation and task sequencing and timing that can be adopted to optimise performance measures in integration processes.

Resource allocation

The resource provisioning in three categories: «rigid», «moldable», and «malleable» [28, 75]. When the number of cores assigned to a job is specified in design time and does not change during runtime, the resource provisioning is considered rigid. When the number of cores assigned to a job is determined by the scheduler and does not change during runtime, the provisioning is considered moldable. When the scheduler can change the number of cores assigned to a job during runtime, it is considered malleable. The classification of resource allocation in the task scheduling of integration processes can be «fixed», «limited», or «elastic» [68], as depicted in Figure 9.2.

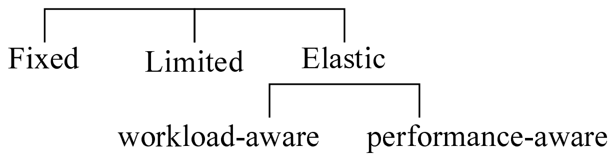


Figure 9.2: *Resource allocation approaches.*

In «fixed» resource allocation, the challenge is to optimise the scheduling with a fixed number of threads, configured in design time, to meet multiple integration process instances. Some algorithms face this challenge by emphasising the prioritisation for jobs and tasks or by the optimisation of resource utilisation [83]. In «limited» resource allocation, the number of threads can increase until a specified limit during runtime, but this number cannot change during the execution of an integration process. Thus, the challenge is to optimise the scheduling with a limited number of threads. «fixed» or «limited» provisioning occurs in environments in which the number of resources is usually limited, such as a single server, clusters, grids; as well as, in Cloud computing, when the infrastructure leased has a limited and unchanged number of computational resources.

«Elastic» resource allocation occurs in virtual environments, like Cloud computing, in which the number of computational resources increases and decreases to better meet the demands of applications running in the infrastructure [41]. Main approaches to «elastic» resource provisioning are «workload-aware» and «performance-aware» [83]. In «workload-aware», the algorithms react according to the workload status, acquiring more resources in peak demand situations or releasing resources in low demand situations. A strategy used by a scheduler can be deciding between reusing or hiring a new resource, based on the deadline of the task. Thus, the additional provisioning resources is more accurate because it is based on the task requirement being scheduled [156]. In «performance-aware», the algorithms react according to the total resource utilisation status, acquiring more resources in peak demand situations or releasing resources in low demand situations. This approach is usually present in environments of homogeneous virtual machines because it simplifies the selection process of the virtual machine by the algorithm [153].

Regarding the number of threads allocated for the execution of tasks of integration processes, although it is not limited, it is more realistic to assume that there is some constraint on it. Parallel task processing is only possible when there are multiple physical cores in a machine where the integration process is executed. The number of cores directly impacts the execution of the tasks by threads created by the runtime system [122], and thus, it influences the performance of the execution of the integration process. The number of threads is configurable and, theoretically, unlimited. However, a high number of threads can degrade the performance of the execution because more time has to be spent on managing these threads and shared resources, such as cache capacity or memory bandwidth tend to quickly saturate [111]. There are several possible configurations of thread pool that a runtime system can implement, such as *(i)* a thread pool with predefined and fixed number of threads, *(ii)* a thread pool with an unlimited number of threads that increases according to demand; and, *(iii)* a thread pool that distributes the workload amongst the available cores.

Task sequencing and timing

There are two approaches to carry out sequencing and timing scheduling [110]. One of them classifies the methods into «transformation-based scheduling» and «non-transformation-based scheduling». Another approach classified the methods into «immediate» and «periodic». These classifications are depicted in Figure 9.3.

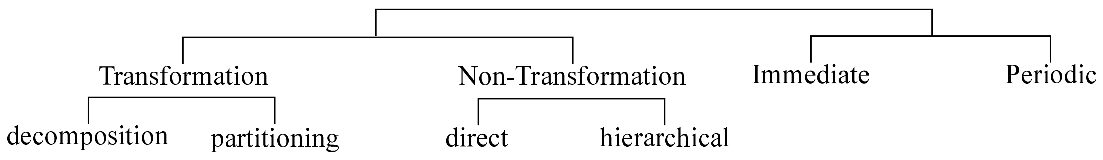


Figure 9.3: *Task sequencing and timing approaches.*

«Transformation-based scheduling» is a method that relies on transformations in integration process conceptual models to simplify constraints imposed to adapt them to known models to use policies or heuristics for scheduling. It is divided into decomposition and partitioning. The former transforms parallel segments of a process in a set of independent sequential segments, obeying precedence constraints. Then, the tasks are scheduled either by static or dynamic scheduling algorithms. In the latter, the tasks are divided into two sets according to a given criterion. Li et al. [113] divide in groups of high and low utilisation tasks. The tasks are allocated to subsets of system cores, which are not necessarily the same. Chronaki et al. [37] divide tasks into groups of «critical» and «non-critical» tasks. Critical tasks are assigned to the fastest cores and the non-critical tasks to the slowest cores. Transformation-based scheduling is a proactive method that usually adds an overhead waiting delay in environments of continuous arrival of messages, in which the actual scheduling takes place only after the previous processing associated with the transformation has occurred. Non-transformation-based scheduling is a reactive method that does not know the integration process conceptual models in advance, and the scheduling decisions are made during runtime. It is classified into two types: «direct» or «hierarchical». The former assigns priorities to job-level and tasks, and this is done during runtime. In the latter, two scheduling schemes are applied: one determines the next job to be executed, and the other determines the task to be executed.

The time complexity for heuristic algorithms must be short to deal with the dynamicity of the environment; then it must use lightweight algorithms.

Meta-heuristics must be avoided in scheduling algorithms because they are computationally intensive to plan the schedule before runtime, in which the planning time may take longer than the integration process execution itself. Alternatively, the reactive or hybrid methods reduce the intensive computing at the planning phase to make quick decisions based on the current status of the environment. These methods are classified in two types: «immediate» or «periodic» scheduling [83]. The immediate scheduling method usually has three phases: first, the task queue is built with some prioritisation method; second, the algorithm selects a particular task; and, third the algorithm selects the appropriate resource for that particular task. Periodic scheduling systematically performs the scheduling to adequate it to the status environment. This approach is considered hybrid because it has a static and a dynamic part. In the static part, the algorithm investigates the characteristics of a set of tasks, such as structures and estimated runtime. The goal is to build an optimal plan, based on a set of tasks available in a certain period, instead of the entire workload. In the dynamic part, the algorithms change the schedule plan periodically. The static part of this method is faster than those purely static scheduling methods because it only includes a small fraction of workload to be optimised before runtime.

9.4 Challenges and guidelines

Parunak [140] outlines the main challenges in scheduling: desirability, stochasticity, tractability, chaos, and decidability. Desirability means finding a schedule that provides results that are closer to the ones expected by the company. Mathematical and dynamic programming [119] and evaluation [106] are the most used approaches to address this challenge. Mathematical and dynamic programming seeks optimal scheduling using interacting constraints; however, in complex environments, this can demand excessive computational resources. Evaluation techniques experiment and evaluate several solutions by simulations or by simplification of the behaviour of the environment. However, in complex environments, the exhaustive search of the set of solutions to evaluate can demand excessive time.

Stochasticity characterises the deviation of the real parameters defined in the computational model. This deviation is used for scheduling. Such deviation arises from unexpected events, such as breakdowns, overload, bottlenecks, variation in processing times, and interference. Simulation techniques can provide mechanisms to describe pseudo-random samples from a variety of standard statistical distributions efficiently, but, the number of required simulations to encompass a statistically significant number of events can become prohibitive. Rescheduling [16], Deferred Commitment [206], and Tweaking [72] are the standard techniques for this challenge. Rescheduling monitors the deviation, changing the schedule when this deviation exceeds a given constraint; however, this technique is only feasible when the time required to schedule is short enough to accompany changes in the environment. Deferred Commitment scheduling designs high-level decisions and, during runtime, refines these decisions based on the current status of the environment. Tweaking analyses the deviation of a mathematical model of the scheduling within certain limits to decide when to reschedule.

Tractability refers to with the computational difficulty to analyse complex environments behaviours in a reasonable time and cost. Usually, heuristics techniques are used to find a near-optimal solution, such as dispatching rules [8], constraint propagation [204], stochastic search [135], and predictive performance modelling [197]. Dispatching rules selects the next task to be executed at a resource, obeying predefined rules. Constraint propagation deals directly with predefined constraints, identifying interactions amongst them and performing changes to reduce disparity amongst them. Stochastic search uses meta-heuristics that allow the search on a broader solution space from an initial set of random solutions. Predictive performance modelling extrapolates historical observations to predictions of future situations without requiring detailed information about the workload.

Chaos concerns with an environment in which these conditions never repeat themselves and small differences in these initial conditions can result in exponential divergences in the environment during runtime, so, it is not possible to predict its behaviour. The techniques suggested are bidding interval [43], information uncertainty [31], and structure of chaos [89]. Bidding interval seeks a threshold below which the environment is not chaotic. Usually, this value is determined by simulation. Information uncertainty inserts noise to the environment to block chaotic behaviour. Although this strategy conflicts with the problem of stochasticity it can be used as a component in a compromise solution. Structure of chaos identifies configurations of the environment favourable to chaos to avoid them.

Decidability concerns with the capacity for analysing environmental behaviour to identify undecidable problems. Undecidable problems are the ones in which there is no computationally feasible solution in a reasonable time. An strategy used in such problems is the participatory scheduling, which involves job processing as a partner in the scheduling process. Techniques, as neural models [46], allow the computational adaptation or learning, adjusting the model to the characteristics of the problem domain.

Participatory scheduling addresses other challenges as to deal with the complexity through heuristic search; to deal with the stochasticity through real-time distributed computation, and to deal with the desirability through adaptive behaviour.

9.5 Features

The elements of task scheduling of integration processes are not prior known and that change in runtime, so they require an dynamic approach, can be «proactive», «reactive» or «hybrid».

- The scheduling has the following features:
  - ◇ number of tasks and computational resources,in our approach, threads, are unknown;
  - ◇ there are different and unpredictable paths of tasks;
  - ◇ each task performs an atomic operation;
  - ◇ task processing time is variable;
  - ◇ tasks wait for available computational resources into queues;
  - ◇ there are parallel segments, in which tasks can be executed in different cores;
  - ◇ there are sequential segments, in which tasks must be executed in a precedence order;
  - ◇ there are no precedence constraints amongst instances of tasks of different instances of jobs.
- The computational resources type must be «elastic» to take advantage of the cloud computing capacity of increasing and decreasing the number of cores, according to the demand.
- The resource allocation can have the following approaches: «workload-aware» and «performance-aware». In the former, the algorithms use the deadline of the task to decide whether to reuse the core after the current execution or to get a new core, hiring a new virtual machine. In the latter, the algorithms react based on the total resource utilisation status.
- The task sequencing and timing can have two approaches: The former is to use low complexity heuristic to deal with dynamic environments. The latter is to adopt hybrid methods to make quick decisions based on the current environment status.

- The scheduling can have the following representation: an Integration Operation Typed Graph that enables the description of constraints on the execution and logic operation of the tasks.

9.6 Summary

We characterised the resource allocation and the task sequencing and timing involved in the execution of integration processes, providing directives that allow researchers to propose and evaluate heuristics to deal with disturbances present in the integration of applications environment.

Chapter 10

Queue Priority Heuristic

Mankind has made giant steps forward. However, what we know is really very, very little compared to what we still have to know.

Fabiola Gianotti, Italian physicist

In this chapter, we formulate our research problem as mathematical expressions, define our objective function, and propose a heuristic to task scheduling of integration processes. This heuristic, namely Queue Priority (qPrior), deals with overload situations caused by workloads of large volumes of data. We implement qPrior by a lightweight algorithm to increase the performance of the execution of the integration processes.

10.1 Problem formulation

In this chapter we present the matematical modelling of integration processes, with focus on time-related performance metrics, commonly used for scheduling performace bechmarking. Rodriguez and Buyya [157] propose Equation 10.1 to calculate the total processing time of a task in a virtual machine (VM) in scientific workflow scheduling in a cloud environment. «$ET_{t_i}^{VM_j}$» is the task execution time t_i in a virtual machine of type VM_j; «$TT_{e_{ij}}$» is the time it takes to transfer data between a task t_i and its successor t_j; k is the number of edges in which t_i is the predecessor task and s_k is 0 whenever t_i and t_j run on the same virtual machine or 1 otherwise.

$$TP_{t_i}^{VM_j} = ET_{t_i}^{VM_j} + \left(\sum_{1}^{k} TT_{e_{ij}} * s_k \right) \tag{10.1}$$

Makespan is a well-known metric for performance within the integration community. According to Chirkin et al. [36], the processing time in a workflow can be represented by Equation 10.2. In this equation, «ET» is the task execution time, «T_R» is the resource preparation time, «T_Q» is the queuing time, «T_D» is the data transfer time, and «T_O» is the overhead system time, such as time spent in analysing the task structure, selecting the resource, amongst others.

$$TP = ET + (T_R + T_Q + T_D + T_O) \tag{10.2}$$

For Shishido et al. [178], the total processing time of a task «t_i» in a virtual machine of type vm_s^k is the sum of the task execution time «$TE(t_i, vm_s^k)$», transfer time «$TT(t_i)$», and security services overhead of the «$SC(t_i)$» as defined in Equation 10.3.

$$TP\left(t_i, vm_s^k\right) = TE\left(t_i, vm_s^k\right) + [TT\left(t_i\right) + SC\left(t_i\right)] \tag{10.3}$$

Several researchers use the makespan arithmetic average as a performance metric for scheduling algorithms [1, 35, 115]. According to Canon and Jeannot [30], makespan is computed by instantiating every computation and communication duration according to random variables, i.e. it is the end-time of the processing last task. For Abdulhamid et al. [1], the lower the makespan, the better the processing efficiency, meaning less processing time. They defined Equation 10.4, in which makespan is the maximum time needed to complete processing «$\max C_i{}'$».

$$Makespan = \{\max C_i{}'\} = \max\{C_1{}', C_2{}', ..., C_n{}'\} \tag{10.4}$$

In integration processes, we define the internal task total processing time in a thread pool as computed by Equation 10.5. In this equation, «TP_{t_i}» is the total processing time; «ET_{t_i}» is a task execution time in a thread into its respective thread pool; «TQ_{t_i}» is the task waiting time in a task queue; and, «k» is the number of edges to which the current task is its successor.

$$TP_{t_i} = ET_{t_i} + \sum_{1}^{k} TQ_{t_i} \tag{10.5}$$

We defined the total processing time of a message «TP_{m_i}» as the elapsed time interval between the time a message entered and the time it leaves the integration process. «TP_{m_i}» is the sum of the execution time of all the tasks of the path by which the message must flow for its complete processing, c.f. Equation 10.6. We assume that the execution time of a task, «ET_{t_k}» includes all the times involved, such as the total CPU time, the annotation time of tasks in queues, the waiting time of the tasks in a queue; and, the waiting time of the task in request and response operations with external applications. The number of tasks in the path is represented by «tot». We also assume that the range of the execution time of a task «t_k» is defined as «$[ET_{t_{k_{ini}}}, ET_{t_{k_{fin}}}]$».

$$TP_{m_i} = \sum_{1}^{tot} ET_{t_k}, \textit{ where } \{ET_{t_k} \in \mathbb{R} \mid ET_{t_{k_{ini}}} \leq ET_{t_k} \leq ET_{t_{k_{fin}}}\} \tag{10.6}$$

Then, we define the makespan of an execution of an integration process as the total execution time of an integration process. It is the elapse time between the start time of the first message that entered «ST_{m_1}» and the end time of the last message that leaves the workflow «$ET_{m_{np}}$», cf. Equation 10.7. Throughput is a performance metric based on the amount of work a system can perform in a given time in a particular environment. The throughput of a computer system is a function of the environment and workload characteristics. Improvements resulting from system changes can be evaluated by throughput metrics [190, 199]. In case of execution of integration processes, throughput corresponds to the number of messages processed per time unit and it is calculated by computing the division of the total number of processed messages «np» by the total execution time «$Makespan$», cf. Equation 10.8. This formulation is represented by the objective function shown in Equation 10.9.

$$Makespan = ET_{m_{np}} - ST_{m_1} \tag{10.7}$$

$$Throughput = \frac{np}{Makespan} \tag{10.8}$$

$$\max\{Throughput\} \tag{10.9}$$

Time optimisation becomes fundamental in situations in which the process execution duration must meet specific constraints or deadlines, because, their violations increase the business processes costs [142]. Thus, the problem can be formulated as:

> *to find out a heuristic for task scheduling that processes more messages per time unit in the execution of integration processes in overload situations.*

10.2 Queue Priority heuristic

We have development the Queue Priority (qPrior) heuristic. The goal is to keep a good performance of the execution of integration processes even if such overload situations occur. In qPrior, there are multiple task queues. Every task queue maintains instances of a specific task of integration process. Tasks that can be executed in parallel are maintained in same queue. Threads, grouped in a thread pool, check the queues following in an order of priority. In this order, a task that has more predecessor tasks has more priority in its execution. Each time threads check a queue, a specific number of tasks are captured to be executed by those threads. We refer to number of tasks that is executed as preemption. Tasks are allocated permanently to threads till the tasks are entirely executed.

To exemplify the qPrior heuristic, we present four possible iterations of the heuristic illustrated in Figure 10.1. This example is a snapshot of a particular moment(*n-th* iteration) of the integration process in runtime, in which, instances tasks wait in three task queues. The queue of high priority maintains the annotations for executions of the instances of task t_3. This task is the task that has more predecessor tasks. The queue of medium priority maintains the annotations for executions of the instances of task t_2. The queue of low priority maintains the annotations for executions of the instances of task t_1. This task has no predecessor task. Assuming the software engineer set the preemption to the value six, which means that, at every time that threads of pool check a queue, they will execute six tasks that have annotation in this queue. The annotations of the tasks that are caught by threads are shaded in grey. The iterations take happen as follow:

***n-th* iteration:** threads poll the queue of high priority and, then, they execute six tasks of queue of tasks t_3.

***(n+1)-th* iteration:** because the queue of high priority is empty, threads poll the next queue of more top priority and, then, they execute six tasks t_2 of this queue. After, the executions of tasks t_2 produce outbound messages that are inbound messages for successor task, t_3. So, the scheduler annotates the executions in the queue of tasks $t - 3$. While this happens, new inbound messages continue arriving for the integration process generating new annotations in the queue of task t_1.

***(n+2)-th* iteration:** threads poll the queue of high priority and, then, they execute six tasks of queue of tasks t_3.While new tasks continue being annotated in the queue that maintains the tasks t_1.

***(n+3)-th* iteration:** because the two queues more top priority are empty, threads poll the queue of lowest priority and, then, they execute six tasks t_1 of this queue. After, the executions of tasks t_1 produce outbound messages that are inbound messages for successor task, t_2. So, the scheduler annotates the executions in the queue of tasks t_2.

This process iteratively continues till up all queues are empty or till up it is interrupted.

10.3 qPrior algorithm

We have implemented qPrior algorithm that the software engineer to perform the Queue Priority heuristic. The activities of this heuristic is shown in Figure 10.2 and the pseudo-code of the algorithm that implements the qPrior heuristic is shown in Algorithm 10.1.

This algorithm receives the total number of tasks and the number of tasks performed at a time «preemption». The algorithm starts by initialising the auxiliary variables: «totalSize», «preempt», and «qPrior». The first variable corresponds to the total size of queues, the second variable is the preemption, and the third variable maintains the queue indication that will be polled. The algorithm verifies the queues from the highest priority queue until the lowest priority queue. After the algorithm selects a queue, it sets the preemption according to the following rule: if the queue size is smaller than «preempt», the threads executes all tasks that are in the queue; otherwise, the algorithm executes an amount of task equals «preempt». The algorithm keeps allocating threads to execute tasks until all the queues are empty. The algorithm uses a thread pool that creates threads as needed. However, it is possible to reuse previously constructed threads when they are available. When there is no available thread, a new thread will be created and added to the pool. Threads that have not been used for sixty seconds are shut up and removed from the pool [†1].

10.4 PSO modelling

When there is a high arrival rate of messages, the queues accumulate more annotations of the initial tasks and threads keep busy in the

[†1]docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executors.html

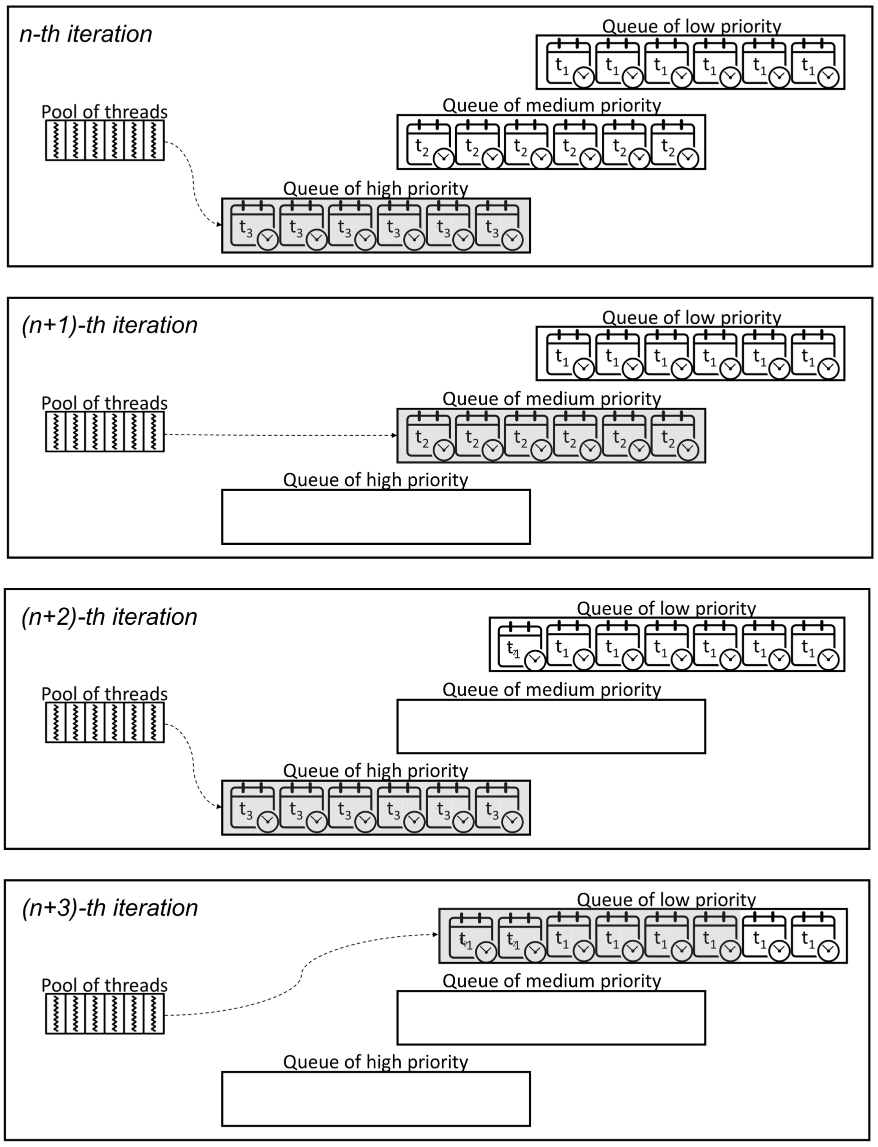


Figure 10.1: *Iterations of the qPrior heuristic.*

execution of these tasks to the detriment of the others. This accumulation occurs either in the task queue of the FIFO heuristic and in queues of initial tasks of qPrior heuristic. Therefore, the preemption used by qPrior algorithm impacts the total execution time of an integration process, if the preemption is large, the algorithm spends more time in execution of the initial tasks by threads, since the size of queues of initial tasks tends to be bigger. On the other hand, if the preemption is too small, the algorithm can spend more time with the exchange of queues. It is a challenge for the software en-

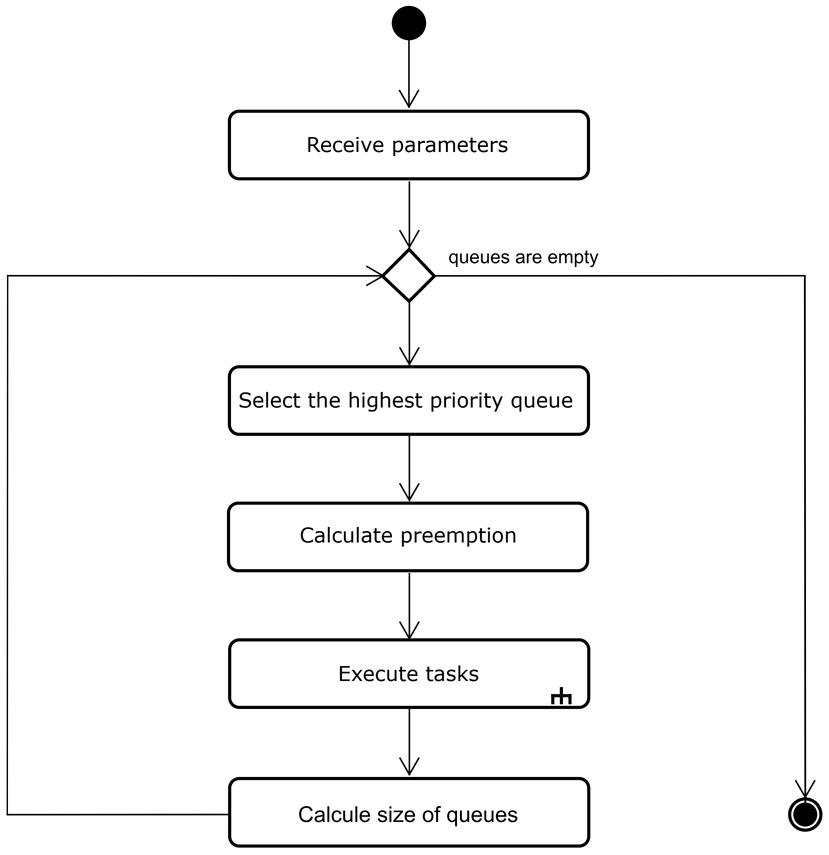


Figure 10.2: *qPrior heuristic.*

gineer to find the ideal preemption, i.e., the optimum number of task instances that must be executed by threads at each polling to a queue.

