

## Repositório ISCTE-IUL

---

Deposited in *Repositório ISCTE-IUL*:

2023-07-12

Deposited version:

Accepted Version

Peer-review status of attached file:

Peer-reviewed

Citation for published item:

Freire, D. L., Frantz, R. Z., Roos-Frantz, F. & Basto-Fernandes, V. (2022). Integration process simulator: A tool for performance evaluation of task scheduling of integration processes. *Journal of Simulation*. 16 (6), 604-623

Further information on publisher's website:

10.1080/17477778.2022.2041989

Publisher's copyright statement:

This is the peer reviewed version of the following article: Freire, D. L., Frantz, R. Z., Roos-Frantz, F. & Basto-Fernandes, V. (2022). Integration process simulator: A tool for performance evaluation of task scheduling of integration processes. *Journal of Simulation*. 16 (6), 604-623, which has been published in final form at <https://dx.doi.org/10.1080/17477778.2022.2041989>. This article may be used for non-commercial purposes in accordance with the Publisher's Terms and Conditions for self-archiving.

---

### Use policy

Creative Commons CC BY 4.0

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in the Repository
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

---

# Integration Process Simulator: a tool for performance evaluation of task scheduling of integration processes

Daniela L. Freire · Rafael Z. Frantz ·  
Fabricia Roos-Frantz · V. Basto-Fernandes ·

[daniela@icmc.usp.br](mailto:daniela@icmc.usp.br)

[rafael@unijui.br](mailto:rafael@unijui.br)

[frantz@unijui.br](mailto:frantz@unijui.br)

[vbasto@iscte-iul.pt](mailto:vbasto@iscte-iul.pt)

**Abstract** The need to integrate efficiently the applications and services that compose the software ecosystem of companies increase with the scenarios of high data volume from Cloud Computing and Internet of Things. The improvement of the performance of runtime systems of integration platforms is directly related to the task scheduling strategies of integration processes. It still is a challenge to find the proper heuristic for a given integration process subject to high inbound data rates. In this article, we proposed a simulation tool for the study field of the Enterprise Application Integration, which implements different scheduling heuristics and allows the extraction of performance metrics. We simulate three heuristics of task scheduling on the execution of integration processes and evaluated the results by statistical tests.

**Keywords** Application integration · task scheduling algorithm · workflow scheduling · simulation tool · high volume data

---

Daniela L. Freire

Department of Exact Sciences and Engineering, Unijui University, Ijuí, RS, Brazil

Instituto Universitário de Lisboa (ISCTE-IUL) ISTAR-IUL, Lisbon, Portugal

*Present address:*

Institute of Mathematics and Computer Sciences University of Sao Paulo at Sao Carlos, SP,

Brazil E-mail: [danielalfreire@icmc.usp.br](mailto:danielalfreire@icmc.usp.br)

Rafael Z. Frantz, Fabricia Roos-Frantz

Unijui University, Department of Exact Sciences and Engineering, Ijuí, Brazil

V. Basto-Fernandes, [vbasto@iscte-iul.pt](mailto:vbasto@iscte-iul.pt)

Instituto Universitário de Lisboa (ISCTE-IUL) ISTAR-IUL, Lisboa, Portugal

[daniela@icmc.usp.br](mailto:daniela@icmc.usp.br)

[rafael@unijui.br](mailto:rafael@unijui.br)

[frantz@unijui.br](mailto:frantz@unijui.br)

[vbasto@iscte-iul.pt](mailto:vbasto@iscte-iul.pt)

## 1 Introduction

Enterprises are being impacted by the vast amount of data to be processed by services of Cloud Computing (CC) in the big-data era (Ritter et al., 2017), having to meet new requirements from the end-users, such as fast-response and low latency time (Fernández-Cerero et al., 2018). The increasing number of heterogeneous devices connected to the Internet of Things (IoT) produce a large volume of information, providing knowledge and create more business opportunities for enterprises. To cope with the big data, enterprise need to integrate the IoT with CC to create more value from the data generated and develop smart applications for the users (Aazam et al., 2016).