Thus, the problem can be formulated as:

to determinate the number of tasks included in the preemption for the task scheduling carried out by the qPrior algorithm, which maximises the number of messages processed per time unit in the execution of integration processes under high workloads.

We model this problem as a PSO problem and use this meta-heuristic to find the optimum preemption for the task scheduling heuristic by qPrior. For our scheduling problem, a particle represents an execution of an integration process in an overload situation. The position of a particle is a range for the preemption. Thus, the particle position is two-dimensional, specified by

```
Input: Task queues: $queues[\,]$
Input: Maximum duration of the simulation: $maxDuration$
Input: Time start of the simulation: $start$
Input: Total number of tasks: $numTasks$
Input: Number of tasks performed at a time (preemption): $preemptTask$
   $totalSize \leftarrow 1$
   $preeemp \leftarrow preemptTask$
   $qPrior \leftarrow numTasks$
   while $totalSize > 0$ and $duration < maxDuration)$
     for$[i] = numTasks$ to 1 with step$-1$
         if $queues[i] \neq \emptyset$
             $qPrior \leftarrow i$
             $i \leftarrow 1$
         end if
     end for
     if $(preempt = 0)$ or $(queues[qPrior].size < preempt)$
             $preempt \leftarrow queues[qPrior].size$
     else
             $preempt \leftarrow preemptTask$
     end if

      $Allocate\ Thread(queues[qPrior], preempt)$
     end while
     $totalSize \leftarrow 0$
     for$[i] = 1$ to $numTasks$
        $totalSize \leftarrow totalSize + queues[i].size$
     end for
     $duration \leftarrow$ current.Time $-start$
```

Program 10.1: *qPrior algorithm.*

two coordinates, which are the minimal and maximal value of the preemption range. In our case, the minimal value of the preemption range can be at least 1 and the maximal can be at most the total workload to be processed by the integration process. According to the guidelines of the literature, it is recommended to use half of the total workload as maximal value for this range.

The objective function must be linked to the goals of the scheduling problem because it is used to determine if a potential solution is good enough. The goal of scheduling is to maximise the number of processed messages per time unit in the execution of integration processes. In PSO modelling, the value of the objective function is defined as «maximise», which means maximising the throughput average related to the execution derived from the position of the particle. A strategy to define the initial preemption to the algorithm uses to explore different solutions and achieve the goal of scheduling. This strategy must reflect the unpredictability of the possible paths that a message flows in the execution of an integration process. So, it is necessary to provide enough options for PSO to produce a optimum particle position (solution). If the range is vast, then the search space explored by PSO is also huge, hence the algorithm may take a long time to converge and find the near-optimal solution. A strategy to limit the range of preemption was adopted, based on software engineers expertise in application integration, to reduce the size of the search space. The algorithm stopping criterion parameter was set to the number of iterations supported by the memory capacity of the computer used to execute the PSO. Table 10.1 summarises the PSO parameters for a total workload of 2,000,000 messages. Algorithm 6.1 uses Algorithm 6.1 to calculate the optimal value of the throughput, cf. Equation 10.9.

Parameter	**Value**
ω	1
ϕ_p	1
ϕ_g	1
particle number	20
particle dimension	2
range	[25000, 50000], [50000, 75000]
number of iteration	100

Table 10.1: *PSO parameters.*

10.5 Summary

In this chapter, we mathematically formulated the problem of scheduling heuristic of integration processes, and then, we proposed a new task scheduling heuristic for integration processes in overload situations, qPrior. This heuristic separates the tasks that wait for available threads in multiple prioritised queues and threads select task to execute according to the priority of the queue. At each time that a thread polls a queue, it selects a predefined number of task to execute. We used the Particle Swarm Optimisation meta-heuristic to find the near-optimal number of tasks that must be selected to reach better performance in the execution of integration processes.

Chapter 11
Mathematical Model

Life shrinks or expands in proportion to one's courage.

Anais Nin, French-Cuban American writer (1903–1977)

Mathematical models as linear approaches can express the relationship between the dependent and the independent variables. In this chapter, we propose mathematical models that allow prediction of the average of processed messages in the executions of integration processes, which use the qPrior heuristic.

11.1 Processed messages *vs.* workload

The first mathematical model (model 1) is a polynomial equation of degree 3 that expresses the average of processed messages as a function of the workload. It is shown in Equation 11.1.

$$\overline{\mathrm{pm}} \sim 1 + w + w^2 + w^3 \tag{11.1}$$

where:
$\overline{\mathrm{pm}}$: average of processed messages
w: workload

The terms of Equation 11.1 are indicated in Table 11.1. The column «estimate» indicates the coefficients and constant terms of this equation and are express in Wilkinson notation [196]. Column «ID» indicates the case study,

ID	Term	Estimate	R^2
CS1	(Intercept)	$1.16\ 10^{6}$	1
	w	-1.65	
	w^2	$1.97\ 10^{-6}$	
	w^3	$-4.76\ 10^{-13}$	
CS2	(Intercept)	$6.76\ 10^{5}$	1
	w	-0.46	
	w^2	$1.01\ 10^{-6}$	
	w^3	$-2.25\ 10^{-13}$	
CS3	(Intercept)	$-2.08\ 10^{6}$	1
	w	4.97	
	w^2	$-2.24\ 10^{-6}$	
	w^3	$3.43\ 10^{-13}$	
CS4	(Intercept)	$3.05\ 10^{6}$	1
	w	-4.60	
	w^2	$4.57\ 10^{-6}$	
	w^3	$-1.02\ 10^{-12}$	

Table 11.1: *Coefficients model 1.*

and column «R^2» indicates the correlation coefficient. Table 11.2 shows the observed values of the averages of processed messages and the estimate found by our model.

The behaviour of the average of processed messages as function of workload by model 1 is expresses by dotted lines of scatter chart, shown in Figure 11.1. In the execution considering a total workload of 1,000,000 and 1,500,000 messages (msg), the four case studies processed all messages, thus, up to these workload, the behaviour of the average of processed messages is a linear of slope equals 1. Considering a total workload of 2,000,000 msg, this behaviour remained for `CS2` and `CS4`, but it begin to change for `CS1` and `CS3`, which processed a smaller average of messages than the workload. In the execution of the workload of 2,500,000 msg, all case studies processed a smaller average of messages than the workload, establishing the behaviour of a decreasing curve for all case studies.

ID	Workload	Averages of processed messages	
		Observed	Estimated
CS1	1000000	1000000	1004000
	1500000	1500000	1511000
	2000000	1911304	1932000
	2500000	1876829	1910000
CS2	1000000	1000000	1011523
	1500000	1500000	1554373
	2000000	2000000	2147223
	2500000	2331044	2640073
CS3	1000000	1000000	996798
	1500000	1500000	1489192
	2000000	1651450	1625832
	2500000	1711751	1661716
CS4	1000000	2000000	2015876
	1500000	3000000	3053581
	2000000	4000000	4127006
	2500000	4238093	4486151

Table 11.2: *Comparison observed vs. estimated for model 1.*

11.2 Processed messages vs. IOTG

The second mathematical model (model 2) is a generic polynomial equation that expresses the average of processed messages as a function of all elements of IOTG, namely, number of every type of task, number of communication channels, the number of sequential and parallel segments. It is shown in Equation 11.2.

$$\overline{pm} \sim 1 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} \quad (11.2)$$

where:
$\overline{pm}$: average of processed messages
x_1: number of tasks of type «start»
x_2: number of tasks of type «and»

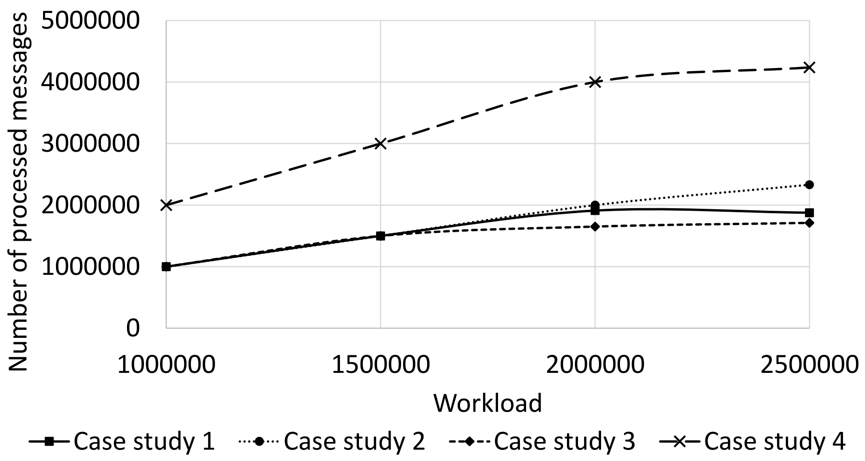


Figure 11.1: *Average of processed messages by model 1.*

x_3: number of tasks of type «or»
x_4: number of tasks of type «xor»
x_5: number of tasks of type «join»
x_6: number of tasks of type «message processor»
x_7: number of tasks of type «external call»
x_8: number of tasks of type «end»
x_9: number of communication channels
x_{10}: number of sequential segments
x_{11}: number of parallel segments

We use the stepwise statistical method to build a specific mathematical model for every workload, which are shown in the «model» column of Table 11.3. In this table, column «workload» indicates the workload in Table 11.3, column «estimate» indicates the coefficients and constant term of this equation, and column «R^2» indicates the correlation coefficient of the model. Table 11.4 shows the observed values of the averages of processed messages and the estimate of these averages found by the model. It is possible to observe that with a workload from 1,000,000 to 1,500,000 messages, qPrior can process all workload in four cases. In case of this interval of workload, tasks of type «join» «x_5» are the ones that have more impact on the average of processed messages. With a workload from 2,000,000 to 2,500,000 msg, qPrior starts to decrease the productivity and can process workload partially. In the case of this interval of workload, the tasks of type «external call» «x_7» are ones that more impact the average of processed messages.

Workload	Model	Term	Estimated	R^2
1000000	$1 + x_5 + x_6$	(Intercept)	$2.50\ 10^6$	1
		x_5	$-5\ 10^5$	
		x_6	$-2.05\ 10^{-10}$	
1500000	$1 + x_1 + x_5$	(Intercept)	$3.75\ 10^6$	1
		x_1	$7.11\ 10^{-10}$	
		x_5	$-7.50\ 10^5$	
2000000	$1 + x_7$	(Intercept)	$6.15\ 10^6$	0.98
		x_7	$-2.15\ 10^6$	
2500000	$1 + x_7$	(Intercept)	$6.50\ 10^6$	0.95
		x_7	$-2.27\ 10^6$	

Table 11.3: *Coefficients of model 2.*

Workload	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	Observed	Estimated
1000000	1	2	1	0	3	6	2	1	18	2	3	1000000	1000000
	1	2	0	1	3	6	2	2	19	2	1	1000000	1000000
	2	2	0	1	3	7	2	2	20	3	2	1000000	1000000
	1	2	0	0	1	8	1	3	16	4	1	2000000	2000000
1500000	1	2	1	0	3	6	2	1	18	2	3	1500000	1500000
	1	2	0	1	3	6	2	2	19	2	1	1500000	1500000
	2	2	0	1	3	7	2	2	20	3	2	1500000	1500000
	1	2	0	0	1	8	1	3	16	4	1	3000000	3000000
2000000	1	2	1	0	3	6	2	1	18	2	3	1911304	1854300
	1	2	0	1	3	6	2	2	19	2	1	2000000	1854300
	2	2	0	1	3	7	2	2	20	3	2	1651450	1854300
	1	2	0	0	1	8	1	3	16	4	1	4000000	4000000
2500000	1	2	1	0	3	6	2	1	18	2	3	1876829	1973200
	1	2	0	1	3	6	2	2	19	2	1	2331044	1973200
	2	2	0	1	3	7	2	2	20	3	2	1711751	1973200
	1	2	0	0	1	8	1	3	16	4	1	4238093	4238100

Table 11.4: *Comparison observed vs. estimated for model 2.*

11.3 Generic model for average processed messages

The third mathematical model (model 3) is a generic polynomial equation that expresses the average of processed messages as a function of both the workload and the elements of IOTG. The method of stepwise linear regression allows removing of the variables less statistically significant or the weakly correlated. The resulting model is shown in Equation 11.3, in which the workload, the tasks of type «$\mathtt{message\ processor}$», the communication channels and the parallel segments are the components that impact the average of processed messages in the execution of integration processes with the qPrior heuristic.

$$\overline{pm} \sim 1 + w + x_6 + x_9 + x_{11} \tag{11.3}$$

where:
$\overline{pm}$: average of processed messages
w: workload
x_6: number of tasks of type «$\mathtt{message\ processor}$»
x_9: number of communication channels.
x_{11}: number of parallel segments.

The coefficients and constant term of Equation 11.3 are indicated in the column «estimate» of Table 11.5. The correlation coefficient indicates that the model explains about 92% of the variability in the response. The observed averages of processed messages and the ones found by the generic model are shown in Table 11.6.

Term	**Estimate**
(Intercept)	$4.70\ 10^6$
w	$0.88\ 10^5$
x_6	$3.13\ 10^5$
x_9	$-3.25\ 10^5$
x_{11}	$-2.30\ 10^5$

Table 11.5: *Coefficients of model 3.*

Workload	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	**Observed**	**Estimated**
1000000	1	2	1	0	3	6	2	1	18	2	3	1000000	914437
	1	2	0	1	3	6	2	2	19	2	1	1000000	1050165
	2	2	0	1	3	7	2	2	20	3	2	1000000	808204
	1	2	0	0	1	8	1	3	16	4	1	2000000	2651927
1500000	1	2	1	0	3	6	2	1	18	2	3	1500000	1352834
	1	2	0	1	3	6	2	2	19	2	1	1500000	1488562
	2	2	0	1	3	7	2	2	20	3	2	1500000	1246601
	1	2	0	0	1	8	1	3	16	4	1	3000000	3090324
2000000	1	2	1	0	3	6	2	1	18	2	3	1911304	1791232
	1	2	0	1	3	6	2	2	19	2	1	2000000	1926960
	2	2	0	1	3	7	2	2	20	3	2	1651450	1684999
	1	2	0	0	1	8	1	3	16	4	1	4000000	3528722
2500000	1	2	1	0	3	6	2	1	18	2	3	1876829	2229630
	1	2	0	1	3	6	2	2	19	2	1	2331044	2365357
	2	2	0	1	3	7	2	2	20	3	2	1711751	2123397
	1	2	0	0	1	8	1	3	16	4	1	4238093	3967120

Table 11.6: *Comparison observed vs. estimated model 3.*

11.4 Comparison of the models

The first mathematical model takes account of only the workload, there is a single equation, but the coefficients are specific for each integration process. Model 1 accurately represents the behaviour of the number of processed messages for all case studies tested, with the correlation coefficient equals to 1. The second mathematical model considers the elements of the IOTG that represent each integration process, so there are specific equations and coefficients for each workload. The elements that influence this model are the number of tasks «start», «join», «message processor», and «external call». Model 2 has a correlation coefficient above 0.95. The third mathematical model regards either the workload and the elements of the IOTG, so there are a single equation and coefficients for all studies cases and workloads. The elements that influence this model are the workload, the number of tasks «message processor», the number of communication channels and the number of parallel segments. Model 3 is the most generic yet also has a high correlation coefficient, which equals to 0.92. Table 11.7 summarises the influences, the coverage, and the accuracy of every mathematical model for the average of processed.

Model	**Influence elements**	**Coverage**	**Accuracy (R^2)**
Model 1	workload	single equation and specific coefficients	1
Model 2	tasks «start», «join», «message processor», and «external call»	specific equations and coefficients	0.95 to 1
Model 3	workload, tasks «message processor», communication channels, and parallel segments	single equation and coefficients	0.92

Table 11.7: *Comparison of the mathematical models.*

11.5 Summary

We have developed mathematical models for prediction of the average of processed messages in executions of integration processes by the qPrior heuristic by means of the statistic technique of linear regression. The first model for this average is a function of the workload; the second, a function of the elements of their representation by an integration operation typed graph; and, the third model, a function of both workload and IOTG elements. For every model, we indicated the coefficients and constant terms as well as the correlation coefficient. Then, we applied the model and compared the observed and estimated values.

Part IV
Validation

Chapter 12
Integration Process Simulator

To be able to code gives you the freedom to build anything and that is just so empowering.

Karlie Kloss, American fashion model and entrepreneur

The behaviour of the execution of integration processes can be observed safely by simulations, which allow the analysis of different scenarios with a good level of details. Besides, simulation is usually less expensive and take less time than experiments with real assets. In this chapter, we describe the Integration Process Simulator (IPS) †1, a simulator that we implemented for heuristics for task scheduling of integration processes.

12.1 Overview

The activities of the IPS shown in Figure 12.1 and its interface in Appendix B. The simulation receives the input parameters, such as the profile of the integration process, and simulates the input of messages by annotation of first tasks in a queue. Then, iteratively, it executes the heuristic selected until the time duration of the simulation to be achieved. After every simulation, it prints and stores performance metrics. The running is stopped when the limit of simulations is achieved.

The architecture of the IPS allows the incorporation of any scheduling heuristic. In its current version, IPS allows the simulation of the heuristics: First-In-First-Out (FIFO), Multi-queue Round Robin (MqRR), and Query

†1 github.com/gca-research-group/Integration-Process-Simulator

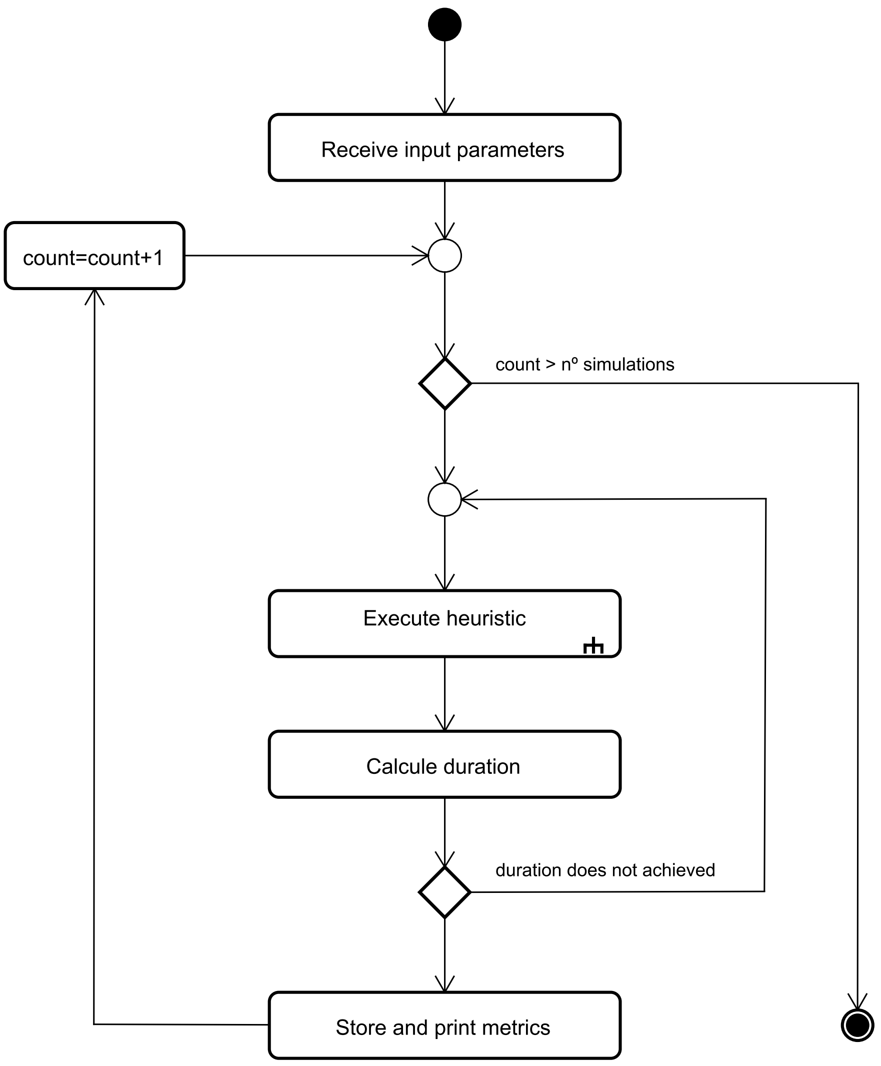


Figure 12.1: *Diagram of activities of IPS.*

Priority (qPrior). FIFO is adopted in most open source integration platforms [68], so we used as a reference of comparison for the heuristic proposed in this research. MqRR is a scheduling heuristic, based on Round Robin, a heuristic popularly known by its simplicity and efficiency in computing [185, 202, 207]. MqRR is another alternative that IPS provides as reference of comparison. qPrior is our heuristic for task scheduling that seeks to increase the performance of the execution of integration processes in an overload situation.

When a FIFO heuristic is adopted, there is a single queue of annotations of executions of tasks that enqueue the instances of all the tasks ready to be executed and are waiting for available threads. Annotations of executions of tasks are queued in ascendant order of time. In the head of the queue is the task that longest waits, and in the tail is the task that arrived most recently. Available threads recurrently poll the head of the queue for a annotation of execution of a task and, if a task is found, a thread executes entirely the task.

When the scheduling follows a MqRR heuristic, there are multiple queues of annotations of executions of tasks, and each queue maintains the annotations of executions of instances of a task. Thus, the number of queues is equals to the number of tasks of the integration process. However, annotations of executions of tasks that belong to parallel sequences are maintained in the same queue because these tasks can be executed in parallel. Annotations of executions of tasks are also maintained in the queues in ascendant order of time, and available threads recurrently poll the queues and execute tasks according to the annotations existing. A thread polls the queues of tasks in a circular order, executing a predefined number of tasks of every queue, each time. We refer to this predefined number of tasks as preemption.

The qPrior heuristic is similar to MqRR heuristic. In this heuristic, there are also multiple queues of annotations of executions of tasks; however, threads poll the queues following order of priority, in which the more predecessors a task has, the higher its execution priority.

Performance metrics, such as makespan, throughput, number of processed messages, and the number of remained messages can be obtained from IPS. It is possible to vary the time of simulation, the initial workload of messages, the total workload of messages, the rate of inbound messages, and the integration process, creating different scenarios for the simulations.

12.2 Algorithms

The IPS is compose of the ten algorithms: «Main», «Profile», «FIFO», «MqRR», «qPrior», «Receive», «Route», «Allocation», «Operation», and «Send». The codes of these algorithms are presented and detailed in the following section.

«Main» is the algorithm responsible for receiving parameters and calling the other ones. It coordinates the simulation of integration processes. Its code is shown in Algorithm 12.1. The input parameters of «Main» are the number of simulations, the maximum duration of the simulation, maximum number of messages (workload), the heuristic to follow, and the number of tasks performed at a time (preemption). The last input parameter is the number of tasks that must be executed every time a thread polls a queue. For the FIFO heuristic, the preemption parameter is not necessary. Firstly, the algorithm creates one or more queues for tasks. If the heuristic is FIFO, it creates a single queue; otherwise, it creates a queue for each task of the integration process. Following, it calls another algorithm that annotates the instances of the first task in the correspondent queue, after, it calls the algorithm that performs the heuristic. When the duration of simulation expires, the algorithm calculates the performance metrics and then, prints and records these numInputTasks in a text file.

«Profile» is the algorithm that contains information about an integration process and is used as parameters in the other algorithms. Its code is shown in Algorithm 12.2. It receives a Vector containing identification of tasks, a vector containing parallel tasks, a vector containing the execution time of the tasks, a vector containing the operations of the tasks, a vector containing the next tasks, and a vector containing last tasks. It calculates the number of tasks of the integration process by the length of a Vector containing identification of tasks. A profile is specific for each integration process, cf. shown in Appendix A.

«FIFO» is the algorithm that implements the First-In-First-Out heuristic for the task scheduling of integration processes and its code is shown in Algorithm 12.3. It receives a queue of tasks, the maximum duration of the simulation, and the start time of input of the first task in the queue. The algorithm starts by initialising the auxiliary variable «totsize» that corresponds to total size of a queue. After it recursively calls another algorithm to allocate threads to perform tasks until the simulation time expires or up to the time that there is no tasks in the queue.

«`MqRR`» is the algorithm that implements the Multi-queue Round Robin heuristic for the task scheduling of integration processes and its code is shown in Algorithm 12.4. It receives a queues of tasks, the maximum duration of the simulation, and the start time of input of the first task in the queue, the total number of tasks, and the number of tasks performed at a time (preemption). This last input parameter is used to indicate the number of tasks that the threads must execute every time a queue is polled. The algorithm starts by initialising two auxiliary variables: «totsize» and «preempt». The former variable corresponds to total size of a queue and the latter variable to the preemption. The algorithm checks the queues from the first to the last task, and it keeps checking them in a circular order. When the size of the queue is smaller than «preempt», the algorithm executes all tasks that are in the queue. This algorithm calls another algorithm to allocate threads to execute tasks, while there are tasks in the queue and the duration of the simulation is lower than the input parameter that stipulates the maximum duration.

«`qPrior`» is the algorithm that implements the Queue Priority heuristic for the task scheduling of integration processes and its code is shown in Algorithm 10.1 in Chapter 10. It receives a queues of tasks, the maximum duration of the simulation, and the start time of input of the first task in the queue, the total number of tasks, and the number of tasks performed at a time. This last input parameter is used to indicate the number of tasks that the threads must execute every time a queue is checked. This algorithm starts by initialising two auxiliary variables: «totsize» and «preempt». The former variable corresponds to total size of a queue and the latter variable to the preemption. The algorithm checks the queues from the highest priority queue until the lowest priority queue. When the size f the queue is smaller than value specified for variable «preempt», the algorithm executes all tasks that are in the queue; otherwise, only the number of task equals «preempt» will be executed. The algorithm calls another algorithm to allocate threads to execute tasks, while there are tasks in the queue and the duration of the simulation is lower than the input parameter that stipulates the maximum duration.

«`Allocation`» is the algorithm responsible for managing and allocating threads that execute the tasks of integration processes and its code is shown in Algorithm 12.5. It receives a queue of tasks, the maximum duration of the simulation, the maximum number of messages, the preemption, and a vector containing the ending tasks. The algorithm starts by initialising the auxiliary variable «preempt», and then it continuous with the creation of a pool of threads, which is elastic and take advantage of the multicore CPU. The algorithm selects tasks of the queue, from head to tail, until it achieves the number of tasks equals to the preemption. When the size f the queue is smaller than value specified for variable «preempt». The algorithm submits the task to the pool of threads and calls the algorithm responsible by executing the operation of tasks. After the execution, the algorithm removes the task from the current queue. Then, it checks if the task belongs to a vector containing the ending tasks. If the task is not an ending task, the algorithm annotates the task in the next queue, according to the logic of the process integration. After all there operation, the algorithm destroys the pool of threads.

«Route» is the algorithm responsible to define to next queue where a task must to be annotated, following the precedence order of the logic of an integration process and its code is shown in Algorithm 12.6. It receives a task, a vector containing the next queues of tasks, and a vector containing the operations of the tasks. It starts by initialising the auxiliary variable «operLeng» that corresponds to the length of a vector containing the operations of the tasks. When the length of this vector is equals to zero, it means that the task has single output and then it must be annotated in only one queue. Otherwise, the task must be annotated in the queues according with information contained in vector «vetorNextTask» of the profile of the integration process. If the operation of the task equals «AND, the task must be annotated in all the next queues. If the operation of the task equals «XOR», the task must be annotated in only one of the queues. The operation «OR» acts as both «AND» and «xOR», i.e., the task can be annotated in one or more queues.

«Operation» is the algorithm that simulates the execution of tasks and its code is shown in Algorithm 12.7. As previously mentioned, the operation of a task can be to transform, filter, split, join, or route messages. Each operation has an execution time with an interval of variation. This algorithm receives a task and a vector containing an interval of the execution time of tasks. Then, it randomly selects a value into an interval of execution times and waits during this time, simulating the time that the task spent executing its operation.

«Send» is the algorithm responsible for annotation of tasks in the queues, cf. Algorithm 12.8. This algorithm receives a task and a vector containing parallel tasks. The vector containing parallel tasks indicates if a task can be executed in parallel with another task. The algorithm checks the heuristic used and annotates the task in a queue according to heuristic. When the heuristic used is FIFO, the algorithm annotates the task in the FIFO queue. When the heuristic used is MqRR or qPrior, it first checks the vector containing parallel tasks and. If these tasks are parallel tasks, the algorithm annotates the task in same queue of the parallel task, else each task is annotated in its queue.

«`Receive`» is the algorithm responsible for annotating of tasks in the first queue, which corresponds to the queue of the first task of an integration process. Thus, it simulates the arrival of messages in the integration process. Its code is shown in Algorithm 12.9. This algorithm receives the number of input tasks «numInputTask» and checks the heuristic used. When the heuristic used is FIFO, the algorithm annotates the task in the FIFO queue. When the heuristic used is MqRR or qPrior, it annotates the task in the queue of the first task.

Input: Number of simulations: $numSimulation$
Input: Maximum duration of the simulation: $maxDuration$
Input: Number of inbound messages: $messInbound$
Input: Heuristic: $policy$
Input: Preemption: $preemptTask$

```
 $starTtime[], endTime[]$
while $count \leq numSimulation$
  if $policy$ = «FIFO»
      $queues[0]$
  else
      for[i] = 1 to $numTasks$
          $queues[i]$
      end for
  end if
  for[i] = 1 to $messInbound$
      $startTime[i] \leftarrow current.Time$
      if $policy$ = «FIFO»
         $queues[0] \leftarrow add(1)$
      else
         $queues[1] \leftarrow add(1)$
      end if
  end for
  switch $policy$
      case «FIFO»
          FIFO()
      end case
      case MqRR
          MqRR( ))
      end case
      case «qPrior»
          qPrior( ))
      end case
  end switch
  $count \leftarrow count + 1$
end while
if $messProc > 0$
   $throughput \leftarrow (messProc/(endTime[messProc] - startTime[1]))$
end if
 $Record\ Archive(simulationFile)$
```

Program 12.1: *Main algorithm.*

```
Input: Vector containing identification of tasks : $idTask[\,]$
Input: Vector containing parallel tasks : $parallelTask[\,]$
Input: Vector containing the execution time of the tasks : $timeExec[\,]$
Input: Vector containing the operations of the tasks: $operTask[\,]$
Input: Vector containing the next tasks : $nextTask[\,]$
Input: Vector containing last tasks: $lastTask[\,]$
    $numTasks \leftarrow idTask.length$
```

Program 12.2: *Profile algorithm.*

```
Input: Queue of tasks: $queues[0]$
Input: Maximum duration of the simulation: $maxDuration$
Input: Time start of the simulation: $start$
    $totSize \leftarrow queues[0].size$
    while $totSize > 0$ and $duration < maxDuration)$
        if $queues[i] \neq \emptyset$
            $Allocation\ (queues[0], totSize)$
        end if
        $duration \leftarrow$ current.Time $-start$
        $totSize \leftarrow queues[0].size$
    end while
```

Program 12.3: *FIFO algorithm.*

```
Input: Queues of tasks: $queues[\,]$
Input: Maximum duration of the simulation: $maxDuration$
Input: Time start of the simulation: $start$
Input: Total number of tasks: $numTasks$
Input: Number of tasks performed at a time (preemption): $preemptTask$
$totSize \leftarrow 1$
$preeemp \leftarrow preemptTask$
 while $totsize > 0$ and $duration < maxDuration)$
 for $[i] = 1$ to $numTasks$
       if $queues[i] \neq \emptyset$
         if $(preempt = 0)$ or $(queues[i].size < preempt)$
             $preempt \leftarrow queues[i].size$
         else
             $preempt \leftarrow preemptTask$
         end if
     $Allocation\ (queues[i], preempt)$
  end if
 end for
   $totSize \leftarrow 0$
  for $[i] = 1$ to $numTasks$
       $totSize \leftarrow totSize + queues[i].size$
  end for
   $duration \leftarrow$ current.Time $-start$
end while
```

Program 12.4: *MqRR algorithm.*

Input: Queue of tasks: $queues[i]$
Input: Maximum duration of the simulation: $maxDuration$
Input: Maximum number of messages: $maxMessages$
Input: Number of tasks performed at a time (preemption): $preempt$
Input: Vector containing last tasks: $lastTask[\,]$

```
preeemp ← preemptTask
Creates elastic pool of threads          for [j] = 1 to preempt
  if duration < maxDuration)
     task ← queues.head
    if task ≠ null
       Submits Operation(task) to a pool of threads
      for [j] = 0 to lastTask[ ].length
        if task ≠ lastTask[j]
          lastTask = 1
        end if
      end for
      if lastTask = 0
         Route (task)
      else
         endTime[] ← current.Time
      end if
       Removes task of the queue[i]
       Shutdown pool of threads
    end if
      if  queue[i].size < preempt
           preempt ← queue[i].size
      else
           preempt ← preemptTask
      end if
 else
    i ← preempt + 1
  end if
  duration ← current.Time − start
  if  maxMessages>startTime.size()
     Receive Add()
  else
     duration ← maxDuration
  end if
end for
```

Program 12.5: *Allocation algorithm.*

```
Input: Task : task
Input: Vector containing next tasks : nextTask[ ]
Input: Vector containing operation tasks: operTask[ ]
    operLeng ← operTask[task].length
    switch operLeng
      case 0
        nextTask ← nextTask[task][1]
        Send (nextTask)
      end case
      case 1 or 2
        if operLeng = 2
          operRad = random operTask[task][ ]
        else
          operRad = operTask[task]
        end if

        if operRad = or
          nextTask = randomnextTask[task][1]
          Send (nextTask)
        else
          if operRad = and
            for [i] = 1 to nextTask[task].length
              nextTask = nextTask[task][i]
              Send (nextTask)
            end for
          end if
        end if
      end case
    end switch
```

Program 12.6: *Route algorithm.*

```
Input: Task : task
Input: Vector containing execution time tasks : timeExec[ ]
  time ← random timeExec[task][ ]
  Waits time
```

Program 12.7: *Operation algorithm.*

```
Input: Task : $task$
Input: Vector containing parallel tasks : $parallelTask[\,]$
    if $policy$ = «FIFO»
        $queues[0] \leftarrow add(task)$
    else
        if $policy$ = «MqRR» or $policy$ = «qPrior»
            if $parallelTask[task] \neq 0$
                $parallel = parallelTask[task]$
                $queues[parallel] \leftarrow add(task)$
            else
                $queues[task] \leftarrow add(task)$
            end if
        end if
    end if
```

Program 12.8: *Send algorithm.*

```
for[i] = 1 to $numInputTask$
    $startTime[] \leftarrow current.Time$
  if $policy$ = «FIFO»
    $queues[0] \leftarrow add(1)$
  else
    if $policy$ = «MqRR» or $policy$ = «qPrior»
        $queues[1] \leftarrow add(1)$
    end if
  end if
end for
```

Program 12.9: *Receive algorithm.*

12.3 Summary

In this chapter, we have described the tool that we have implemented, called Integration Process Simulator. This tool allows the simulation of the execution of integration processes, performing their task scheduling by several heuristics. Its goal ist evaluate the performance of these heuristics. For this, IPS provides performance metrics, such as: such as makespan, throughput, number of processed messages, and the number of remained messages. Currently, our simulator endows the heuristics: FIFO, MqRR, and qPrior, but others can be incorporated. Its source code is compose of ten algorithms.

Chapter 13

Experiments

In a modern and innovative society, where advancements are plentiful, and communication is instantaneous, science and technology are a part of everyday life.

Julie Payette, Canadian Politician

Simulation-approach allows the evaluation of traditional and novel heuristics and measurement of their impact on the performance of the execution of integration processes. In this chapter, we simulate the execution of integration processes to validate our heuristic. We relate two experiments that aim to evaluate the performance of executions of integration processes. The first experiment verifies the impact of the preemption in performance of executions using qPrior and uses the PSO meta-heuristic to find the near-optimal preemption. The second experiment verifies the impact of the workload on the performance of executions using FIFO and qPrior.

13.1 Experimental protocol

The experiments followed a protocol based on Jedlitschka and Pfahl [95], Wohlin et al. [198], and Basili et al. [17], with procedures for controlled experiments in the field of software engineering and its steps are «Definition», «Planning», «Execution», and «Results». «Definition» is the step of main decisions regarding the experiment. «Planning» is the step to the organisation of elements needed to experiment. «Execution» is the step that detail procedures of the experiment. «Results» is the step of present and analysis of the results of the experiment. These steps and their respective activities are shown in Figure 13.1 and, in following, detail them.

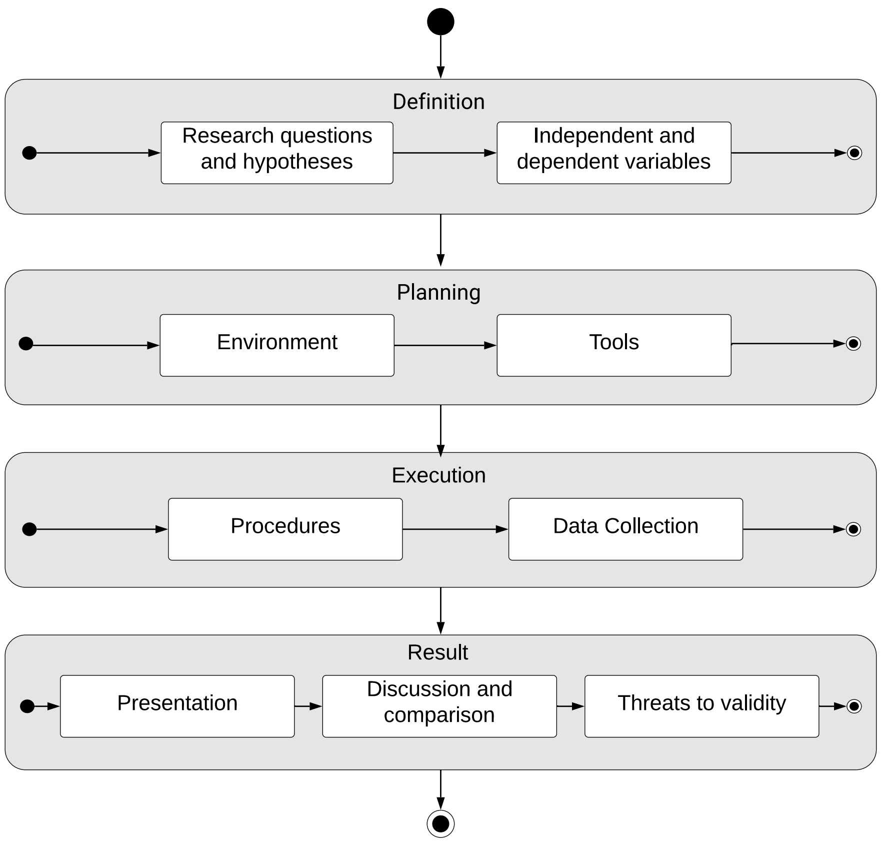


Figure 13.1: *Experiment protocol.*

Definition

- Research questions and hypotheses. Indication of the null hypotheses that are going to be confirmed or refused by the experiments.
- Independent and dependent variables. Indication of the variables to be measured and which will allow further comparison.

Planning

- Environment. Presentation of technical information about the hardware in which the experiment is performed.
- Tools. Presentation of technical information about software that support the conducting of the experiment.

Execution

- Procedure. Description of the scenarios of the experiment and statistical procedure for analysis of results are described.
- Data collection. Indication of how data will be collected for the variables in the experiments.

Results

- Presentation. Tables and charts show the results of the metrics collected in the experiments.
- Discussion and comparison. Argumentation regarding the results to respond to the research question and confirm or refute the hypotheses.
- Threats to validity. Description and evaluation of the factors that could influence the results of the experiment and the strategies to mitigate these threats.

13.2 Preemption evaluation

In this experiment, we compared the performance of executions of an integration process using the heuristic qPrior, varying the preemption. The goal is to verify if the preemption impacts on the performance of executions and if the PSO meta-heuristic found the near-optimal preemption for qPrior.

Research question and hypothesis

This experiment aims to answer the following research question:

RQ: Does the performance of the executions of integration processes improve when the qPrior heuristic uses an optimal preemption?

Our hypothesis to this research question is that:

H: The qPrior heuristic improves the performance of the executions of integration processes when it uses an optimal preemption since, the PSO algorithm can find the optimal or near-optimal, preemption.

Variables

The independent variables controlled in the experiment are:

Integration process. The conceptual model of the integration process taken as input. The model tested for this variable was case study 1.

Workload. The number of inbound messages. The value tested for this variable was 2,000,000 msg.

Initial workload. The initial number of inbound messages. The value tested for this variable was 1,000 msg.

Rate of inbound messages. The number of inbound messages added periodically to the integration process. The value tested for this variable was 1,000 msg.

Preemption. Number of tasks executed at each time queue checking. The values tested for this variable were 100, 500, 1,000, 10,000, 25,000, 50,000, 75,000, 100,000, 500,000, 1,000,000, 1,500,000, and 2,000,000 tasks.

The dependent variable measured in the experiment is:

Throughput. The number of processed messages by time unit.

Environment and supporting tools

We carried out the experiments on a machine equipped with 16 processors Intel Xeon CPU E5-4610 V4, 1.8 GHz, 32GB of RAM, and operating system Windows Server 2016 Datacenter 64-bits. We used the Java programming language, version 8.0 update 152, to implemented and execute the IPS simulator. We used the Genes [38] software, version 2015.5.0, to process the descriptive statistics, ANOVA and Scott & Knoot tests for the performance metrics used in this study.

Execution and data collection

The experiment was conducted using the IPS simulator, which simulates the execution of the case studies aforementioned. The simulation starts with a workload of 1,000 msg and receives 1,000 msg every 100 executions of tasks. We set the maximal total workload to 2,000,000 msg and the maximal time of duration for the simulation to 60s. Thus, the simulator stops the generation of messages when it reaches this maximal number of messages and stops running after 60s. Then, the simulator stores the preemption and throughput in a text file. After we handled and statistically analysed this information. We tested the execution using 12 different size for preemption. For each one of them, we repeated 25 times the execution of the qPrior heuristic, resulting in 300 scenarios, summarised in Table 13.1.

Integration Process	case study 1	1
Total workload	2,000,000 msg	1
Rate of inbound messages	1,000 msg	1
Initial workload	1,000 msg	1
Preemption	100, 500, 1,000, 10,000, 25,000, 50,000, 75,000, 100,000, 500,000, 1,000,000, 1,500,000, and 2,000,000 tasks	12
Repetitions	1...25	25
Scenarios	1 x 1 x 1 x 1 x 12 x 25	**300**

Table 13.1: *Scenarios for preemption evaluation.*

We tested the PSO algorithm with the range for search space [25000:75000]. For the case study 1, the best throughput was 30528,985,783 msg/s, found the near-optimal preemption equals to 50000 tasks, and the execution time of PSO algorithm took 7203 seconds. The output of the PSO algorithm is shown in Table 13.2.

Results

The average throughputs obtained in the 25 repetitions of the simulation for every value of the preemption are shown in Figure 13.2, where the x-axis represents preemption, and the y-axis represents the average throughput in messages per seconds (msg/s). In this figure, we outline the values in

```
Test Function                    :    qPriorPSO
Number of particles              :    20
Iterations                       :    100
Global Best Position             :    [50000,0, 55000,0]
Global Best value                :    0.000032755755697
Preemption                       :    55000,0
Throughput ( msg/s)              :    30528.985
Duration qPrior(s)               :    60.000
f(25000, 75000) : 0.000032755755697


BUILD SUCCESSFUL (total time: 2,001 minutes 3 seconds))
```

Table 13.2: *Output of PSO algorithm.*

which occurs the maximal throughput: 32,672 msg/s using a preemption of 50,000 tasks and 32,659 msg/s using a preemption of 1,500,000 tasks.

The analysis of throughput variance of execution of the case study 1 under a workload of 2,000,000 msg is shown in Table 13.3. The average square of the throughput was 3,374,239 for the preemption and 163,458 for error. The overall average was equal to 32,148 msg, and the coefficient of variation was 1.25%. The Scott & Knott test of the throughput, with an error level of 5%, is present in Tables 13.4. First column represent the preemption and the average of the throughput is represented in the second column. There were four groups: «a», «b», «c» and «d». Group «a» refers to the preemption with the

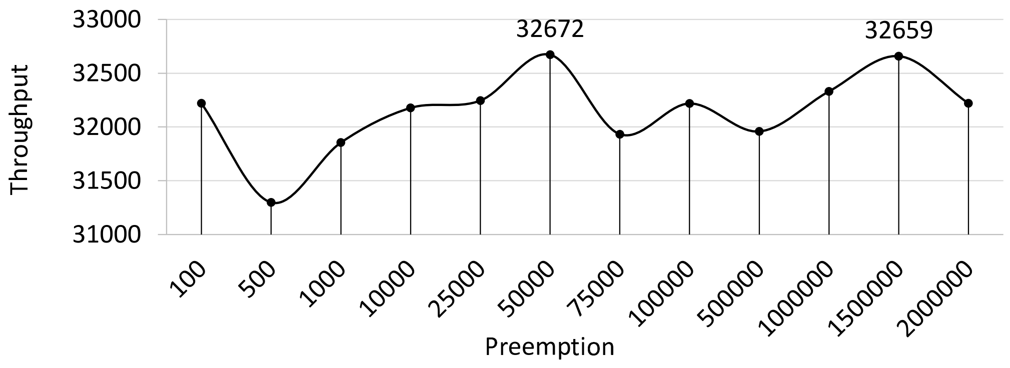


Figure 13.2: *Average throughput regarding preemption.*

Sources of variation	Degree of freedom	Average square
Preemption	11	3374239 †
Error	188	163458
Total	199	
Overall average		32,148
Coefficient of variation (%)		1.25

† *significant statistical by Fisher-Snedecor's - Probability and error level of 5%.*

Table 13.3: *Variance analysis of the throughput for preemption evaluation.*

highest average of the throughput, group «b» refers to the preemption with the second-highest, group «c» refers to the preemption with the third-highest, and group «d» refers to the preemption with the lowest average of the throughput. The preemptions of 50,000 and 1,500,000 tasks were in group «a» with the highest average of of throughput. The preemptions of 100, 10,000, 25,000, 100,000, 1,000,000, and 2,000,000 tasks were in group «b». The preemptions of 1,000, 75,000, and 500,000 tasks were in group «c». The preemption of 500 tasks was in group «d» with the lowest average of of throughput.

Preemption	**Average square**
	Throughput
50,000	32,671 «a»
1,500,000	32,658 «a»
100	32,219 «b»
10,000	32.177 «b»
25,000	32.245 «b»
100,000	32.218 «b»
1,000,000	32.331 «b»
2,000,000	32.219 «b»
1,000	31.855 «c»
75,000	31.932 «c»
500,000	31.958 «c»
500	31.298 «d»

Error level of 5% by the Scott & Knoot model.

Table 13.4: *Scott & Knott test for preemption.*

Discussion

In the experiment with the simulations of execution of the integration processes confirmed the value for near-optimal preemption found by the PSO algorithm. The best average throughput was 32,671 msg/s using a preemption of 50,000 tasks. The worst case was the throughput of 31,298 msg/ms, using a preemption of 500 tasks. A parable of concave down, in the interval between 25,000 and 75,000 msg, represents the behaviour of the throughput as a function of the preemption. This parable confirms that there is a preemption in which the throughput is maximal. The PSO algorithm found the near-optimal preemption, but its response time was 2,001 minutes, far superior to the average execution time of the simulation of the integration process (60s). So, this meta-heuristic must be used as a preliminary method, in cases of high workloads, in which the interval of values for preemption is larger since choosing the proper preemption is a challenge.

The use of different preemption generates a significant difference in the average of the throughput, cf. Table 13.3, so the search for a proper preemption is justified. The low values for the coefficients of variation indicate the adequacy and reliability of the experiment. In the Scott & Knott averages comparison test, there were three different groups of throughputs, cf. Table 13.4. A statistical difference between the three groups of preemption was found, but there was no difference between the preemptions of the same group. In group «a», which contain the best averages of throughput, it is possible to opt by any preemption to obtain the same statistic result for the throughput.

The performance of the qPrior was also evaluated using different preemption, including those found by PSO algorithm. First, we verified if there was a significant difference in the performance of the qPrior heuristic, using different preemptions. The performance metric used was the throughput, and the statistic test was the ANOVA. Then, we use Scott & Knoot test to group the similar ranges, in which there was no statistic difference between the use of the ranges belonged to the same group.

Regarding the research question:

RQ: The performance evaluated by the throughput of the execution of a real-world integration process under high workloads, improved by the use of the preemption found by PSO algorithm in the task scheduling carried out by qPrior heuristic.

By the experiment, we confirm our hypothesis:

H: The PSO algorithm found out a near-optimal preemption for the task scheduling using the qPrior heuristic, resulting in higher throughput of the execution for the integration processes under high workload than using other random values for the preemption.

13.3 Workload evaluation

In this experiment, we compared the performance of executions of the integration processses by the heuristics FIFO and qPrior, varying the workload. The goal is to verify the impact for workload in performance of executions.

Research Question and Hypothesis

This experiment aims to answer the following research question:

RQ: What is the behaviour of the performance of the executions of integration processes using qPrior when the workload increases?

Our hypothesis to this research question is that:

H: The performance of the executions of integration processes is better when using qPrior than when using FIFO when the workload increases.

Variables

The independent variables controlled in the execution of the algorithm are:

Heuristic. The heuristic used to task scheduling. The heuristics tested were: FIFO and qPrior.

Integration process. The conceptual model of the integration process. The model tested were: case study 1, case study 2, case study 3, and case study 4.

Workload. The number of inbound messages. The value tested for this variable were 1,000, 10,000, 100,000, 500,000, 1,000,000, 1,500,000, 2,000,000, and 2,500,000 msg.

Initial workload. The initial number of inbound messages. The value tested for this variable was 1,000 msg.

Rate of inbound messages. The number of inbound messages added periodically to the integration process. The value tested for this variable was 1,000 msg.