Enterprise Application Integration (EAI) is the field of study that provides methodologies, techniques, and tools for the design and implementation of integration processes (Frantz et al., 2016; Ritter et al., 2019). Integration platforms are tools that allow software engineers to design, implement, run, and monitor integration processes. The conceptual model of an integration process is a workflow composed of segments of atomic tasks connected by communication channels that desynchronise one task from another (Kanagaraj and Swamynathan, 2016). Data, wrapped in messages, are received from applications, processed by tasks in the workflows, and delivered to destination applications. Currently, several open-source integration platforms support integration patterns documented by Hohpe and Woolf (2004) and follow the Pipes-and-Filters architectural style (Alexander et al., 1977). Pipes depict message channels, and filters represent atomic tasks that implement a particular integration pattern to process messages. The component of an integration platform responsible for the execution of tasks is the runtime system, which manages jobs (composed of number of tasks) and distributes computational resources between them (Frantz et al., 2016), moreover, it performs scheduling of tasks, which is its primary function (Guo et al., 2015; Hilman et al., 2018). The pay-as-you-go charging model of Cloud Computing motivates a task scheduling concerned with the minimisation of costs (Freire et al., 2019b) and with the handling of large volumes of data from IoT (Shoukry et al., 2019), in the situation of currently emergent market demands for quality of software, flexibility and response times (Fan et al., 2018).

In the runtime system, threads represent the computational resources used to execute the tasks of an integration process. There are two main execution models for runtime systems found in the literature: process-based and task-based (Blythe et al., 2005; Boehm et al., 2011; Frantz et al., 2012; Alkhanak et al., 2016). 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 article, we address the task-based execution model, where a degradation of performance was detected in scenarios of high input rates of messages, caused by the accumulation of requests of execution of the initial tasks

in detriment of the others, due to the heuristic used in task scheduling, which is the First-in-First-Out (FIFO) (Frantz et al., 2012).

The choice of the heuristic to task scheduling in scenarios of high input rates of messages must increase the performance of the execution of integration processes by runtime systems. The heuristic must guarantee an appropriate response time to the message processing and proper harnessing computational resources. However, it is possible that a heuristic is optimal for an integration process and user-parameters, but is unappropriated for other integration processes and user-parameters. Usually, the most straightforward heuristics, such as FIFO, are adopted to achieve consistent results for all the scenarios, including both high and low input rates of messages. However, the performance may vary significantly depending on many factors because a single heuristic is rarely optimal for all cases, including different input rates of messages, workloads, type of message traffic, so on (Qureshi et al., 2011). Simulation-approach allows the evaluation of traditional and novel heuristics and measurement of their impact on the performance of the execution of integration processes. So, it is required to develop a simulation tool able to reproduce the implementation of integration processes subject to high input rates of messages. In our literature review, we found simulators for heuristics in the research field of Cloud Computing and Grid Systems, but in the EAI research field we found a single article that tests the FIFO heuristic (Haugg et al., 2019).

In this article, we propose the Integration Process Simulator (IPS), which allows the evaluation of various heuristics for task scheduling of integration processes carried out by runtime systems. We implemented the traditional FIFO heuristic and two new heuristics, Multi-queue Round Robin (MqRR) and Queue Priority (qPrior). Such proposals aim at fair allocation of threads to tasks of integration processes in scenarios of high volumes of data. However, the architecture of the IPS can incorporate other heuristic-algorithms. It is possible to simulate several scenarios varying messages input rates, the total workload of messages, the initial workload of messages, simulation time, and integration processes. Additionally, several performance metrics are provided by IPS, such as throughput, number of processed messages, and the number of remained messages. As a proof-of-concept for our simulator, we simulated the heuristic in the execution of three integration processes. The results of the simulation were statistically validated with ANOVA and Scott & Knoot tests. Thus, our simulator allows to analyze the adequacy of the runtime systems of integration platforms to tackle high workloads present in modern EAIs.