Preemption. The number of task executed at each time queue checking. The values tested for this variable was 50000 tasks.

The dependent variable measured in the execution of the algorithm is:

Processed messages. This variable corresponds to the number of inbound messages that are entirely processed by the integration process.

Remained messages. This variable corresponds to the number of inbound messages that are not processed by the integration process.

Throughput. The number of processed messages by time unit.

Makespan. The elapse time between the start time of the first message that entered and the end time of the last message that leaves the integration process.

Environment and supporting tools

We carried out the experiments on a machine equipped with 16 processors Intel Xeon CPU E5-4610 V4, 1.8 GHz, 32GB of RAM, and operating system Windows Server 2016 Datacenter 64-bits. We used the Java programming language, version 8.0 update 152, to implemented and execute the IPS simulator. We used the Genes [38] software, version 2015.5.0, to process the descriptive statistics, ANOVA and Scott & Knoot tests for the performance metrics used in this study.

Execution and data collection

The experiment was conducted using the IPS simulator, which simulates the execution of the case studies aforementioned. The simulation starts with a workload of 1,000 msg and receives 1,000 inbound messages every 100 executions of tasks. The execution time of each task varies within an interval, in seconds, according to the profile of the integration process. For the

Heuristics	FIFO and qPrior	2
Integration Processes	case study 1, case study 2, case study 3, and case study 4	4
Total workloads	1,000, 10,000, 100,000, 500,000, 1,000,000, 1,500,000, 2,000,000, and 2,500,000 msg	8
Rate of inbound messages	1,000 msg	1
Initial workload	1,000 msg	1
Preemption	50,000 tasks	1
Repetitions	1...25	25
Scenarios	2 x 4 x 8 x 1 x 1 x 1 x 25	**1600**

Table 13.5: *Scenarios for comparison of heuristics regarding workloads.*

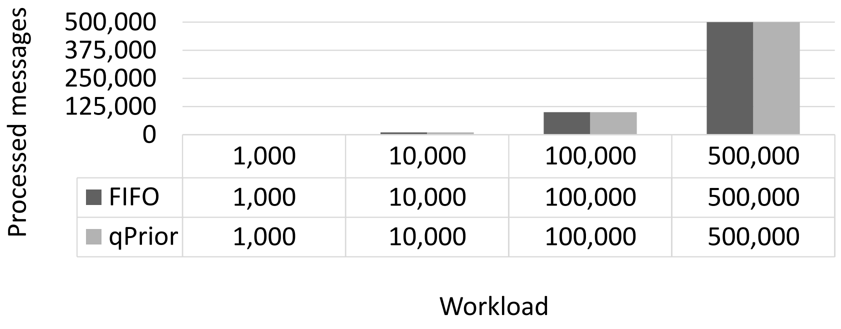
qPrior heuristic, the preemption used, number of tasks performed at a time, was set to 50,000 tasks. We set the maximal number of inbound messages and the maximal time of duration for the simulation to 60s. Thus, the simulator stops the generation of messages when it reaches this maximal number of messages and stops the running after 60s. Then, the simulator stores the workload, the number of processed messages, the number of remained message, the makespan, and the throughput in a text file. After, we handled and statistically analysed these metrics. We tested the execution using 8 workloads and 4 case studies. For each one of them, we repeated 25 times the execution using qPrior and 25 times using FIFO, resulting in 1600 scenarios, summarised in Table 13.5.

Results

The average values of dependent variables obtained in the 25 repetitions of the simulation for every value of the workload are shown in scatter charts. The x-axis represents the workload for every heuristic. The y-axis represents the metric measured. We consider 1,000, 10,000, 100,000, and 500,000 inbound messages as low workloads and 1,000,000, 1,500,000, 2,000,000, and 2,500,000 inbound messages as high workloads. Figures 13.3, 13.11, 13.19, and 13.27 represent the average processed with low workloads; and Figures 13.4, 13.12, 13.20, and 13.28 represent the average processed with high workloads. Figures 13.5, 13.13, 13.21, and 13.29 represent the average remained messages with low workload; and Figures 13.6, 13.14, 13.22, and 13.30 represent the average remained messages with high workloads. Figures 13.7, 13.15, 13.23, and 13.31 represent the average throughput with low workloads; and Figures 13.8, 13.16, 13.24, and 13.32 represent the average throughput with high workloads. Figures 13.9, 13.17, 13.25, and 13.33 represent the average makespan with low workloads; and Figures 13.10, 13.18, 13.26, and 13.34 represent the average makespan with high workloads.

Results for case study 1

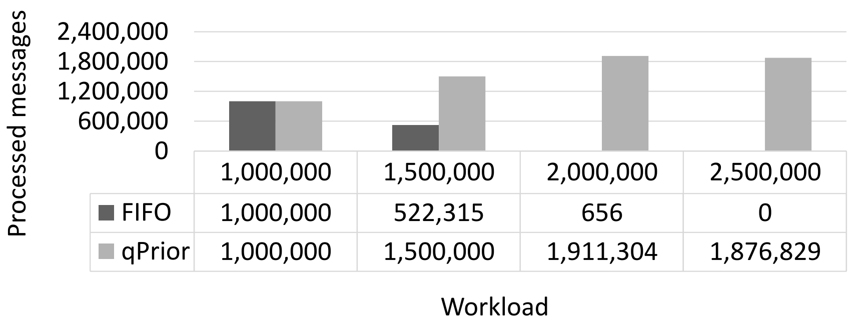
The average of processed messages in the execution of the case study 1 are shown in Figure 13.3 and Figure 13.4. In the simulations with low workloads: 1,000, 10,000, 100,000, and 500,000 msg, both heuristics, FIFO and qPrior, processed the inbound workloads entirely. In the simulation with a workload of 1,000,000 msg, the average of processed messages was equal to 1,000,000 msg with two heuristics: FIFO and qPrior. In the simulation with a workload of 1,000,000 msg, the average of processed messages was equal to 1,000,000 msg with two heuristics: FIFO and qPrior. In the simulation with 1,500,000 msg, the average of processed messages when using FIFO was equal to 522,315 msg, whereas using qPrior, all messages were successfully processed. In the simulation with 2,000,000 msg, 656 msg in average, were processed when using FIFO and 1,911,304 msg were processed when using qPrior. In the simulation with 2,500,000 msg, no message was processed when using FIFO in the elapsed time of the simulation; and, in case of qPrior, the average of processed messages was equal to 1,876,829 msg.



	1,000	10,000	100,000	500,000
■ FIFO	1,000	10,000	100,000	500,000
■ qPrior	1,000	10,000	100,000	500,000

Figure 13.3: *Average processed messages in case study - low workload.*

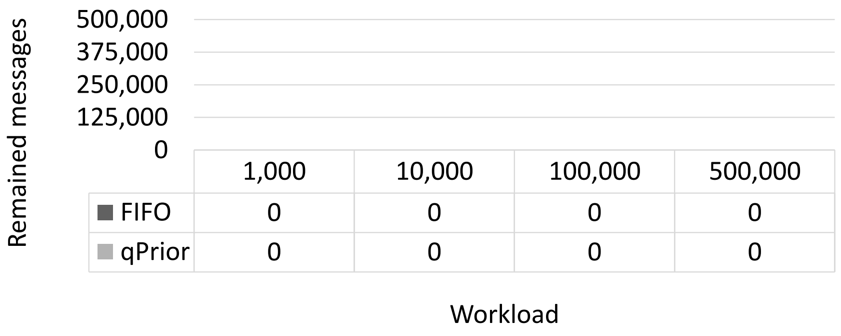
The average of remained messages in the execution of the case study 1 are shown in Figure 13.5 and Figure 13.6. In the simulations with low workloads: 1,000, 10,000, 100,000, and 500,000 msg, and with a high workload of 1,000,000 msg, there were no remained messages. In the simulation with 1,500,000 msg, the average of remained messages was equal to 1,359,833 msg



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	1,000,000	522,315	656	0
■ qPrior	1,000,000	1,500,000	1,911,304	1,876,829

Figure 13.4: *Average processed messages in case study 1 - high workload.*

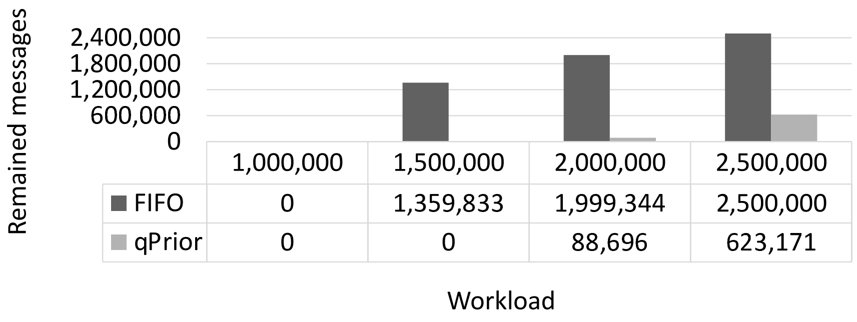
when using FIFO and no message when using qPrior. In the simulation with 2,000,000 msg, 1,999,344 msg remained on average when using FIFO and 88,696 msg when using qPrior. Then, in the simulation with 2,500,000 msg, all message remained when using FIFO and 623,171 msg remained on average when using qPrior.



	1,000	10,000	100,000	500,000
■ FIFO	0	0	0	0
■ qPrior	0	0	0	0

Figure 13.5: *Average remained messages in case study 1 - low workload.*

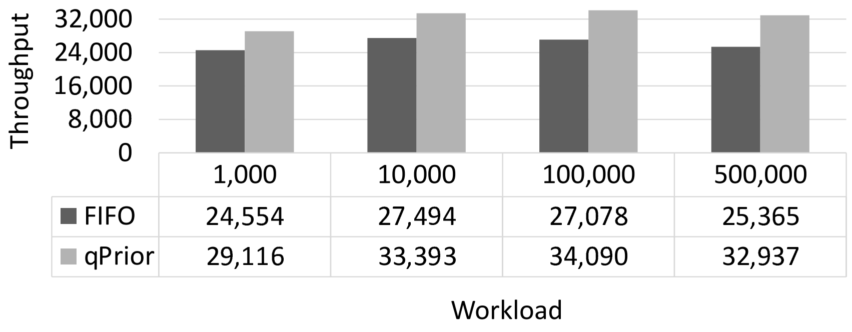
The throughput in the execution of the case study 1 are shown in Figure 13.7 and Figure 13.8. In the simulation with a workload of 1,000 msg, the average throughput achieved, in messages per seconds, was 24,554 msg/s when using FIFO and 29,116 msg/s when using qPrior. In the simulation with 10,000 msg, the average throughput achieved was 27,494 msg/s when using FIFO and 33,393 msg/s when using qPrior. In the simulation with 100,000 msg, the average throughput achieved was 27,078 msg/s when using FIFO and 34,090 msg/s when using qPrior. Then, in the simulation with 500,000 msg, the average throughput achieved was 25,365 msg/s when using FIFO and 32,937 msg/s when using qPrior.



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	0	1,359,833	1,999,344	2,500,000
▨ qPrior	0	0	88,696	623,171

Figure 13.6: *Average remained messages in case study 1 - high workload.*

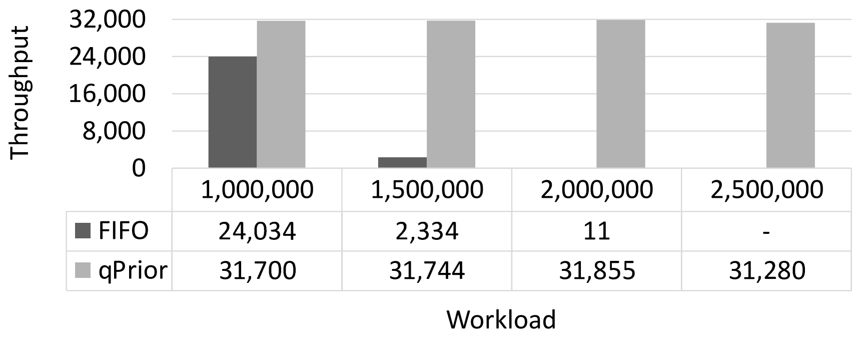
In the simulation with a workload of 1,000,000 msg, the average throughput achieved, in messages per seconds, was 24,034 msg/s when using FIFO and 31,700 msg/s when using qPrior. In the simulation with 1,500,000 msg, the average throughput achieved was 2,334 msg/s when using FIFO and 31,744 msg/s when using qPrior. In the simulation with 2,000,000 msg, the average throughput achieved was 11 msg/s when using FIFO and 31,855 msg/s when using qPrior. Then, in the simulation with 2,500,000 msg, the average throughput achieved was 0 msg/s when using FIFO and 31,280 msg/s when using qPrior.



	1,000	10,000	100,000	500,000
■ FIFO	24,554	27,494	27,078	25,365
▨ qPrior	29,116	33,393	34,090	32,937

Figure 13.7: *Average throughput in case study 1 - low workload.*

The makespan in the execution of the case study 1 are shown in Figure 13.9 and Figure 13.10. In the simulation with a workload of 1,000 msg, the average makespan achieved, in seconds, was 0.04s when using both FIFO and qPrior. In the simulation with 10,000 msg, the average makespan achieved was 0.36s when using FIFO and 0.30s when using qPrior. In the simulation with 100,000 msg, the average makespan achieved was 4s when using FIFO

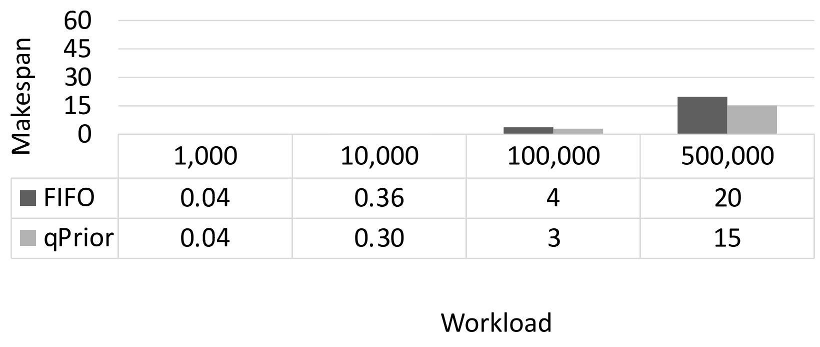


	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	24,034	2,334	11	-
■ qPrior	31,700	31,744	31,855	31,280

Figure 13.8: *Average throughput in case study 1 - high workload.*

and 3s when using qPrior. In the simulation with 500,000 msg, the average makespan achieved was 20s when using FIFO and 15s when using qPrior.

In the simulation with a workload of 1,000,000 msg, the average makespan achieved, in seconds, was 42s when using FIFO and 32s when using qPrior. In the simulation with 1,500,000 msg, the average makespan achieved was 60s when using FIFO and 47s when using qPrior. In the simulation with 2,000,000 msg, the average makespan achieved was 60s with both heuristics. In the simulation with 2,500,000 msg, the average makespan did not calculate when using FIFO because no message was processed in the elapsed time of the simulation; and in case of qPrior, the average makespan achieved was 60s.



	1,000	10,000	100,000	500,000
■ FIFO	0.04	0.36	4	20
■ qPrior	0.04	0.30	3	15

Figure 13.9: *Average makespan in case study 1 - low workload.*

Results for case study 2

In case study 2, there are three possible paths to an inbound message, cf. mentioned in 8.3. However, only 0.1% of the messages pass-through of the path, composed by tasks $\{t_{start}, t_1, t_2, t_{2end}\}$, which the path corresponds to

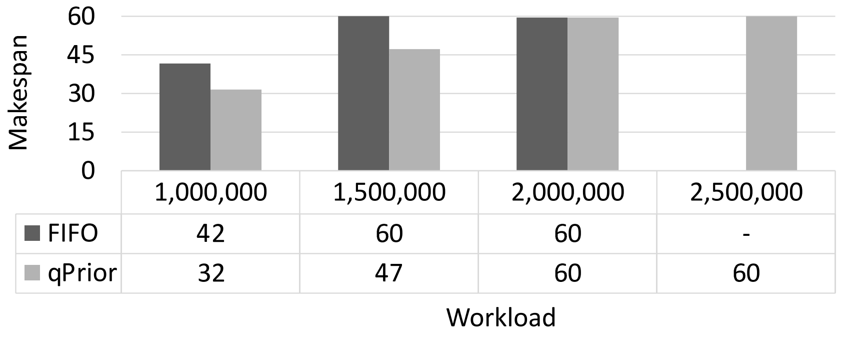


Figure 13.10: *Average makespan in case study 1 - high workload.*

«Invalid Items Log». So, we consider that this path has 0.1% of chance of occurs during a simulation. The average of processed messages in the execution of this case study are shown in Figure 13.11 and Figure 13.12. In the simulation with the workloads of 1,000, 10,000, 100,000, 500,000, 1,000,000 and 1,500,000 msg, all messages were successfully processed with both heuristics, FIFO and qPrior. In the simulation with 2,000,000 msg, 669,026 msg, in average, were processed when using FIFO and, when using qPrior, all messages were successfully processed. Then, in the simulation with 2,500,000 msg, 883,017 msg was processed when using FIFO and 2,331,044 msg when using qPrior.

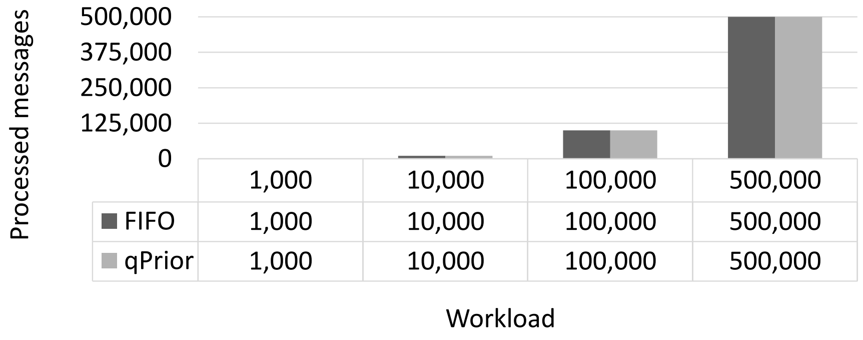
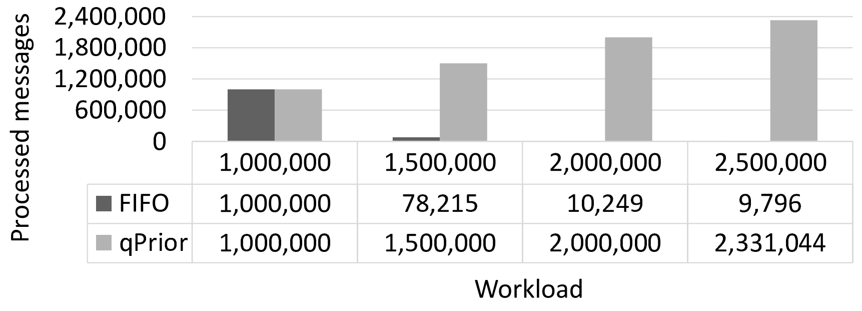


Figure 13.11: *Average processed messages in case study 2 - low workload.*

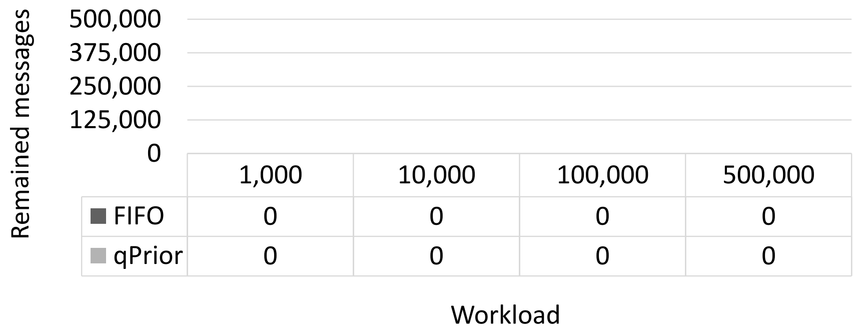
The average of remained messages in the execution of the case study 2 are shown in Figure 13.13 and Figure 13.14. In the simulation with the workloads of of 1,000, 10,000, 100,000, 500,000, 1,000,000 and 1,500,000 msg, there were no remained messages in neither case. In the simulation with 2,000,000 msg, 1,330,974 msg on average remained when using FIFO and no messages remained when using qPrior. Then, in the simulation with 2,500,000 msg,



	1,000,000	1,500,000	2,000,000	2,500,000
▪ FIFO	1,000,000	78,215	10,249	9,796
▫ qPrior	1,000,000	1,500,000	2,000,000	2,331,044

Figure 13.12: *Average processed messages in case study 2 - high workload.*

1,666,983 msg remained when using FIFO and 168,956 msg remained when using qPrior.

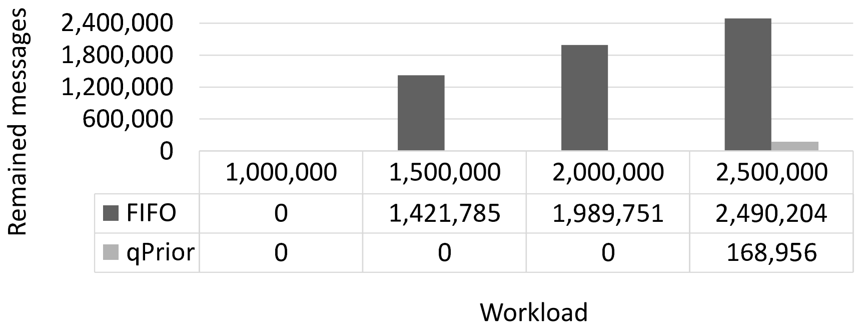


	1,000	10,000	100,000	500,000
▪ FIFO	0	0	0	0
▫ qPrior	0	0	0	0

Figure 13.13: *Average remained messages in case study 2 - low workload.*

The throughput in the execution of the case study 2 are shown in Figure 13.15 and Figure 13.16. In the simulation with a workload of 1,000 msg, the average throughput achieved was 30,078 msg/s when using FIFO and 34,236 msg/s when using qPrior. In the simulation with 10,000 msg, the average throughput achieved was 33,373 msg/s when using FIFO and 39,878 msg/s when using qPrior. In the simulation with 100,000 msg, the average throughput achieved was 33,005 msg/s when using FIFO and 41,354 msg/s when using qPrior. Then, in the simulation with 500,000 msg, the average throughput achieved was 30,226 msg/s when using FIFO and 39,023 msg/s when using qPrior.

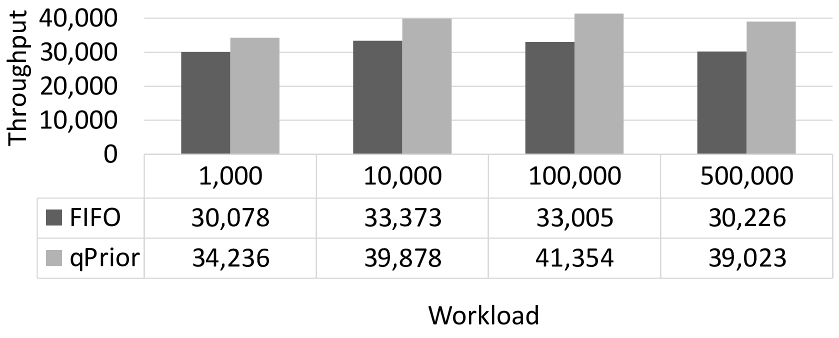
In the simulation with a workload of 1,000,000 msg, the average throughput achieved was 28,301 msg/s when using FIFO and 39,566 msg/s when using qPrior. In the simulation with 1,500,000 msg, the average throughput



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	0	1,421,785	1,989,751	2,490,204
■ qPrior	0	0	0	168,956

Figure 13.14: *Average remained messages in case study 2 - high workload.*

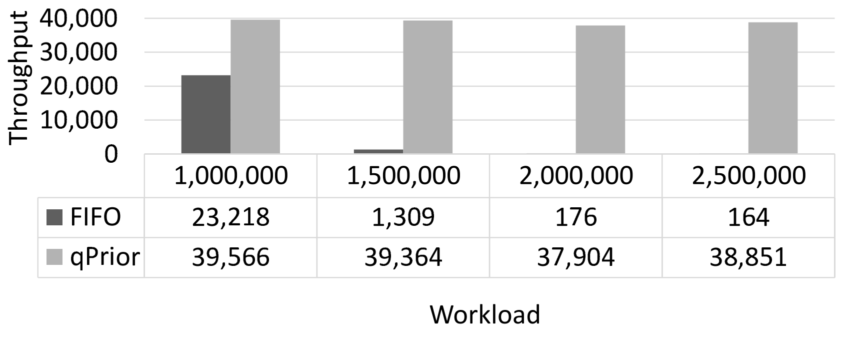
achieved was 27,594 msg/s when using FIFO and 39,364 msg/s when using qPrior. In the simulation with 2,000,000 msg, the average throughput achieved was 11,250 msg/s when using FIFO and 37,904 msg/s when using qPrior. Then, in the simulation with 2,500,000 msg, the average throughput achieved was 14,667 msg/s when using FIFO and 38,851 msg/s when using qPrior.



	1,000	10,000	100,000	500,000
■ FIFO	30,078	33,373	33,005	30,226
■ qPrior	34,236	39,878	41,354	39,023

Figure 13.15: *Average throughput in case study 2 - low workloads.*

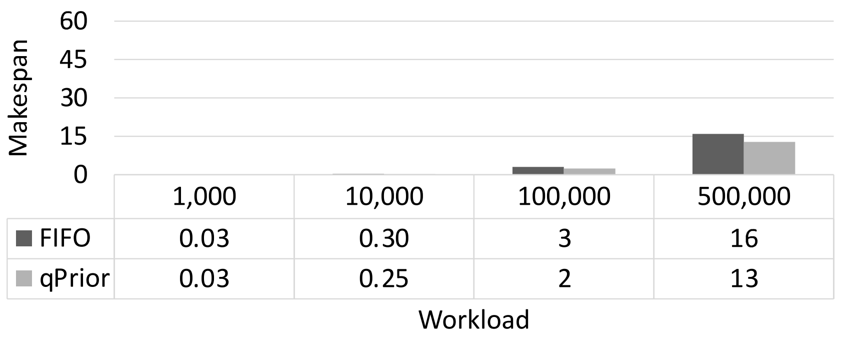
The makespan in the execution of the case study 2 are shown in Figure 13.17 and Figure 13.18. In the simulation with a workload of 1,000 msg, the average makespan achieved was 0.03s when using FIFO and 0.03s when using qPrior. In the simulation with 10,000 msg, the average makespan achieved was 0.30s when using FIFO and 0.25s when using qPrior. In the simulation with 100,000 msg, the average makespan achieved was 3s when using FIFO and 2s when using qPrior. Then, in the simulation with 500,000 msg, the average makespan was 16s when using FIFO and 13s when using qPrior.



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	23,218	1,309	176	164
■ qPrior	39,566	39,364	37,904	38,851

Figure 13.16: *Average throughput in case study 2 - high workloads.*

In the simulation with a workload of 1,000,000 msg, the average makespan achieved was 35s when using FIFO and 25s when using qPrior. In the simulation with 1,500,000 msg, the average makespan achieved was 54s when using FIFO and 38s when using qPrior. In the simulation with 2,000,000 msg, the average makespan achieved was 59s when using FIFO and 52s when using qPrior. Then, in the simulation with 2,500,000 msg, the average makespan was 57s when using FIFO and 60s when using qPrior.

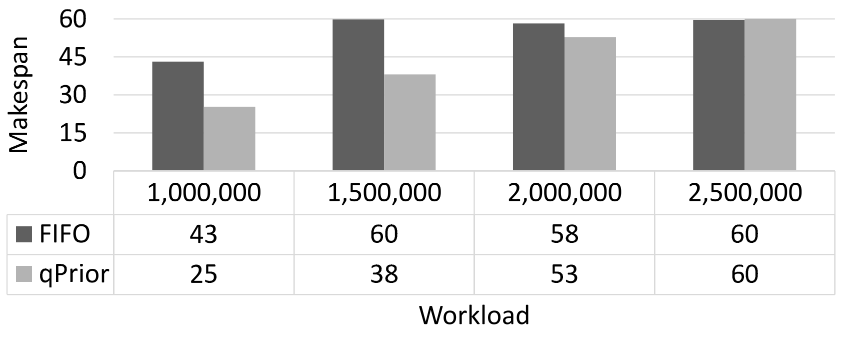


	1,000	10,000	100,000	500,000
■ FIFO	0.03	0.30	3	16
■ qPrior	0.03	0.25	2	13

Figure 13.17: *Average makespan in case study 2 - low workloads.*

Results for case study 3

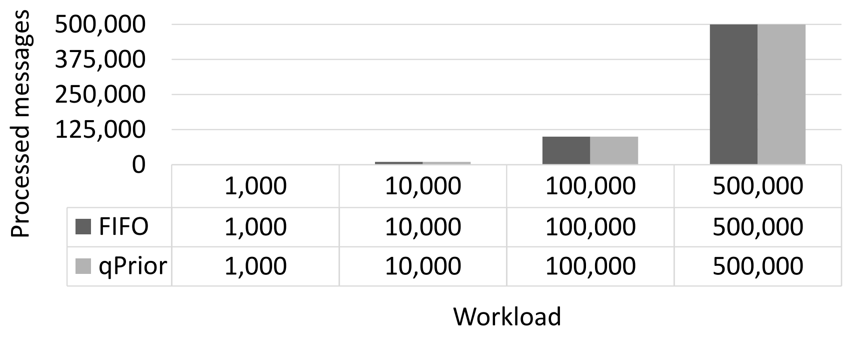
The average of processed messages in the execution of the case study 3 are shown in Figure 13.19 and Figure 13.20. In the simulation with a workload of 1,000, 10,000, 100,000, 500,000, and 1,000,000, all messages were successfully processed with both heuristics, FIFO and qPrior. In the simulation with 1,500,000 msg, 696,806 msg, in average, were processed when



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	43	60	58	60
▨ qPrior	25	38	53	60

Figure 13.18: *Average makespan in case study 2 - high workloads.*

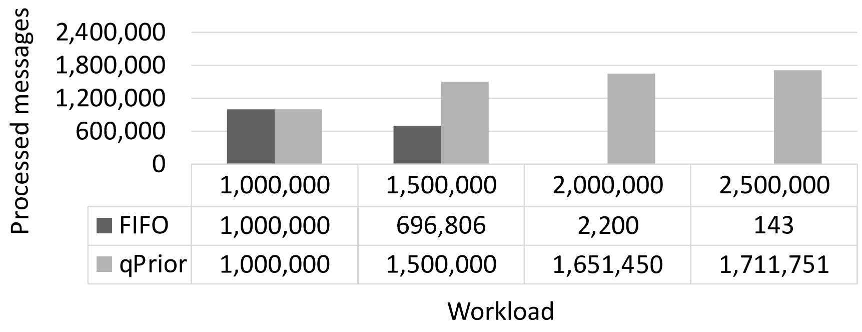
using FIFO and, when using qPrior, all messages were successfully processed. In the simulation with 2,000,000 msg, 2,200 msg was processed when using FIFO and 1,651,450 msg when using qPrior. Then, in the simulation with 2,500,000 msg, 143 msg was processed when using FIFO and 1,711,751 msg when using qPrior.



	1,000	10,000	100,000	500,000
■ FIFO	1,000	10,000	100,000	500,000
▨ qPrior	1,000	10,000	100,000	500,000

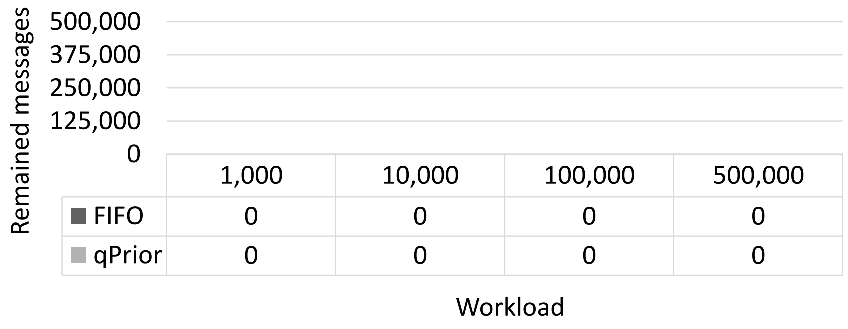
Figure 13.19: *Average processed messages in case study 3 - low workloads.*

The average of remained messages in the execution of the case study 3 are shown in Figure 13.22. In the simulation with a workload of of 1,000, 10,000, 100,000, 500,000, and 1,000,000 msg, there were no remained messages in both heuristics. In the simulation with 1,500,000 msg, 1,045,748 msg on average remained when using FIFO and no messages remained when using qPrior. In the simulation with 2,000,000 msg, 1,997,800 msg on average remained when using FIFO and 348,550 msg remained when using qPrior. Then, in the simulation with 2,500,000 msg, 2,499,857 msg remained when using FIFO and 788,249 msg remained when using qPrior.



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	1,000,000	696,806	2,200	143
■ qPrior	1,000,000	1,500,000	1,651,450	1,711,751

Figure 13.20: *Average processed messages in case study 3 - high workloads.*

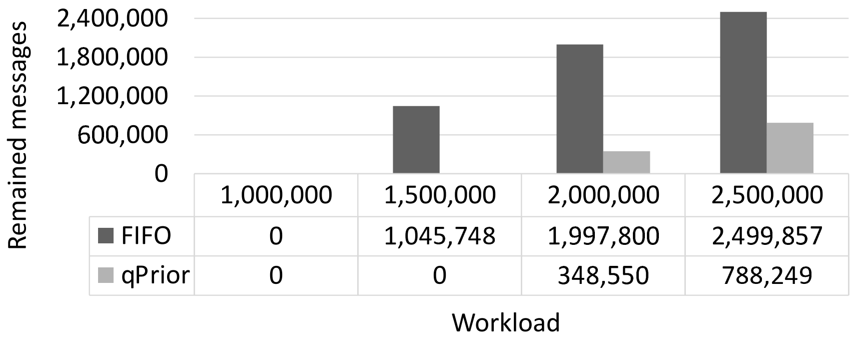


	1,000	10,000	100,000	500,000
■ FIFO	0	0	0	0
■ qPrior	0	0	0	0

Figure 13.21: *Average remained messages in case study 3 - low workloads.*

The throughput in the execution of the case study 3 are shown in Figure 13.23 and Figure 13.24. In the simulation with a workload of 1,000 msg, the average throughput achieved was 22,745 msg/s when using FIFO and 26,446 msg/s when using qPrior. In the simulation with 10,000 msg, the average throughput achieved was 24,895 msg/s when using FIFO and 30,453 msg/s when using qPrior. In the simulation with 100,000 msg, the average throughput achieved was 24,420 msg/s when using FIFO and 31,034 msg/s when using qPrior. Then, in the simulation with 500,000 msg, the average throughput achieved was 22,860 msg/s when using FIFO and 29,685 msg/s when using qPrior.

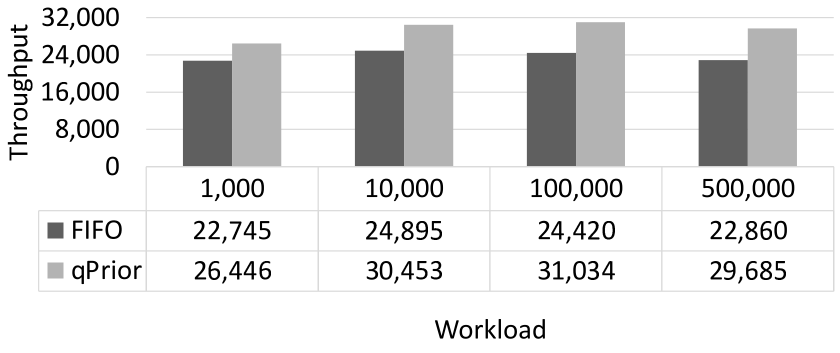
In the simulation with a workload of 1,000,000 msg, the average throughput achieved was 21,765 msg/s when using FIFO and 28,780 msg/s when using qPrior. In the simulation with 1,500,000 msg, the average throughput achieved was 7,573 msg/s when using FIFO and 28,880 msg/s when using qPrior. In the simulation with 2,000,000 msg, the average throughput achieved was 37 msg/s when using FIFO and 27,524 msg/s when using



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	0	1,045,748	1,997,800	2,499,857
■ qPrior	0	0	348,550	788,249

Figure 13.22: *Average remained messages in case study 3 - high workloads.*

qPrior. Then, in the simulation with 2,500,000 msg, the average throughput achieved was 2 msg/s when using FIFO and 28,529 msg/s when using qPrior.

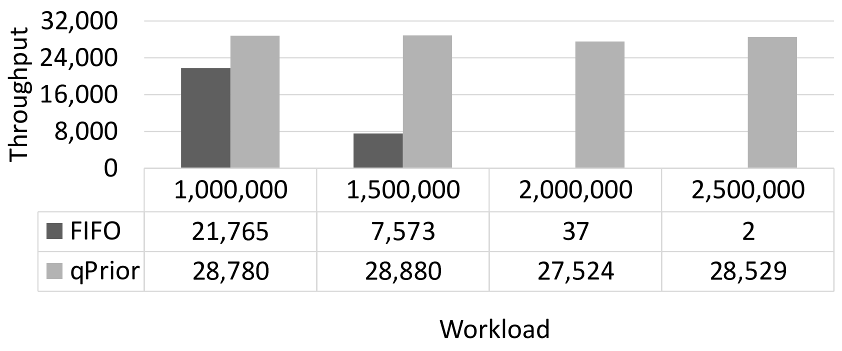


	1,000	10,000	100,000	500,000
■ FIFO	22,745	24,895	24,420	22,860
■ qPrior	26,446	30,453	31,034	29,685

Figure 13.23: *Average throughput in case study 3 - low workloads.*

The makespan in the execution of the case study 3 are shown in Figure 13.25 and Figure 13.26. In the simulation with a workload of 1,000 msg, the average makespan achieved was 0.05s when using FIFO and 0.04s when using qPrior. In the simulation with 10,000 msg, the average makespan achieved was 0.40s when using FIFO and 0.33s when using qPrior. In the simulation with 100,000 msg, the average makespan achieved was 4s when using FIFO and 3s when using qPrior. Then, in the simulation with 500,000 msg, the average makespan was 22s when using FIFO and 17s when using qPrior.

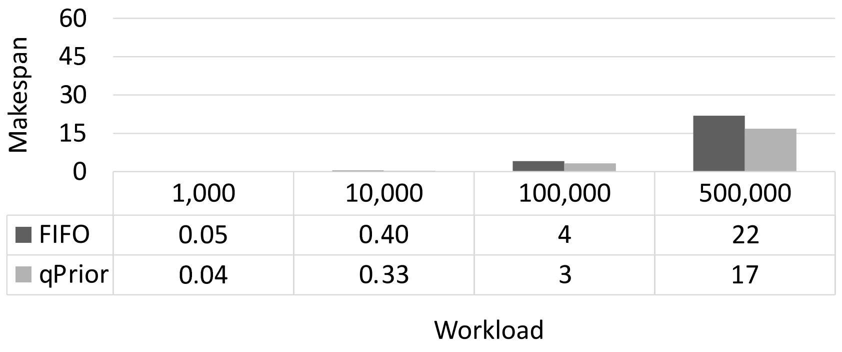
In the simulation with a workload of 1,000,000 msg, the average makespan achieved was 46s when using FIFO and 35s when using qPrior. In the simulation with 1,500,000 msg, the average makespan achieved was 60s when using FIFO and 52s when using qPrior. In the simulation with 2,000,000 msg, the



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	21,765	7,573	37	2
■ qPrior	28,780	28,880	27,524	28,529

Figure 13.24: *Average throughput in case study 3 - high workloads.*

average makespan achieved was 59s when using FIFO and 60s when using qPrior. Then, in the simulation with 2,500,000 msg, the average makespan was 58s when using FIFO and 60s when using qPrior.

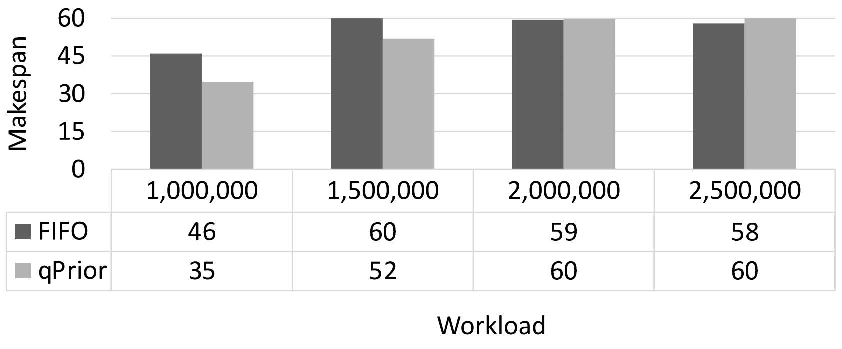


	1,000	10,000	100,000	500,000
■ FIFO	0.05	0.40	4	22
■ qPrior	0.04	0.33	3	17

Figure 13.25: *Average makespan in case study 3 - low workloads.*

Results for case study 4

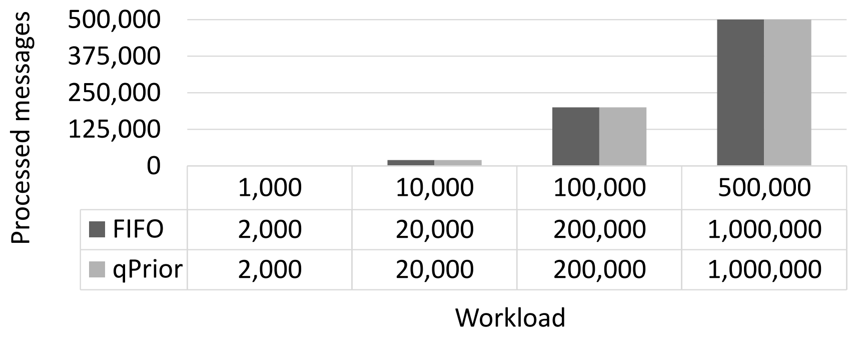
The case study 4 has two mandatory outputs, thus the number of outbound messages is higher than the number of inbound messages. The average of processed messages in the execution of this case study are shown in Figure 13.27 and Figure 13.28. In the simulation with a workload of 1,000, 10,000, 100,000, 500,000, and 1,000,000, all messages were successfully processed with both heuristics, FIFO and qPrior. In the simulation with 1,500,000 msg, 2,913,107 msg, in average, were processed when using FIFO and, when using qPrior, 3,000,000 msg were processed. In the simulation with 2,000,000 msg, 209,779 msg was processed when using FIFO and 4,000,000 msg when using qPrior. Then, in the simulation with 2,500,000 msg,



	1,000,000	1,500,000	2,000,000	2,500,000
▆ FIFO	46	60	59	58
▨ qPrior	35	52	60	60

Figure 13.26: *Average makespan in case study 3 - high workloads.*

3,640 msg was processed when using FIFO and 4,238,093 msg when using qPrior.

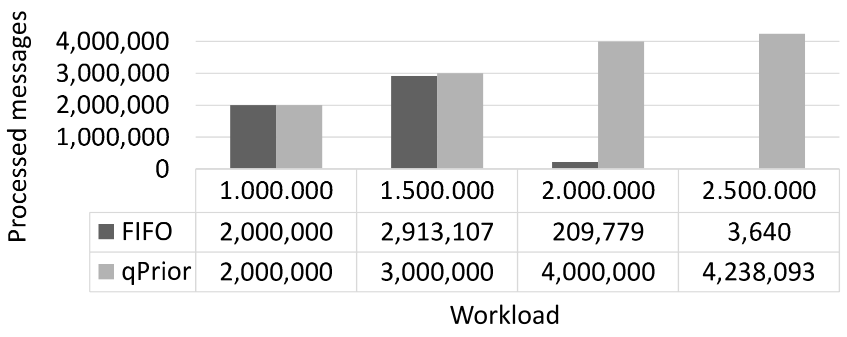


	1,000	10,000	100,000	500,000
▆ FIFO	2,000	20,000	200,000	1,000,000
▨ qPrior	2,000	20,000	200,000	1,000,000

Figure 13.27: *Average processed messages in case study 4 - low workloads.*

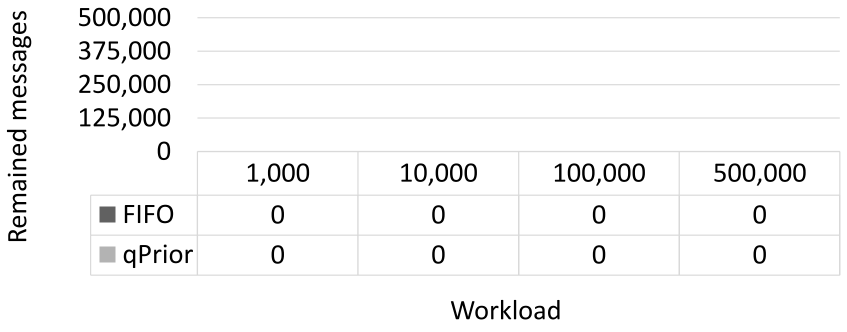
The average of remained messages in the execution of the case study 4 are shown in Figure 13.29 and Figure 13.30. In the simulation with a workload of 1,000, 10,000, 100,000, 500,000, and 1,000,000, there were no remained messages in neither case. In the simulation with 1,500,000 msg, 86,893 msg on average remained when using FIFO and no messages remained when using qPrior. In the simulation with 2,000,000 msg, 2,944,075 msg on average remained when using FIFO and no messages remained when using qPrior. Then, in the simulation with 2,500,000 msg, 2,516,476 msg remained when using FIFO and 380,954 msg remained when using qPrior.

The throughput in the execution of the case study 4 are shown in Figure 13.31 and Figure 13.32. In the simulation with a workload of 1,000 msg, the average throughput achieved was 56,098 msg/s when using FIFO and 64,619 msg/s when using qPrior. In the simulation with



	1.000.000	1.500.000	2.000.000	2.500.000
■ FIFO	2,000,000	2,913,107	209,779	3,640
■ qPrior	2,000,000	3,000,000	4,000,000	4,238,093

Figure 13.28: *Average processed messages in case study 4 - high workloads.*

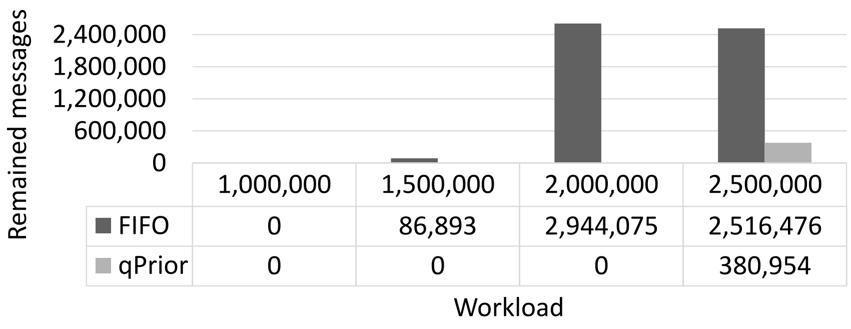


	1,000	10,000	100,000	500,000
■ FIFO	0	0	0	0
■ qPrior	0	0	0	0

Figure 13.29: *Average remained messages in case study 4 - low workloads.*

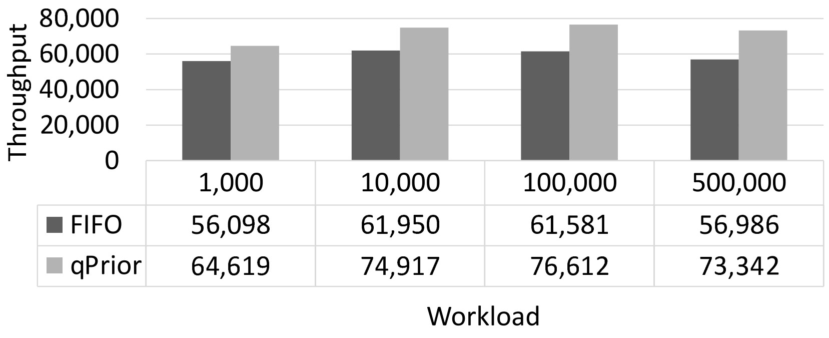
10,000 msg, the average throughput achieved was 61,950 msg/s when using FIFO and 74,917 msg/s when using qPrior. In the simulation with 100,000 msg, the average throughput achieved was 61,581 msg/s when using FIFO and 76,612 msg/s when using qPrior. Then, in the simulation with 500,000 msg, the average throughput achieved was 56,986 msg/s when using FIFO and 73,342 msg/s when using qPrior.

The throughput in the execution of the case study 4 is shown in Figure 13.32. In the simulation with a workload of 1,000,000 msg, the average throughput achieved was 49,937 msg/s when using FIFO and 72,950 msg/s when using qPrior. In the simulation with 1,500,000 msg, the average throughput achieved was 48,681 msg/s when using FIFO and 72,489 msg/s when using qPrior. In the simulation with 2,000,000 msg, the average throughput achieved was 3,511 msg/s when using FIFO and 72,491 msg/s when using qPrior. Then, in the simulation with 2,500,000 msg, the average throughput achieved was 62 msg/s when using FIFO and 70,635 msg/s when using qPrior.



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	0	86,893	2,944,075	2,516,476
■ qPrior	0	0	0	380,954

Figure 13.30: *Average remained messages in case study 4 - high workloads.*

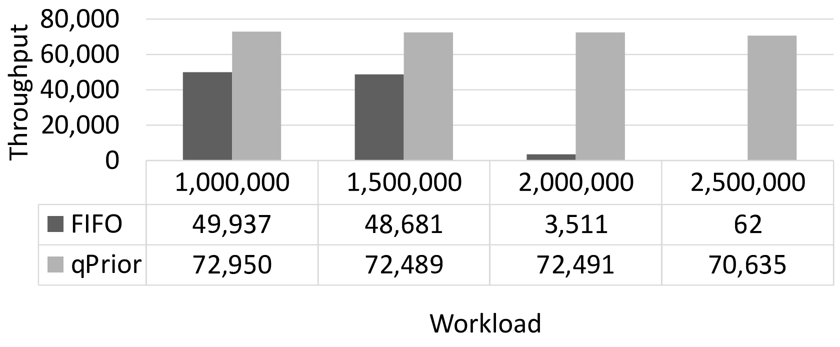


	1,000	10,000	100,000	500,000
■ FIFO	56,098	61,950	61,581	56,986
■ qPrior	64,619	74,917	76,612	73,342

Figure 13.31: *Average throughput in case study 4 - low workloads.*

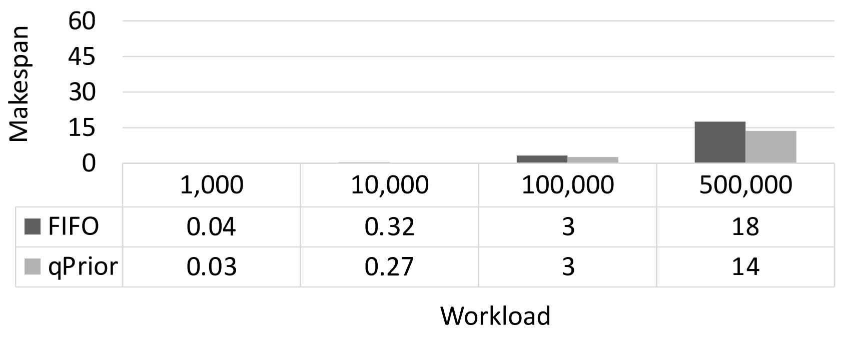
The makespan in the execution of the case study 4 are shown in Figure 13.33 and Figure 13.34. In the simulation with a workload of 1,000 msg, the average makespan achieved was 0.04s when using FIFO and 0.03s when using qPrior. In the simulation with 10,000 msg, the average makespan achieved was 0.32s when using FIFO and 0.27s when using qPrior. In the simulation with 100,000 msg, the average makespan achieved was 3s when using FIFO and 3s when using qPrior. Then, in the simulation with 500,000 msg, the average makespan was 18s when using FIFO and 14s when using qPrior.

In the simulation with a workload of 1,000,000 msg, the average makespan achieved was 40s when using FIFO and 27s when using qPrior. In the simulation with 1,500,000 msg, the average makespan achieved was 60s when using FIFO and 41s when using qPrior. In the simulation with 2,000,000 msg, the average makespan achieved was 60s when using FIFO and 55s when using qPrior. Then, in the simulation with 2,500,000 msg, the average makespan was 59s when using FIFO and 60s when using qPrior.



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	49,937	48,681	3,511	62
■ qPrior	72,950	72,489	72,491	70,635

Figure 13.32: *Average throughput in case study 4 - high workloads.*

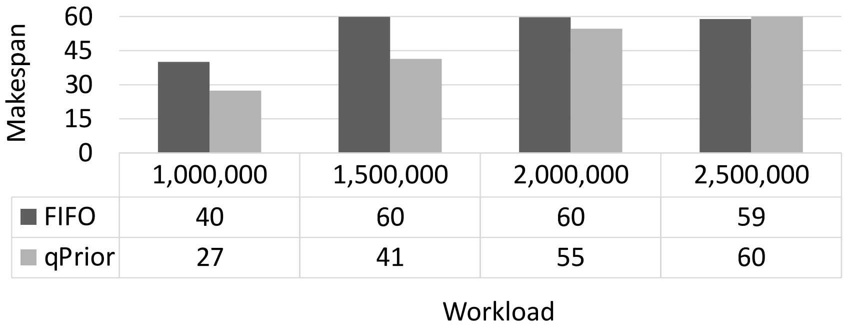


	1,000	10,000	100,000	500,000
■ FIFO	0.04	0.32	3	18
■ qPrior	0.03	0.27	3	14

Figure 13.33: *Average makespan in case study 4 - low workloads.*

Statistical analysis

We used the ANOVA test to verify the influence of random factors in the measurements of the dependent variables. The ANOVA test was applied for the scenarios in which the averages of the dependent variables, for both heuristics, were different of zero. The workload of 2,500,000 msg for case study 3 was ignored because the throughput in this case was near zero. The selected scenarios are summarised in Table 13.6. The analysis of variance for every value of the workload is shown in Tables 13.7, 13.9, 13.11, and 13.13. Since there were statistical differences between the results with different scenarios, we follow the comparison of averages of the dependent variables by Scott & Knott test. The results of the Scott & Knoot test are shown in Tables 13.8, 13.10, 13.12, and 13.14. In these tables, the heuristics are in the first column. For each dependent variable, there is a column for the average and a column for the group of Scott & Knott. Two groups were found:



	1,000,000	1,500,000	2,000,000	2,500,000
■ FIFO	40	60	60	59
■ qPrior	27	41	55	60

Figure 13.34: *Average makespan in case study 4 - high workloads.*

«a» and «b». Group «a» refers to the heuristic with the highest of the dependent variables, namely, average of processed messages, remained messages, throughput, and makespan and group «b» refers to the heuristic with the lowest average of these variables.

Integration Process	Workload
case study 1	2,000,000
case study 2	2,500,000
case study 3	2,000,000
case study 4	2,500,000

Table 13.6: *Scenarios for statistical tests.*

Statistical analysis for case study 1

The analysis of variance was applied for the workload of 2,000,000 msg, cf. Table 13.7. In this analysis of the the processed messages, the average square was 45,632,197,248,800 for the heuristics and 164,179,376 for error. The overall average was equal to 955,980 msg, and the coefficient of variation was 1.34%. In the analysis of variance of the remained messages, the average square was 45,632,197,248,800 for the heuristics and 164,179,376 for error. The overall average was equal to 1,044,020 msg, and the coefficient of variation was 1.22%. In the analysis of variance of the throughput, the average square was 12,675,555,787 for the heuristics and 45,605 for error. The overall average was equal to 15,933 msg/s, and the coefficient of variation was 1.34%. Then, in the analysis of variance of the makespan, the average square was 2.94 for the heuristics and 0.17 for error. The overall average was equal to 59.75s, and the coefficient of variation was 0.69%.

Sources of variation	**Degree of freedom**	**Average square**			
		Processed messages	Remained messages	Throughput	Makespan
Heuristics	1	45632197248800 †	45632197248800 †	12675555787 †	2.94 †
Error	48	164179376	164179376	45605	0.17
Total	49				
Overall average		955980	1044020	15933	59.75
Coefficient of variation (%)		1.34	1.22	1.34	0.69

† *significant statistical by Fisher-Snedecor's - Probability and error level of 5%.*

Table 13.7: *ANOVA test for case study 1.*

The results of the Scott & Knoot test for workload of 2,000,000 msg are shown in Table 13.8. Regarding processed messages, FIFO was in group «b» with the lowest average of 656 msg and qPrior in group «a» with the highest average of 1,911,304 msg. Regarding the remained messages, FIFO was in group «a» with the highest average of 1,999,344 msg and qPrior in group «b» with the highest average 88,696 msg. When analysing throughout, FIFO was in group «b» with the lowest average of 11 msg/s and qPrior in group «a» with the highest average of 31,855 msg/s. Regarding makespan, FIFO was in group «b» with the lowest average of 59.51s and qPrior in group «a» with the highest average of 60s.

Heuristic	**Average square**			
	Processed messages	Remained messages	Throughput	Makespan
FIFO	656 «b»	1999344 «a»	11 «b»	59.51 «b»
qPrior	1911304 «a»	88696 «b»	31855 «a»	60 «a»

Error level of 5% by the Scott & Knoot model.

Table 13.8: *Scott & Knott test for case study 1.*

Statistical analysis for case study 2

The analysis of variance was applied for the workload of 2,500,000 msg, cf. Table 13.9. In this analysis of the processed messages, the average square was 28,051,074,641,357 for the heuristics and 685,476,247 for error. The overall average was equal to 1,582,030 msg, and the coefficient of variation was 1.65%. In the analysis of variance of the remained messages, the average square was 2,8051,074,641,357 for the heuristics and 685,476,247 for error.

The overall average was equal to 917,969 msg, and the coefficient of variation was 2.85%. In the analysis of variance of the throughput, the average square was 7,310,834,073 for the heuristics and 275,672 for error. The overall average was equal to 26,758 msg/s, and the coefficient of variation was 1.96%. Then, in the analysis of variance of the makespan, the average square was 125 for the heuristics and 1.15 for error. The overall average was equal to 58s, and the coefficient of variation was 1.83%.

Sources of variation	**Degree of freedom**	**Average square**			
		Processed messages	Remained messages	Throughput	Makespan
Heuristics	1	28051074641357 †	28051074641357 †	7310834073 †	125 †
Error	48	685476247	685476247	275672	1.15
Total	49				
Overall average		1582030	917969	26758	58
Coefficient of variation (%)		1.65	2.85	1.96	1.83

† *significant statistical by Fisher-Snedecor's - Probability and error level of 5%.*

Table 13.9: *ANOVA test for case study 2.*

The results of the Scott & Knoot test for 2,500,000 msg are shown in Table 13.10. Regarding processed messages, FIFO was in group «b» with the lowest average of 833,016.9 msg and qPrior in group «a» with the highest average of 2,331,044.3 msg. Regarding the remained messages, FIFO was in group «a» with the highest average of 1,666,983.0 msg and qPrior in group «b» with the lowest average of 168,955.68 msg. When analysing throughout, FIFO was in group «b» with the lowest average of 14,666.7 msg/s and qPrior in group «a» with the highest average of 8,850.74 msg/s. Regarding makespan, FIFO was in group «b» with the lowest average of 56.83s and qPrior in group «a» with the highest average of 60s.

Heuristic	**Average square**			
	Processed messages	Remained messages	Throughput	Makespan
FIFO	833016.96 «b»	1666983.04 «a»	14666.72 «b»	56.83 «b»
qPrior	2331044.32 «a»	168955.68 «b»	38850.74 «a»	60 «a»

Error level of 5% by the Scott & Knoot model.

Table 13.10: *Scott & Knott test for case study 2.*

Statistical analysis for case study 3

The analysis of variance was applied for the workload of 2,000,000 msg, cf. Table 13.11. In this analysis of the processed messages, the average square was 34,000,312,934,250 for the heuristics and 101,107,444 for error. The overall average was equal to 826,824 msg, and the coefficient of variation was 1.21%. In the analysis of variance of the remained messages, the average square was 34,000,312,934,250 for the heuristics and 101,107,444 for error. The overall average was equal to 1,173,175 msg, and the coefficient of variation was 0.85%. In the analysis of variance of the throughput, the average square was 9,444,360,032 for the heuristics and 28,087 for error. The overall average was equal to 13,780 msg/s, and the coefficient of variation was 1.21%. Then, in the analysis of variance of the makespan, the average square was 3.88 for the heuristics and 0.34 for error. The overall average was equal to 59s, and the coefficient of variation was 0.98%.

Sources of variation	**Degree of freedom**	**Average square**			
		Processed messages	Remained messages	Throughput	Makespan
Heuristics	1	34000312934250 †	34000312934250 †	9444360032 †	3.88 †
Error	48	101107444	101107444	28087	0.34
Total	49				
Overall average		826824	1173175	13780	59
Coefficient of variation (%)		1.21	0.85	1.21	0.98

† *significant statistical by Fisher-Snedecor's - Probability and error level of 5%.*

Table 13.11: *ANOVA test for case study 3.*

The results of the Scott & Knoot test for workload of 2,000,000 msg are shown in Table 13.12. Regarding processed messages, FIFO was in group «b» with the lowest average of 2,200 msg and qPrior in group «a» with the highest average of 1,651,449.80 msg. Regarding the remained messages, FIFO was in group «a» with the highest average of 1,997,800 msg and qPrior in group «b» with the lowest average of 348,550.16 msg. When analysing throughout, FIFO was in group «b» with the lowest average of 36.92 msg/s and qPrior in group «a» with the highest average of 27,524.17 msg/s. Regarding makespan, FIFO was in group «b» with the lowest average of 59.44s and qPrior in group «a» with the highest average of 60s.

Heuristic	Average square			
	Processed messages	Remained messages	Throughput	Makespan
FIFO	2200.00 «b»	1997800.00 «a»	36.92 «b»	59.44 «a»
qPrior	1651449.84 «a»	348550.16 «b»	27524.17 «a»	60 «b»

Error level of 5% by the Scott & Knoot model.

Table 13.12: *Scott & Knott test for case study 3.*

Statistical analysis for case study 4

The analysis of variance was applied for the workload of 2,500,000 msg, cf. Table 13.13. In the analysis of variance of the processed messages, the average square was 224,132,428,021,831 for the heuristics and 2,091,387,333 for error. The overall average was equal to 2,120,866 msg, and the coefficient of variation was 2.15%. In the analysis of variance of the remained messages, the average square was 57,005,699,011,272 for the heuristics and 544,808,966 for error. The overall average was equal to 1,448,714 msg, and the coefficient of variation was 1.61%. In the analysis of variance of the throughput, the average square was 62,257,273,316 for the heuristics and 580,932 for error. The overall average was equal to 35,348 mgs/s, and the coefficient of variation was 2.15%. Then, in the analysis of variance of the makespan, the average square was 15.09 for the heuristics and 0.30 for error. The overall average was equal to 59s, and the coefficient of variation was 0.93%.

Sources of variation	Degree of freedom	Average square			
		Processed messages	Remained messages	Throughput	Makespan
Heuristics	1	224132428021831 †	57005699011272 †	62257273316 †	15.09†
Error	48	2091387333	544808966	580932	0.30
Total	49				
Overall average		2120866	1448714	35348	59
Coefficient of variation (%)		2.15	1.61	2.15	0.93

† *significant statistical by Fisher-Snedecor's - Probability and error level of 5%.*

Table 13.13: *ANOVA test for case study 4.*

The results of the Scott & Knoot test for workload of 2,500,000 msg are shown in Table 13.14. Regarding processed messages, FIFO was in group «b» with the lowest average of 3,640 msg and qPrior in group «a» with the highest average of 4,238,093.24 msg. Regarding the remained messages, FIFO was in group «a» with the highest average of 2,516,47 msg and qPrior in group «b» with the lowest average of 380,953.60 msg. When analysing throughout, FIFO was in group «b» with the lowest average of 61.66 msg/s and qPrior in group «a» with the highest average of 70,634.90 msg/s. Regarding makespan, FIFO was in group «b» with the lowest average of 58.90s and qPrior in group «a» with the highest average of 60s.

Heuristic	**Average square**			
	Processed messages	Remained messages	Throughput	Makespan
FIFO	3640.00 «b»	2516476.00 «a»	61.66 «b»	58.90 «a»
qPrior	4238093.24 «a»	380953.60 «b»	70,634.90 «a»	60 «b»

Error level of 5% by the Scott & Knoot model.

Table 13.14: *Scott & Knott test for case study 4.*

Discussion and comparison

In the simulation of the execution of the case study 1 with workload till up 1,000,000 msg, all messages were successfully processed when using FIFO and when using qPrior heuristic. No overload was observed. However, in execution with a workload higher than 1,500,000 msg, the number of remained messages was much higher than the number of processed messages when using FIFO. It allow us to conclude that the integration process was overload when using this heuristic, but qPrior had better result than FIFO. In executions with a workload higher than 2,000,000 msg, less than 1% of inbound messages were processed when using FIFO; whereas when using qPrior, about 96% of inbound messages were processed. In executions with a workload higher than 2,500,000 msg, no message was processed when using FIFO in the elapsed time of the simulation, indicating that there was a threshold from which this heuristic does not process messages in a given elapsed time; whereas when using qPrior, about 75% of inbound messages were processed.

In the simulation of the execution of the case study 2 with workload till up 1,500,000 msg, all messages were successfully processed when using FIFO and when using qPrior. No overload was observed. However, in execution with a workload higher than 2,000,000 msg, the number of remained messages was much higher than the number of processed messages when using FIFO. It allow us to conclude that the integration process was overload when using this heuristic, but qPrior had better result than FIFO, processing successfully all messages. In execution with a workload higher than 2,500,000 msg, about 33% of inbound messages were processed when using FIFO; whereas when using qPrior, about 93% of inbound messages were processed.

In the simulation of the execution of the case study 3 with workload till up 1,000,000 msg, all messages were successfully processed when using FIFO and when using qPrior. No overload was observed. However, in execution with a workload higher than 1,500,000 msg, the number of remained messages was much higher than the number of processed messages when using FIFO. It allow us to conclude that the integration process was overload when using this heuristic, but qPrior had better result than FIFO, processing successfully all messages. In execution with a workload higher than 2,000,000 msg, less than 1% of inbound messages were processed when using FIFO; whereas when using qPrior, about 82% of inbound messages were processed. In execution with a workload higher than 2,500,000 msg, less than 1% of inbound messages were processed when using FIFO in the elapsed time of the simulation; whereas when using qPrior, about 68% of inbound messages were processed.

In the simulation of the execution of the case study 4 with workload till up 1,500,000 msg, approximately all messages were successfully processed when using FIFO and when using qPrior.No overload was observed. However, in execution with a workload higher than 2,000,000 msg, the number of remained messages was much higher than the number of processed messages when using FIFO; thus, the integration process was overload when using this heuristic, but qPrior had better result than FIFO, processing successfully all messages. In execution with a workload higher than 2,500,000 msg, less than 1% of inbound messages were processed when using FIFO in the elapsed time of the simulation; whereas when using qPrior, about 84% of inbound messages were processed.

The averages of the throughput in execution of integration processes were higher when using qPrior than FIFO for all workloads and integration processes tested. ANOVA test confirmed that the use of different heuristics generates a significant difference in the average of processed messages, remained messages, throughput, and makespan cf. Tables 13.7, 13.7, 13.7, and 13.13. The coefficients of variation were reduced, indicating that the experiment is adequate and reliable. The Scott & Knott test showed that the best performance was achieved with the qPrior heuristic, in the execution of integration processes under an overload situation. This test confirmed that there was a statistically significant difference between the heuristics, cf. Tables 13.8, 13.10, 13.12, and 13.14. Regarding the research question and hypothesis:

RQ: When the workload increases and the integration processes enter in an overload situation, qPrior improved the performance of the executions of integration processes. ANOVA and Scott & Knott test proved that there was a statistical difference between the heuristics, in these cases.

Our hypothesis for the research question was confirmed:

H: In terms of the number of processed messages per time unit, using the qPrior heuristic, the performance of executions of integration processes remains good even in overload situations in contrast to other tested heuristics that degraded the performance.

13.4 Threats to validity

We evaluated the factors that can influence the results of the experiments and how we tried to mitigate them. These factors are threats to validity present in any empirical research [39, 198]. Four types of threats to validity are discussed here, they are: constructor validity, conclusion validity, internal validity, and external validity.

Constructor validity

Constructor validity discusses whether the planning and execution of the study are well adequate to answer the research question. We planned the experiments according to procedures from empirical software engineering [17, 95, 198]. First, we defined our research questions, formulated our hypotheses, and defined the independent and dependent variables. We also provided information about the execution environment, supporting tools, execution and data collection. Then, we performed the experiments in different scenarios and used statistical techniques to evaluate the results.

Conclusion validity

As reported by Wohlin et al. [198], conclusion validity "is concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment". To assure that the actual outcome observed in our experiment is related to the used heuristics and that there was a significant difference amongst them, we used statistical techniques such as ANOVA, Scott & Knott, and Regression analysis.

Internal validity

Internal validity aims to ensure that the treatment caused the outcome, mitigating effects of other uncertain factors or not measured [57]. Instrumentation and source of noise are possible threats. We experimented with the same machine, which was on security mode, with minimal features and disconnected from the Internet during the executions to minimise interference in the execution time of the experiments. We built our algorithms in Java, so, the first executions of codes are slower, and it is advisable to let the virtual machine eventually perform code optimisation [148]. Then, first execution were to warm up the Java virtual machine and so dropped. Additionally, the researchers accurately inspected the procedures and used statistical tests to validate the measures.

External validity

External validity focuses on the generalisation of the results outside the scope of our study [57]. This study is generalised for integration platforms that adopt the integration patterns by Hohpe and Woolf [85], the style Pipes-and-Filters, and task-based execution model. We reported this study following a practical guideline [198], so that exact repetition is possible, required by scientific methods. The experiments are valid to test other parameters, such as integration processes, message arrival rate, simulation duration.

13.5 Summary

The goal of this chapter was reported two experiments that validated the proposed heuristic. For this, we simulated the execution of four case studies, of which, two are benchmarks of the EAI literature, and two are real integration processes. These experiments were designed according to procedures from empirical software engineering and statistical techniques were used to evaluate the results. In the first of them, we used the PSO meta-heuristic to find the near-optimal preemption for qPrior to maximise the throughput of executions of an integration process with a given workload. In the second, we compared the performance of the executions of integration processes when using the qPrior heuristic with the performance when using FIFO heuristic. We analysed statistically the results of these experiments and examined their restrictions and elements that could bias them.

Part V

Final Remarks

Chapter 14
Conclusion

A ship in port is safe, but that's not what ships are built for.

Grace Hopper, American computer scientist (1906–1992)

Cloud computing has provided several services to companies that have leveraged their business processes. The business processes require the integration of different applications that compose the software ecosystem. Thus, companies use software tools such as integration platforms that allow for keeping information on all these systems consistent and synchronised. Usually, an integration platform is composed of a domain-specific language, a development toolkit, a run-time system, and monitoring tools. The runtime system is responsible for running integration processes and, therefore, its performance is what most frequently drives the decision of companies when choosing an integration platform.

We reviewed the runtime systems of integration platforms to identify properties that can have an impact on their performance when executing integration processes. These properties were organised into two dimensions, namely: message processing and fairness execution. The former dimension addresses the efficiency of the runtime system to process a message, which refers to the improvement of the rate of messages processed in an integration process. The latter dimension focuses on the assignment of threads to tasks to provide a minimum process time for a message in an integration process. We evaluated the runtime systems of nine different current open source message-based integration platforms. Taking these dimensions and properties, we analysed research gaps to be solved to improve the performance of runtime systems from the perspective of message processing and fairness execution. One of the gaps concerns with optimisation of the task scheduling and the allocation of threads in the runtime systems of integration platforms. Current runtime systems are not endowed with a proper task scheduling heuristic to handle high and continuous rates of data of the contemporaneous environments. Overload situations cause a degradation in performance of the execution of integration processes with the current heuristics and consequently, financial damage for the companies.

The current studies regarding task scheduling propose taxonomies and methods that provide directives to deal with dynamic scheduling in similar environments. Based on these studies, we characterised the task scheduling of the execution of integration processes. Besides, we developed a representation for integration processes as a directed graph, called the Integration Operation Typed Graph or IOTG, that takes into account the function and the logic operation of the task. Our approach for task sequencing and timing uses low complexity heuristic as well as hybrid methods to make decisions based on the current integration processes status.

In the context of Cloud computing, task scheduling concerns with the reduction of costs [68], with the handling of high workloads [179], and with quality of software in terms of flexibility and response time [56]. The high workloads can cause situations of overload in integration processes; in such cases, the average of processed messages is lower than the average of messages that remained in queues waiting to be processed. Most of the open source integration platforms adopt the FIFO heuristic for their task scheduling. This heuristic is mainly used when the scheduling requirements are unknown because it is a straightforward heuristic and achieves consistent results in the major of the scenarios. However, the performance of this heuristic depends on several factors and is rarely optimal for all situations, such as, situations of overload. In these situations, the scheduling following FIFO tends to concentrate on initial tasks of the workflow, degrading the performance of the execution of the integration processes. So, a fair task scheduling that provides an optimal resources allocation to maximise performance is necessary. This optimal resources allocation requires an active and dynamic task scheduling heuristic, which is capable of increasing the number of processed messages per time unit.

In this thesis, we develop a heuristic called Queue Priority (qPrior), that addresses task scheduling for the execution of integration processes, maintaining performance even in overload situations. In this heuristic, there are multiple prioritised queues to store instances of tasks that wait for available threads to execute. In the order of priority, a task that has more predecessor tasks has more priority in its execution. Threads available in the pools check the queues according to this order of priority and execute a predefined number of tasks at each checking. The number of tasks is called preemption and was found by the Particle Swarm Optimisation meta-heuristic, in order to maximise the number of processed messages per time unit.

We also modelled the average processed messages as a function of the elements of the characterisation and of the workload. We created three models, namely: «average processed messages *vs*. workload», «average processed messages *vs*. IOTG», and «generic average processed messages». In every model, we verified the elements of the characterisation that impacted the average of messages processed. We applied the models and compared the observed values and estimated values for average processed messages. The high correlation coefficient confirmed that the model indeed explains the behaviour of the average of messages processed.

We develop a software tool, called Integration Process Simulator (IPS), to simulate the execution of integration processes. In our literature review, we found simulators for heuristics in the research field of Cloud computing and Grid systems. However, in the research field of EAI, we found a single article that proposed an incipient tool to experiment a heuristic FIFO [80]. Our tool allows the evaluation of heuristics for task scheduling of integration processes carried out by runtime systems. The architecture of the IPS is flexible and allows the incorporation of other heuristics by addition of Java classes. Besides, IPS allows the variation of the simulation scenarios with different messages input rates, the total workload of messages, the initial workload of messages, simulation time, and integration processes. The performance metrics computed by IPS are the makespan, the throughput, the number of processed messages, and the number of remained messages in the execution of integration processes .

We performed two experiments to validate the qPrior heuristic. In the first experiment, we evaluated the feasibility of the optimisation of the preemption of the qPrior using the PSO meta-heuristic. The results showed that the throughput of the executions of integration processes with the preemption found by PSO is higher than the performance using other random values for preemption. The statistical tests confirmed that different preemptions generate significant differences in the averages of throughput, legitimating the optimisation of the preemption. Additionally, statistical tests showed that for some intervals of values of preemption there is no difference in statistic terms, i.e., any value into a given interval produces the same throughput average. In the second experiment, we evaluated the performance of executions of the integration processes, varying the workload. The results showed that the executions when using qPrior responded more rapidly, processing messages than when using FIFO when the integration processes are under overload situations. Additionally, the performance metrics of executions with qPrior were better than with FIFO. We also observed that the FIFO heuristic had the threshold at which did not process messages in time measured; whereas, the qPrior heuristic had few variations in the average of processed messages. The fact that we have implemented qPrior algorithm in the Java programming language, used by many open source integration platforms, facilities its insertion into these platforms, because this dispenses with the need for compatibility of codes or learning by software engineers.

We summarise the main contributions as follow:

- We have developed mathematical models for number of processed messages of the execution of integration processes.
- We have implemented an heuristic for task scheduling for integration processes and demonstrated that it works. The results prove that it is better than existing ones in terms of throughput.
- We have developed a model for representation of integration processes according to their type of logic operation of the tasks.
- We have characterised the task scheduling of the execution of integration processes. This characterisation provides guidelines for that new heuristics can be proposed and tested.
- We have implemented a simulation tool for the execution of integration processes, used it for evaluation of task scheduling heuristics and for proving that the qPrior heuristic is better than existing ones in terms of throughput.

This research work will continue in our research group, in which we intend to experiment the qPrior heuristic with an extensive data set, involving a large number of study cases and workloads; to evaluate the performance of this heuristic with different amount of computational resource and varied traffic of messages; to extract other performance metrics, such as memory use, CPU cycles, and CPU time; to optimise other objectives, such as cloud cost and quality of service; to implement qPrior heuristic in an integration platform that follows the integration patterns documented by Hohpe and Woolf [85] and the architectural style Pipes-and-Filters [2], and that adopt the task-based execution model, such as the Guaraná integration platform; and to validate the improvement of runtime systems by use of our heuristic in a real scenario that involves cloud services with high data input rate.

Appendix A

Profiles of Integration Processes

In this section, we present the profiles of case studies described in Chapter 9. A profile is a set of parameters that characterises a conceptual model of an integration process. The parameters are indicated as follow.

vectorIdTask - Identification of the integration process tasks.

vectorNextTask - Identification of the next task of each task of the integration process.

vectorTimeExec - Identification of the execution time range of task of each task of the integration process.

vectorOper - Identification of the logic operation type of task of each task of the integration process.

vetorParallelTask - Identification of the parallel tasks of the integration process.

lastTask - Identification of the last tasks of the integration process.

A.1 Profile of case study 1

vectorIdTask = {1,2,3,4,5,6,4,8,9,10,11,12,13,14,15,16};

vectorNextTask = {2}, {3}, {4,6}, {5}, {6}, {7}, {8}, {9,11,14}, {10}, {}, {12}, {13}, {}, {15}, {16}, {};

vectorTimeExec = {{1,2}, {1,2}, {2,3}, {1,2}, {1,2}, {3,4}, {1,2}, {2,3}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}};

vectorOper = {{}, {}, {"and"}, {}, {}, {}, {}, {"and"}, {}, {}, {}, {}, {}, {}, {}, {}};

vetorParallelTask = {0,0,0,0,0,4,0,0,0,0,9,10,0,9,10,13};

lastTask = {11}.

A.2 Profile of case study 2

vectorIdTask = {1,2,3,4,5,6,4,8,9,10,11,12,13,14,15,16,17};

vectorNextTask = {{2}, {3}, {4,12,17}, {5,7}, {6}, {7}, {8}, {9}, {10}, {11}, {}, {13,15}, {14}, {15}, {16}, {9}, {}};

vectorTimeExec = {{1,2}, {1,2}, {2,3}, {2,3}, {1,2}, {1,2}, {3,4}, {1,2}, {3,4}, {1,2}, {1,2}, {2,3}, {1,2}, {1,2}, {3,4}, {1,2}, {1,2}};

vectorOper= {{}, {}, {"or"}, {"and"}, {}, {}, {}, {}, {}, {}, {}, {"and"}, {}, {}, {}, {}, {}};

vetorParallelTask = {0,0,0,0,0,0,0,0,0,0,0,4,5,6,7,8,4};

lastTask = {11,17}.

A.3 Profile of case study 3

vectorIdTask = {1,2,3,4,5,6,4,8,9,10,11,12,13,14,15,16,17,18,19};

vectorNextTask = {{3}, {3}, {4}, {5,7}, {6}, {7}, {8}, {9}, {10,18}, {11}, {12,14}, {13}, {14}, {15}, {16}, {17}, {}, {19}, {}};

vectorTimeExec = {{1,2}, {1,2}, {3,4}, {2,3}, {1,2}, {1,2}, {3,4}, {1,2}, {2,3}, {1,2}, {2,3}, {1,2}, {1,2}, {3,4}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}};

vectorOper= {{}, {}, {}, {"and"}, {}, {}, {}, {}, {"or"}, {}, {"and"}, {}, {}, {}, {}, {}, {}, {}, {}};

vetorParallelTask = {0,1,0,0,0,0,5,0,0,0,0,0,0,12,0,0,0,16,17};

lastTask = {17,19}.

A.4 Profile of case study 4

vectorIdTask = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

vectorNextTask = {{2}, {3}, {4,6}, {5}, {6}, {7}, {8}, {9,11,14}, {10}, {}, {12}, {13}, {}, {15}, {16}, {}};

vectorTimeExec = {{1,2}, {1,2}, {2,3}, {1,2}, {1,2}, {3,4}, {1,2}, {2,3}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}, {1,2}, {1, }};

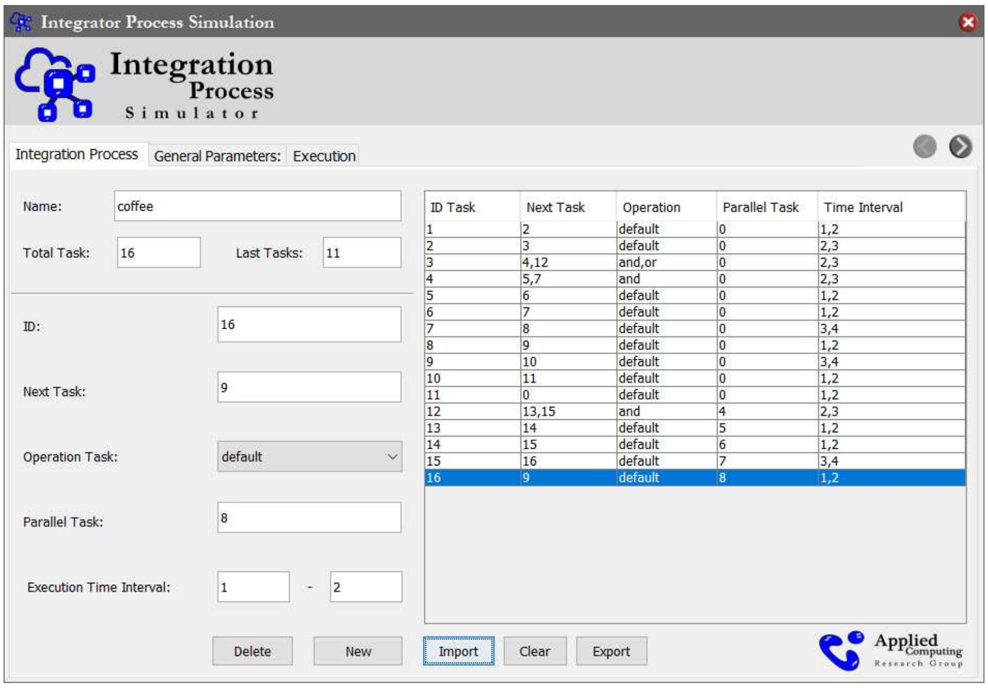
vectorOper = {{}, {}, {"and"}, {}, {}, {}, {}, {"and"}, {}, {}, {}, {}, {}, {}, {}, {}};

vetorParallelTask = {0,0,0,0,0,4,0,0,0,0,0,9,10,0,9,10,13

lastTask = {11,14,17}.

Appendix B

Integration Process Simulator


Integrator Process Simulation
Integration Process Simulator
Integration Process
General Parameters:
Execution
Delete
New
Import
Clear
Export
Applied Computing Research Group

Name: coffee

Total Task: 16 Last Tasks: 11

ID: 16

Next Task: 9

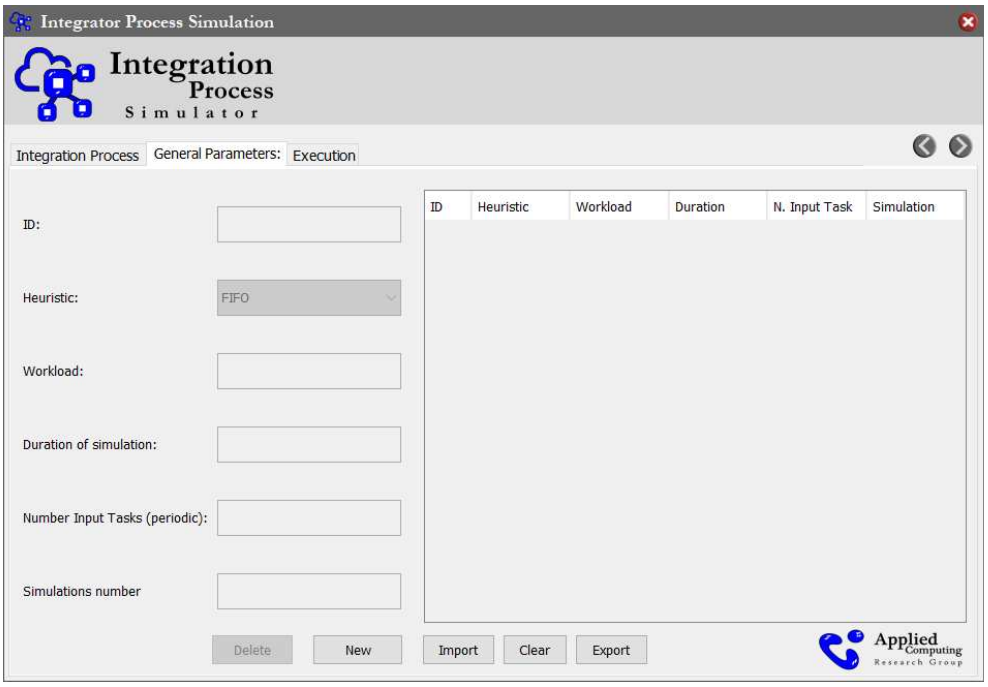
Operation Task: default

Parallel Task: 8

Execution Time Interval: 1 - 2

ID Task	Next Task	Operation	Parallel Task	Time Interval
1	2	default	0	1,2
2	3	default	0	2,3
3	4,12	and,or	0	2,3
4	5,7	and	0	2,3
5	6	default	0	1,2
6	7	default	0	1,2
7	8	default	0	3,4
8	9	default	0	1,2
9	10	default	0	3,4
10	11	default	0	1,2
11	0	default	0	1,2
12	13,15	and	4	2,3
13	14	default	5	1,2
14	15	default	6	1,2
15	16	default	7	3,4
16	9	default	8	1,2

Figure B.1: *Graphic interface of IPS for integration process.*


Integrator Process Simulation
Integration Process Simulator
Integration Process
General Parameters:
Execution
Delete
New
Import
Clear
Export
Applied Computing Research Group

ID:

Heuristic: FIFO

Workload:

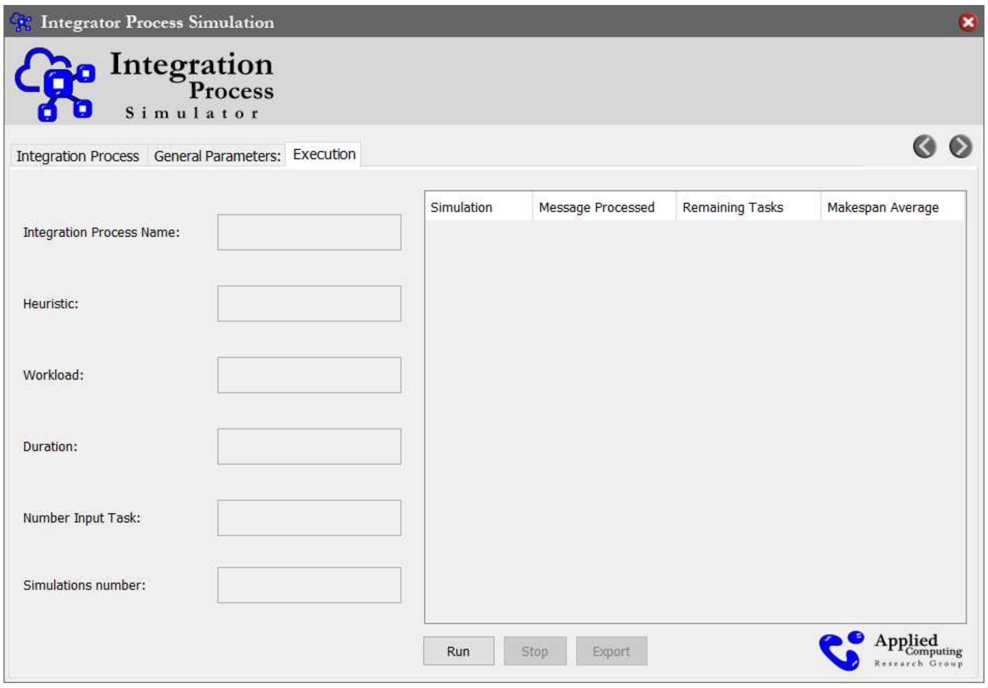
Duration of simulation:

Number Input Tasks (periodic):

Simulations number

ID	Heuristic	Workload	Duration	N. Input Task	Simulation

Figure B.2: *Graphic interface of IPS for general parameters.*


Integrator Process Simulation
Integration Process Simulator
Integration Process
General Parameters:
Execution
Run
Stop
Export
Applied Computing Research Group

Integration Process Name:

Heuristic:

Workload:

Duration:

Number Input Task:

Simulations number:

Simulation	Message Processed	Remaining Tasks	Makespan Average

Figure B.3: *Graphic interface of IPS for execution.*

Bibliography

[1] S. Abdulhamid, A. L. Shafie, and I. Idris. *Tasks Scheduling Technique Using League Championship Algorithm for Makespan Minimization in IaaS Cloud*. *Journal of Engineering and Applied Sciences*, 9(1): 2528–2533, 2014.

[2] C. Alexander, S. Ishikawa, and M. Silvertein. *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

[3] Y. Alipouri, M. H. Sebt, A. Ardeshir, and W. T. Chan. *Solving the fs-rcpsp with hyper-heuristics: A policy-driven approach*. *Journal of the Operational Research Society*, 70(3):1–17, 2018.

[4] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi. *Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues*. *Journal of Systems and Software*, 113(1):1–26, 2016.

[5] M. R. AlRashidi and M. E. El-Hawary. *A survey of particle swarm optimization applications in electric power systems*. *IEEE Transactions on Evolutionary Computation*, 13(4):913–918, 2009.

[6] S. Anand, P. Singh, and B. M. Sagar. *Working with cassandra database*. In *Information and Decision Sciences*, pages 531–538, 2018.

[7] D. C. andreas Heinig, P. Marwedel, and A. Mallik. *Automatic extraction of pipeline parallelism for embedded software using linear programming*. In *International Conference on Parallel and Distributed Systems*, pages 699–706, 2011.

[8] S. Angra, A. Chanda, and V. Chawla. *Comparison and evaluation of job selection dispatching rules for integrated scheduling of multi-load automatic guided vehicles serving in variable sized flexible manufacturing system layouts: A simulation study*. *Management Science Letters*, 8(4):187–200, 2018.

[9] N. Anwar and H. Deng. *Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. Future Internet*, 10(5):1–23, 2018.

[10] Apache Camel Version 2.20. *Design notes for threadpool configuration*, 2017. Access date: 27 feb. 2017.

[11] Apache Synapse Documentation Version 2.0.0. *Apache synapse*, 2017. Access date: 27 feb. 2017.

[12] Z. R. M. Azmi, K. A. Bakar, A. H. Abdullah, M. S. Shamsir, and W. N. W. Manan. *Performance comparison of priority rule scheduling algorithms using different inter arrival time jobs in grid environment. International Journal of Grid and Distributed Computing*, 4(3): 61–70, 2011.

[13] F. Bahadur, M. Naeem, M. J. Shad, and H. Wahab. *FBOS: Frequency based optimization strategy for thread pool system. Nucleus*, 51(1): 93–107, 2014.

[14] K. R. Baker and D. Trietsch. *Principles of sequencing and scheduling.* Wiley, 2018.

[15] S. Balko and A. Barros. *In-Memory Business Process Management.* In *International Conference on Enterprise Distributed Object Computing*, pages 74–83, 2015.

[16] F. Ballestín, Á. Pérez, and S. Quintanilla. *Scheduling and rescheduling elective patients in operating rooms to minimise the percentage of tardy patients. Journal of Scheduling*, 22(1):1–12, 2018.

[17] V. R. Basili, D. Rombach, K. S. B. Kitchenham, D. Selby, and R. W. Pfahl. *Empirical software engineering issues.* Springer Berlin/Heidelberg, 2007.

[18] K. Bauer, D.Y.Kuznetsov, and A. Pominov. *Designing the search service for enterprise portal based on oracle universal content management. Journal of Physics: Conference booktitle*, 803(1):1–5, 2017.

[19] H. Behera, R. Mohanty, and D. Nayak. *A new proposed dynamic quantum with re-adjusted round robin scheduling algorithm and its performance analysis. International Journal of Computer Applications*, 975(1):8887, 2011.

[20] C. L. M. Belusso, S. Sawicki, F. Roos-Frantz, and R. Z. Frantz. *A study of petri nets, markov chains and queueing theory as mathematical modelling languages aiming at the simulation of enterprise application integration solutions: a first step*. *Procedia Computer Science*, 100 (1):229–236, 2016.

[21] R. S. Bhadoria, N. S. Chaudhari, and G. S. Tomar. *The performance metric for enterprise service bus (ESB) in SOA system: Theoretical underpinnings and empirical illustrations for information processing*. *Information Systems*, 65(1):158–171, 2017.

[22] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, M. Sterna, and J. Weglarz. *Handbook on scheduling: From theory to practice*. Springer International Publishing, 2019.

[23] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. *Task scheduling strategies for workflow-based applications in grids*. In *IEEE International Symposium on Cluster Computing and the Grid*, pages 759–767, 2005.

[24] M. Boehm, D. Habich, S. Preissler, W. Lehner, and U. Wloka. *Cost-based vectorization of instance-based integration processes*. *Information Systems*, 36(1):3–29, 2011.

[25] Z. Brahmi and C. Gharbi. *Temporal reconfiguration-based orchestration engine in the cloud computing*. In *International Conference on Business Information Systems*, pages 73–85, 2014.

[26] D. Bratton and J. Kennedy. *Defining a standard for particle swarm optimization*. In *Swarm Intelligence Symposium*, pages 120–127, 2007.

[27] D. W. Burns, J. D. Allen, M. D. Upton, D. D. Boggs, and A. B. Kyker. *Determination of approaching instruction starvation of threads based on a plurality of conditions*, 2003. US Patent 6,651,158.

[28] R. Buyya. *High performance cluster computing: Architectures and systems*. Prentice Hall, 1999.

[29] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility*. *Future Generation computer systems*, 25(6):599–616, 2009.

[30] L.-C. Canon and E. Jeannot. *A comparison of robustness metrics for scheduling DAGs on heterogeneous systems.* In *International Conference on Cluster Computing*, pages 558–567, 2007.

[31] O. Cats and Z. Gkioulou. *Modeling the impacts of public transport reliability and travel information on passengers′ waiting-time uncertainty. EURO Journal on Transportation and Logistics*, 6(3):247–270, 2017.

[32] V. Černỳ. *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[33] T. Chaari, S. Chaabane, N. Aissani, and D. Trentesaux. *Scheduling under uncertainty: Survey and research directions.* In *International conference on advanced logistics and transport*, pages 229–234, 2014.

[34] W. Chen and E. Deelman. *Workflow overhead analysis and optimizations.* In *Workflows in support of large-scale science*, pages 11–20, 2011.

[35] A. Chhabra and Oshin. *Hybrid psacga algorithm for job scheduling to minimize makespan in heterogeneous grids.* In *Industry Interactive Innovations in Science, Engineering and Technology*, pages 107–120, 2018.

[36] A. M. Chirkin, A. S. Z. Belloum, S. V. Kovalchuk, M. X. Makkes, M. A. Melnik, A. A. Visheratin, and D. A. Nasonov. *Execution time estimation for workflow scheduling. Future Generation Computer Systems*, 75(1):376–387, 2017.

[37] K. Chronaki, A. Rico, M. Casas, M. Moretó, R. M. Badia, E. Ayguadé, J. Labarta, and M. Valero. *Task scheduling techniques for asymmetric multi-core systems. IEEE Transactions on Parallel and Distributed Systems*, 28(7):2074–2087, 2017.

[38] C. D. Cruz. *Programa genes: estatística experimental e matrizes.* Editora Universidade Federal de Viçosa, 2006.

[39] D. S. Cruzes and L. ben Othman. *Threats to validity in empirical software security research.* In *Empirical Research for Software Security*, pages 295–320, 2017.

[40] A. da Silva Dias. *Computação em nuvem elástica auxiliada por agentes computacionais e baseada em histórico para web services.* PhD thesis, Universidade de São Paulo, 2015.

[41] A. da Silva Dias, L. H. V. Nakamura, J. C. Estrella, R. H. C. Santana, and M. J. Santana. *Providing iaas resources automatically through prediction and monitoring approaches.* In *Symposium on Computers and Communication*, pages 1–7, 2014.

[42] M. Davari and E. Demeulemeester. *The proactive and reactive resource-constrained project scheduling problem. Journal of Scheduling*, 22(2): 1–27, 2017.

[43] G. De, Z. Tan, M. Li, L. Huang, and X. Song. *Two-stage stochastic optimization for the strategic bidding of a generation company considering wind power uncertainty. Energies*, 11(12):1–21, 2018.

[44] A. Dekkers and E. Aarts. *Global optimization and simulated annealing. Mathematical programming*, 50(1-3):367–393, 1991.

[45] P. Demo. *Metodologia do conhecimento científico. 7a reimpr.* Atlas, 2009.

[46] D. Ding, Z. Wang, Q.-L. Han, and G. Wei. *Neural-network-based output-feedback control under round-robin scheduling protocols. IEEE transactions on cybernetics*, 75(99):1–13, 2018.

[47] M. Dorigo. *Optimization, learning and natural algorithms.* PhD thesis, Politecnico di Milanoy, 1992.

[48] M. Dorigo and T. Stützle. *The ant colony optimization metaheuristic: Algorithms, applications, and advances.* In *Handbook of metaheuristics*, pages 250–285, 2003.

[49] D. Dossot, J. D'Emic, and V. Romero. *Mule in action.* Manning, 2014.

[50] R. Eberhart and J. Kennedy. *Particle swarm optimization.* In *Proceedings of the IEEE international joint conference on neural networks*, pages 1942–1948, 1995.

[51] N. Ebert and K. Weber. *Integration Platform as a Service in der Praxis: Eine Bestandsaufnahme.* In *Multikonferenz Wirtschaftsinformatik*, pages 1675–1685, 2016.

[52] N. Ebert, K. Weber, and S. Koruna. *Integration platform as a service. Business & Information Systems Engineering*, 59(5):375–379, 2017.

[53] S. Elmougy, S. Sarhan, and M. Joundy. *A novel hybrid of shortest job first and round robin with dynamic variable quantum time task scheduling technique. Journal of Cloud computing*, 6(1):1–12, 2017.

[54] C. Emmersberger and F. Springer. *Tutorial: Open source enterprise application integration - introducing the event processing capabilities of apache camel.* In *International Conference on Distributed Event-based Systems e tecnologias da informação*, pages 259–268, 2013.

[55] A. P. Engelbrecht. *Computational intelligence: an introduction.* John Wiley & Sons, 2007.

[56] K. Fan, Y. Zhai, X. Li, and M. Wang. *Review and classification of hybrid shop scheduling. Production Engineering*, 12(5):597–609, 2018.

[57] R. Feldt and A. Magazinius. *Validity threats in empirical software engineering research-an initial survey.* In *International Conference on Software Engineering and Knowledge Engineering*, pages 374–379, 2010.

[58] M. Fisher, J. Partner, M. Bogoevice, and I. Fuld. *Spring integration in action.* Manning Publications Co., 2012.

[59] R. Z. Frantz, R. Corchuelo, and J. L. Arjona. *An efficient orchestration engine for the cloud.* In *International Conference on Cloud computing Technology and Science*, pages 711–716, 2011.

[60] R. Z. Frantz, R. Corchuelo, and C. Molina-Jiménez. *A proposal to detect errors in enterprise application integration solutions. Journal of Systems and Software*, 85(3):480–497, 2012.

[61] R. Z. Frantz, R. Corchuelo, and F. Roos-Frantz. *On the design of a maintainable software development kit to implement integration solutions. Journal of Systems and Software*, 111(1):89–104, 2016.

[62] R. Z. Frantz, A. M. R. Quintero, and R. Corchuelo. *A domain-specific language to design enterprise application integration solutions. International Journal of Cooperative Information Systems*, 20(02):143–176, 2011.

[63] R. Z. Frantz. *Enterprise application integration: an easy-to-maintain model-driven engineering approach.* PhD thesis, University of Seville, 2012.

[64] D. L. Freire, R. Z. Frantz, and F. Roos-Frantz. *Ranking enterprise application integration platforms from a performance perspective: An experience report. Software: Practice and Experience*, 49(5):921–941, 2019.

[65] D. L. Freire, R. Z. Frantz, and F. Roos-Frantz. *Towards optimal thread pool configuration for run-time systems of integration platforms. International Journal of Computer Applications in Technology*, 62(2): 129–147, 2020.

[66] D. L. Freire, R. Z. Frantz, F. Roos-Frantz, and S. Sawicki. *A methodology to rank enterprise application integration platforms from a performance perspective: an analytic hierarchy process-based approach. Enterprise Information Systems*, 13(9):1292–1322, 2019.

[67] D. L. Freire, R. Z. Frantz, F. Roos-Frantz, and S. Sawicki. *Optimization of the size of thread pool in runtime systems to enterprise application integration: a mathematical modelling approach. Trends in Applied and Computational Mathematics*, 20(1):169–188, 2019.

[68] D. L. Freire, R. Z. Frantz, F. Roos-Frantz, and S. Sawicki. *Survey on the run-time systems of enterprise application integration platforms focusing on performance. Software: Practice and Experience*, 49(3):341–360, 2019.

[69] V. Garousi, M. Felderer, and M. V. Mäntylä. *Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Information and Software Technology*, 106(1):1–22, 2018.

[70] A. Georges, D. Buytaert, and L. Eeckhout. *Statistically rigorous Java performance evaluation. ACM SIGPLAN Notices*, 42(10):57–76, 2007.

[71] R. Ghafouri, A. Movaghar, and M. Mohsenzadeh. *A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds. Peer-to-Peer Networking and Applications*, 12(1):241–268, 2019.

[72] F. Giesemann, G. Payá-Vayá, L. Gerlach, H. Blume, F. Pflug, and G. von Voigt. *Using a genetic algorithm approach to reduce register file pressure during instruction scheduling.* In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 179–187, 2017.

[73] G. Gleyzer and J. Howes. *System and method for supporting dynamic thread pool sizing in a distributed data grid*, 2017. US Patent 9,547,521 B2.

[74] D. E. Goldberg. *Genetic algorithms in search. optimization and machine learning.* Addison-Wesley, 1989.

[75] I. E. Grossmann, R. M.Apap, B. A. Calfa, P. García-Herreros, and Q. Zhang. *Recent advances in mathematical programming techniques for the optimization of process systems under uncertainty. Computers & Chemical Engineering*, 91(4):3–14, 2016.

[76] F. Guo, L. Yu, S. Tian, and J. Yu. *A workflow task scheduling algorithm based on the resources' fuzzy clustering in Cloud computing environment. International Journal of Communication Systems*, 28(6): 1053–1067, 2015.

[77] K. Guttridge, M. Pezzini, E. Golluscio, E. Thoo, K. Iijima, and M. Wilcox. *Magic quadrant for enterprise integration platform as a service 2017.* Technical report, Gartner, Inc, 2017.

[78] K. Guttridge, M. Pezzini, P. Malinverno, K. Iijima, J. Thompson, E. Thoo, and E. Golluscio. *Magic quadrant for enterprise integration platform as a service, worldwide.* Technical report, Gartner, Inc, 2016.

[79] M. Harman, K. Lakhotia, J. Singer, D. R. White, and S. Yoo. *Cloud engineering is search based software engineering too. Journal of Systems and Software*, 86(9):2225–2241, 2013.

[80] I. G. Haugg, R. Z. Frantz, F. Roos-Frantz, S. Sawicki, and B. Zucolotto. *Towards optimisation of the number of threads in the integration platform engines using simulation models based on queueing theory. Revista Brasileira de Computação Aplicada*, 11(1):48–58, 2019.

[81] W. He and L. D. Xu. *Integration of distributed enterprise applications: A survey. IEEE Transactions on Industrial Informatics*, 10(1): 35–42, 2014.

[82] I. Hernández, S. Sawicki, F. Roos-Frantz, and R. Z. Frantz. *Cloud Configuration Modelling: A Literature Review from an Application Integration Deployment Perspective. Procedia Computer Science*, 64(1): 977–983, 2015.

[83] M. H. Hilman, M. A. Rodriguez, and R. Buyya. *Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions*. *ACM Computing Surveys*, 1(1):1–33, 2018.

[84] G. Hohpe. *Your coffee shop doesn't use two-phase commit [asynchronous messaging architecture]*. *IEEE software*, 22(2):64–66, 2005.

[85] G. Hohpe and B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[86] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, 1975.

[87] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[88] A. J. L. Hors and S. Speicher. *The linked data platform (ldp)*. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1–2, 2013.

[89] Y. Hu, F. Zhu, L. Zhang, Y. Lui, and Z. Wang. *Scheduling of manufacturers based on chaos optimization algorithm in cloud manufacturing*. *Robotics and Computer-Integrated Manufacturing*, 58(1):13–20, 2019.

[90] K. Hwang, J. Dongarra, and G. C. Fox. *Distributed and Cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.

[91] C. Ibsen and J. Anstey. *Camel in action*. Manning Publications Co., 2010.

[92] K. Indrasiri. *Introduction to WSO2 ESB*. Springer, 2016.

[93] M. Janetschek, R. Prodan, and S. Benedict. *A workflow runtime environment for manycore parallel architectures*. *Future Generation Computer Systems*, 75(1):330–347, 2017.

[94] D. Jayasinghe and A. Azeez. *Apache Axis2 web services*. Packt Publishing Ltd, 2011.

[95] A. Jedlitschka and D. Pfahl. *Reporting guidelines for controlled experiments in software engineering.* In *International Symposium on Empirical Software Engineering*, pages 95–104, 2005.

[96] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman. *Scottknott: a package for performing the scottknott clustering algorithm in r.* *Trends in Applied and Computational Mathematics*, 15(1):3–17, 2014.

[97] S. Jeon and I. Jung. *Experimental evaluation of improved IoT middleware for flexible performance and efficient connectivity.* *Ad Hoc Networks*, 70(1):61–72, 2018.

[98] Jitterbit SuccessCentral Harmony 9.0. *Chunking*, 2017. Access date: 27 feb. 2017.

[99] K. A. D. Jong. *Genetic algorithms: A 10 year perspective.* In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 169–177, 1985.

[100] F. Kaytez, M. C. Taplamacioglu, E. Cam, and F. Hardalac. *Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines.* *International Journal of Electrical Power and Energy Systems*, 67(1):431–438, 2015.

[101] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. *Optimization by simulated annealing.* *Science*, 220(4598):671–680, 1983.

[102] B. Kitchenham, S. Linkman, and D. Law. *Desmet: a methodology for evaluating software engineering methods and tools.* *Computing Control Engineering Journal*, 8(3):120–126, 1997.

[103] B. A. Kitchenham. *Evaluating software engineering methods and tools, part 7: planning feature analysis evaluation.* *ACM SIGSOFT software engineering Notes*, 22(4):21–24, 1997.

[104] M. J. Klein, S. Sawicki, F. Roos-Frantz, and R. Z. Frantz. *On the formalisation of an application integration language using Z notation.* In *International Conference on Enterprise Information Systems*, pages 314–319, 2014.

[105] H. Konsek. *Instant apache servicemix how-to.* Packt Publishing, 2013.

[106] A. Kozik and R. Rudek. *An approximate/exact objective based search technique for solving general scheduling problems.* *Applied Soft Computing*, 62(1):347–358, 2018.

[107] A. R. Kraisig, F. C. Welter, I. G. Haugg, R. Cargnin, F. Roos-Frantz, S. Sawicki, and R. Z. Frantz. *Mathematical model for simulating an application integration solution in the academic context of unijuí university*. *Procedia Computer Science*, 100(1):407–413, 2016.

[108] R. Kuhn, B. Hanafee, and J. Allen. *Reactive design patterns*. Manning Publications Company, 2017.

[109] K. Kurniawan and A. Ashari. *Service orchestration using enterprise service bus for real-time government executive dashboard system*. In *International Conference on Data and Software Engineering*, pages 207–212, 2015.

[110] T. Langer, L. Osinski, and J. Mottok. *A survey of parallel hard-real time scheduling on task models and scheduling approaches*. In *International Conference on Architecture of Computing Systems*, pages 1–8, 2017.

[111] J. Lee, H. Wu, M. Ravichandran, and N. Clark. *Thread tailor: dynamically weaving threads together for efficient, adaptive parallel applications*. *ACM SIGARCH Computer Architecture News*, 38(3): 270–279, 2010.

[112] K.-L. Lee, H. N. Pham, H. S. Kim, H. Y. Youn, and O. Song. *A novel predictive and self – adaptive dynamic thread pool management*. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 93–98, 2011.

[113] J. Li, J. J. Chen, K. Agrawal, Chenyang, C. Gill, and A. Saifullah. *Analysis of federated and global scheduling for parallel real-time tasks*. In *Euromicro Conference on Real-Time Systems*, pages 85–96, 2014.

[114] D. J. Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge University Press, 2005.

[115] S.-W. Lin and K.-C. Ying. *Makespan optimization in a no-wait flowline manufacturing cell with sequence-dependent family setup times*. *Computers & Industrial Engineering*, 128(1):1–7, 2019.

[116] X. Lin, S. L. Janak, and C. A. Floudas. *A new robust optimization approach for scheduling under uncertainty: I. bounded uncertainty*. *Computers & chemical engineering*, 28(6–7):1069–1085, 2004.

[117] D. S. Linthicum. *Cloud computing Changes Data Integration Forever: What's Needed Right Now*. *IEEE Cloud computing*, 4(3):50–53, 2017.

[118] C. L. Liu and J. W. Layland. *Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM*, 20(1): 46–61, 1973.

[119] D. Liu, Y. Xu, Q. Weii, and X. Liu. *Residential energy scheduling for variable weather solar energy based on adaptive dynamic programming. Journal of Automatica Sinica*, 5(1):36–46, 2018.

[120] M. Locatelli. *Simulated annealing algorithms for continuous global optimization: convergence conditions. Journal of Optimization Theory and applications*, 104(1):121–133, 2000.

[121] S. Lupetti and D. Zagorodnov. *Data popularity and shortest-job-first scheduling of network transfers.* In *International Conference on Digital Telecommunications*, pages 26–26, 2006.

[122] L. Ma, K. Agrawal, and R. D. Chamberlain. *A memory access model for highly-threaded many-core architectures. Future Generation Computer Systems*, 30(1):202–215, 2014.

[123] K. Manikas. *Revisiting software ecosystems research: A longitudinal literature study. Journal of Systems and Software*, 117:84–103, 2016.

[124] J. I. C. Manuel, R. B. M. Baquirin, K. S. Guevara, and D. R. Tandingan. *Fittest job first dynamic round robin (FJFDRR) scheduling algorithm using dual queue and arrival time factor: a comparison. IOP Conference Series: Materials Science and Engineering*, 482(1):1–9, 2019.

[125] M. Manzini, D. Erik, M. Urgo, et al.. *A proactive-reactive approach to schedule an automotive assembly line.* In *International Conference on Project Management and Scheduling*, pages 152–155, 2018.

[126] F. Marini and B. Walczak. *Particle swarm optimization (PSO). A tutorial. Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015.

[127] Mark Fisher and Marius Bogoevici and Iwein Fuld and Jonas Partner and Oleg Zhurakousky and Gary Russell and Dave Syer and Josh Long and David Turanski and Gunnar Hillert and Artem Bilan and Amol Nayak. *Spring integration reference manual version*, 2017. Access date: 27 feb. 2017.

[128] R. V. McCarthy, M. M. McCarthy, W. Ceccucci, and L. Halawi. *Predictive models using regression*, pages 89–121. Springer International Publishing, 2019.

[129] P. Mell and T. Grance. *The nist definition of cloud computing*. Information Technology Laboratory, 2011.

[130] Mule User Guide Version 3.8. *Tuning the performance of mule*, 2017. Access date: 27 feb. 2017.

[131] N. Muthuvelu, C. Vecchiola, I. Chai, E. Chikkannan, and R. Buyya. *Task granularity policies for deploying bag-of-task applications on global grids*. *Future Generation Computer Systems*, 29(1):170–181, 2013.

[132] K. Myroshnychenko. *Maturidade de plataformas SOA open source vs proprietários*. Master's thesis, Escola de Comunicação, arquitetura, artes e tecnologias da informação, 2013.

[133] M. Najjari and R. Guilbault. *Formula derived from particle swarm optimization (PSO) for optimum design of cylindrical roller profile under EHL regime*. *Mechanism and Machine Theory*, 90:162–174, 2015.

[134] S. Nazeer, F. Bahadur, M. A. Khan, A. Hakeem, M. Gul, and A. I. Umar. *Prediction and frequency based dynamic thread pool system a hybrid model*. *International Journal of Computer Science and Information Security*, 14(5):299–30, 2016.

[135] T. Nowatzki, N. Ardalani, K. Sankaralingam, and J. Weng. *Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign*. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 1–15, 2018.

[136] R. T. Ogawa and B. Malen. *Towards rigor in reviews of multivocal literatures: Applying the exploratory case study method*. *Review of Educational Research*, 61(3):265–286, 1991.

[137] S.-K. Oh and J.-S. Kim. *A history-based dynamic thread pool method for reducing thread creation and removal overheads*. *The Journal of Advanced Navigation Technology*, 17(2):189–195, 2013.

[138] S. Palanimalai and I. Paramasivam. *An enterprise oriented view on the cloud integration approaches–hybrid cloud and big data*. *Procedia Computer Science*, 50(1):163–168, 2015.

[139] C. Pandey. *Spring integration essentials*. Packt Publishing Ltd, 2015.

[140] H. V. D. Parunak. *Characterizing the manufacturing scheduling problem*. *Journal of manufacturing systems*, 10(3):241–259, 1991.

[141] D. Penciuc, A. Durupt, F. Belkadi, B. Eynard, and H. Rowson. *Towards a PLM interoperability for a collaborative design support system. Conference on Industrial Product-Service Systems*, 25(1):369–376, 2014.

[142] J. L. Pereira and J. o Varajão. *The temporal dimension of business processes: requirements and challenges. International Journal of Computer Applications in Technology*, 59(1):74–81, 2019.

[143] Petals Documentation Version 5.1.0. *Topology configuration*, 2017. Access date: 27 feb. 2017.

[144] H. Peyret, A. Cullen, A. Kramer, and D. Lynch. *The Forrester Wave^{TM}: iPaaS For Dynamic Integration, Q3 2016*. Technical report, Forrester, Inc, 2016.

[145] M. Pezzini and B. J. Lheureux. *Integration platform as a service: moving integration to the cloud*, 2011.

[146] M. Pezzini, Y. V. Natis, P. Malinverno, K. Iijima, J. Thompson, E. Thoo, and K. Guttridge. *Magic quadrant for enterprise integration platform as a service*. Technical report, Gartner, Inc, 2015.

[147] M. L. Pinedo. *Scheduling theory, algorithms, and systems*. Springer International Publishing, 2016.

[148] G. Pinto, F. Castor, and Y. D. Liu. *Understanding energy behaviors of thread management constructs*. In *ACM SIGPLAN Notices*, pages 345–360, 2014.

[149] A. V. K. Prasad. *Exploring the convergence of big data and the internet of things*. IGI Global, 2017.

[150] C. C. Prodanov and E. C. de Freitas. *Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2^{a} edição*. Editora Feevale, 2013.

[151] Product Documentation for Red Hat Fuse 7.0. *Configuring broker persistence*, 2017. Access date: 27 feb. 2017.

[152] T. Rademakers and J. Dirksen. *Open-source esbs in action*. Manning, 2008.

[153] B. P. Rimal and M. Maier. *Workflow scheduling in multi-tenant cloud computing environments. IEEE Transactions on parallel and distributed systems*, 28(1):290–304, 2017.

[154] D. Ritter, F. N. Forsberg, and S. Rinderle-Ma. *Optimization strategies for integration pattern compositions.* In *International Conference on Distributed and Event-based Systems*, pages 88–99, 2018.

[155] D. Ritter, N. May, K. Sachs, and S. Rinderle-Ma. *Benchmarking integration pattern implementations.* In *International Conference on Distributed and Event-based Systems*, pages 125–136, 2016.

[156] M. A. Rodriguez and R. Buyya. *Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Future Generation Computer Systems*, 79(1):739–750, 2018.

[157] M. A. Rodriguez and R. Buyya. *Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. Transactions on Cloud computing*, 2(2):222–235, 2014.

[158] F. Roos-Frantz, M. Binelo, R. Z. Frantz, S. Sawicki, and V. Basto-Fernandes. *Using petri nets to enable the simulation of application integration solutions conceptual models.* In *Conference on Enterprise Information Systems*, pages 87–96, 2015.

[159] P. Rosati, G. Fox, D. Kenny, and T. Lyn. *Quantifying the financial value of cloud investments: A systematic literature review.* In *International Conference on Cloud computing Technology and Science*, pages 194–201, 2017.

[160] J. Russell and R. Cohn. *Fuse esb.* Book on Demand, 2012.

[161] J. Russell and R. Cohn. *Jitterbit integration server.* Book on Demand, 2012.

[162] J. Russell and R. Cohn. *Petals ESB.* Book on Demand, 2012.

[163] T. L. Saaty. *How to make a decision: the analytic hierarchy process. European Journal of Operational Research*, 48(1):9–26, 1990.

[164] T. Sahu, S. K. Verma, M. Shakya, and R. Pandey. *An enhanced round-robin-based job scheduling algorithm in Grid computing.* In *International Conference on Computer Networks and Communication Technologies*, pages 799–807, 2019.

[165] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill. *Multi-core real-time scheduling for generalized parallel task models. Real-Time Systems*, 49 (4):404–435, 2013.

[166] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch. *Elastic business process management: State of the art and open challenges for bpm in the cloud. Future Generation Computer Systems*, 46(1): 36–50, 2015.

[167] A. J. Scott and M. Knott. *A cluster analysis method for grouping means in the analysis of variance. Biometrics*, 30(3):507–512, 1974.

[168] D. F. Sellaro. *Execução de soluções de integração de aplicações empresariais na nuvem: Perspectivas e desafios. IV SFCT*, 7(1):25, 2016.

[169] D. F. Sellaro, R. Z. Frantz, I. Hernández, F. Roos-Frantz, and S. Sawicki. *Task scheduling optimization on enterprise application integration platforms based on the meta-heuristic particle swarm optimization.* In *Brazilian Symposium on Software Engineering*, pages 273–278, 2017.

[170] S. Sengupta, S. Basak, and R. Peters. *Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives. Machine Learning and Knowledge Extraction*, 1(1):157–191, 2018.

[171] F. R. Sequeira, R. Z. Frantz, I. Yevseyeva, M. T. M. Emmerich, and V. Basto-Fernandes. *An EAI based integration solution for science and research outcomes information management. Procedia Computer Science*, 64(1):894–901, 2015.

[172] A. Sethi and H. Kushwah. *Multicore processor technology - advantages and challenges. International Journal of Research in Engineering and Technology*, 4(9):87–89, 2015.

[173] U. Shafi, M. Shah, A. Wahid, M. Kamran, Q. Javaid, M. Asghar, and M. Haider. *A novel amended dynamic round robin scheduling algorithm for timeshared systems. International Arab Journal of Information Technology*, 16(4):1–9, 2019.

[174] R. Shah, F. Bahdur, N.-U.-A. Hu, A. Umer, and M. J. Shad. *Implementation of multiple thread pools based on distribution of service times. Journal Applied Environmental and Biological Sciences*, 7(12): 201–2014, 2017.

[175] S. Sharma. *Ovum decision matrix: Selecting an integration paas (ipaas).* Technical report, Ovum Consulting, 2015.

[176] S. Sharma. *Ovum decision matrix highlights the growing importance of ipaas and API platforms in hybrid integration*. Technical report, Ovum Consulting, 2017.

[177] Y. Shi and R. C. Eberhart. *Parameter selection in particle swarm optimization*. In *International Conference on Evolutionary Programming*, pages 591–600, 1998.

[178] H. Y. Shishido, J. C. Estrella, C. F. M. Toledo, and M. S. Arantes. *Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds*. *Computers & Electrical Engineering*, 69(1):378–394, 2018.

[179] A. Shoukry, J. Khader, and S. Gani. *Improving business process and functionality using IoT based E3-value business model*. *Electronic Markets*, 1(1):1–10, 2019.

[180] G. G. P. B. G. A. Silberschatz. *Operating systems concepts*. JOHN WILEY, 2000.

[181] P. Singh, M. Dutta, and N. Aggarwal. *A review of task scheduling based on meta-heuristics approach in cloud computing*. *Knowledge and Information Systems*, 52(1):1–51, 2017.

[182] W. Stallings. *Operating systems: internals and design principles*. Prentice Hall, 2012.

[183] A. Strauss and J. Corbin. *Basics of qualitative research: grounded theory procedures and techniques*. Newbury Park, CA: Sage Publications, 1990.

[184] A. Sudarsanam, M. Srinivasan, and S. Panchanathan. *Resource estimation and task scheduling for multithreaded reconfigurable architectures*. In *International Conference on Parallel and Distributed System*, pages 323–330, 2004.

[185] D. Sun, H. Yan, S. Gao, X. Liu, and R. Buyya. *An integrated approach to workflow mapping and task scheduling for delay minimization in distributed environments*. *Journal of Parallel and Distributed Computing*, 84(1):51–64, 2015.

[186] E.-G. Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[187] A. S. Tanenbaum and H. Bos. *Modern operating systems.* Pearson, 2015.

[188] W. Tang, W. Feng, J. Deng, M. Jia, and H. Zuo. *Parallel computing for geocomputational modeling.* In *Advances in geographic information science*, pages 37–54, 2018.

[189] D. Terekhov, D. G. Down, and J. C. Beck. *Queueing-theoretic approaches for dynamic scheduling: a survey. Surveys in Operations Research and Management Science*, 19(2):105–129, 2014.

[190] V. Thakur and S. Kumar. *A pragmatic study and analysis of load balancing techniques in parallel computing.* In *Information and Decision Sciences*, pages 447–45400, 2018.

[191] A. Tirkey and K. A. Gary. *Curricular change management with git and drupal: A tool to support flexible curricular development workflows.* In *Software Engineering Research, Management and Applications*, pages 247–253, 2017.

[192] J.-T. Tsai, J.-C. Fang, and J.-H. Chou. *Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Computers & Operations Research*, 40 (12):3045–3055, 2013.

[193] M. L. R. Varela and R. A. Ribeiro. *Distributed manufacturing scheduling based on a dynamic multi-criteria decision model.* In *Recent Developments and New Directions in Soft Computing*, pages 81–93. Springer, 2014.

[194] A. Verma and S. Kaushal. *Cost minimized PSO based workflow scheduling plan for cloud computing. International Journal Information Technology and Computer Science*, 8(1):37–43, 2015.

[195] G. E. Vieira, J. W. Herrmann, and E. Lin. *Rescheduling manufacturing systems: a framework of strategies, policies, and methods. Journal of scheduling*, 6(1):39–62, 2003.

[196] G. N. Wilkinson and C. E. Rogers. *Symbolic description of factorial models for analysis of variance. Journal of the Royal Statistical Society Series C*, 22(3):392–399, 1973.

[197] C. Witt, M. Bux, W. Gusew, and U. Leser. *Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. Information Systems*, 82(1):33–52, 2019.

[198] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. R. anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[199] D. C. Wood and E. H. Forman. *Throughput measurement using a synthetic job stream*. In *Fall Joint Computer Conference*, pages 51–56, 1971.

[200] WSO2 Documention Version 6.3.0. *Configuring properties*, 2017. Access date: 27 feb. 2017.

[201] H. Wu, Z. Tang, and R. Li. *A priority constrained scheduling strategy of multiple workflows for cloud computing*. In *International Conference on Advanced Communication Technology*, pages 1086–1089, 2012.

[202] Y. Xie, Y. Zhu, Y. Wang, Y. Cheng, R. Xu, A. S. Sani, D. Yuan, and Y. Yang. *A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment. Future Generation Computer Systems*, 97(1):36–378, 2019.

[203] K. Yao and B. Liu. *Uncertain regression analysis: an approach for imprecise observations. Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 22(17):5579–5582, 2018.

[204] M. Yavuz and H. Ergin. *Advanced constraint propagation for the combined car sequencing and level scheduling problem. Computers & Operations Research*, 100(1):128–139, 2018.

[205] M. K. Yoon, K. Kim, S. Lee, W. Woo, Ro, and M. Annavaram. *Virtual thread: Maximizing thread-level parallelism beyond gpu scheduling limit*. In *International Symposium on Computer Architecture*, pages 609–621, 2016.

[206] M. F. Younis, T. J. Marlowe, A. D. Stoyen, and G. Tsai. *Statically safe speculative execution for real-time systems. IEEE Transactions on software engineering*, 25(5):701–721, 1999.

[207] Y. Zhang, Z.-J. M. Shen, and S. Song. *Exact algorithms for distributionally β-robust machine scheduling with uncertain processing times. INFORMS Journal on Computing*, 30(4):662–676, 2018.

[208] W. Zheng, L. Tang, and R. Sakellariou. *A priority-based scheduling heuristic to maximize parallelism of ready tasks for dag applications*. In *International Symposium on Cluster, Cloud and Grid Computing*, pages 596–605, 2015.

[209] S. Zouaoui, L. Boussaid, and A. Mtibaa. *Priority based round robin (PBRR) CPU scheduling algorithm. International Journal of Electrical and Computer Engineering*, 9(1):19–202, 2019.

This document was typeset on March 18, 2020 using class RC-BOOK α2.14 for LaTeX$_{2\epsilon}$. As of the time of writing this document, this class is not publicly available. Only members of The Distributed Group (TDG) and the Applied Computing Research Group (GCA) are allowed to typeset their documents using this class.