The rest of this article is organised as follows: Section 2 discusses the related work regarding simulators for heuristics of task scheduling; Section 3 describes characteristics of the scheduling in integration processes; Section 4 formulates the problem of task scheduling of integration processes; Section 5 presents the algorithms of our simulator; Section 6 presents a proof-of-concept to validate our simulator of heuristics; and, Section 7 presents our conclusions and future work.

## 2 Related Work

In this section, we discuss new tools for simulation of heuristics to task scheduling. Most of the articles that propose heuristics to task scheduling develop their algorithms in a programming language, without concerns on promoting

benchmarking standardized methods, frameworks and tools, see for instance, Riaz et al. (2018), Zhang et al. (2018a), and Gupta et al. (2019). Despite the lack of standardisation in the tools, we found some simulation tools, mainly regarding the task scheduling for Cloud Computing. Table 1 lists the tools found in related works and indicates the research field tackled by each tool. In the following paragraph, we detail these proposal.

GridSim is proposed by Buyya and Murshed (2002). It is a simulator build on top of SimJava library (Howell and McNab, 1998). It allows the modelling and simulation of heterogeneous Grid resources, users and application models, in the research field of Grid Systems. The simulator evaluates popular scheduling algorithms and new proposals. Some of the performance metrics provided are average budget spent by each user for processing jobs, average time at which the user experiment is terminated with varying number of users competing for resources, and number of jobs processed for each user when a varying number of users are competing for resources.

CloudSim, proposed by Calheiros et al. (2011), is based on SimJava (Howell and McNab, 1998) and GridSim (Buyya and Murshed, 2002) and focuses operation environment features to Infrastructure as a Service (IaaS), in Cloud Computing. It allows several virtual machines allocation and migration policies. This simulator allows the simulation of scheduling policies, such as space-shared and time-shared. Some of the performance metrics used are makespan, which is computed as the fastest completion time, and cost.

WorkflowSim, proposed by Kanagaraj and Swamynathan (2016), is a simulator built on top of CloudSim (Calheiros et al., 2011) that offers support for workflow scheduling and execution, in Cloud Computing research field. This simulator allows the comparison of new heuristics with the popular scheduling algorithms, such as Critical Path Algorithm, First Come First Serve, MaxMin, and MinMin. The performance metric provided is makespan.

TrueTime, proposed by Cervin and Årzén (2018), is a Matlab/Simulink-based simulation tool that has been developed at Lund University since 1999 (Eker and Cervin, 1999). It provides models of multi-tasking real-time kernels and networks used in simulation models for Networked Embedded Control Systems. TrueTime gathers metrics such as input-output latency and deadline overruns.

SCORE, proposed by Fernández-Cerero et al. (2018) is based on the Google Omega lightweight simulator (Schwarzkopf et al., 2013). It tests energy-efficiency, security, and scheduling strategies in Cloud Computing environments. This simulator evaluates heuristics, such as Random, Spread tasks to the maximum, Greedy minimising energy, Greedy minimising makespan, Spread tasks to the minimum, Spread tasks to the minimum with randomness. SCORE stores the performance metrics, such as the time a job waits in the queue until its first task is scheduled, time a job remains in the queue until it is scheduled, and percentage of scheduler utilisation on average.

Sphere, proposed by Fernández-Cerero et al. (2019), is based on SCORE (Cervin and Årzén, 2018) and follows the design and simplifications developed for the Google Omega simulator (Schwarzkopf et al., 2013). Sphere enables the simulation of large-scale Edge Computing scenarios by focusing resource-managing, scheduling and energy-efficiency policies. Some of the performance metrics provided are: makespan, time jobs wait for their first task to be scheduled, and time jobs wait for all their tasks to be scheduled.

GAME-SCORE, proposed by Fernández-Cerero et al. (2019) is the extension of SCORE (Cervin and Årzén, 2018) and follows the design pattern: a hybrid approach between discrete-event and multi-agent simulation. Its main aim is the simulation of energy-efficient IaaS on Cloud Computing. Some of the performance metrics provided are: makespan, time jobs wait for their first task to be scheduled, and time jobs remain in the system while all their tasks are scheduled.

**Table 1** Related works summary.

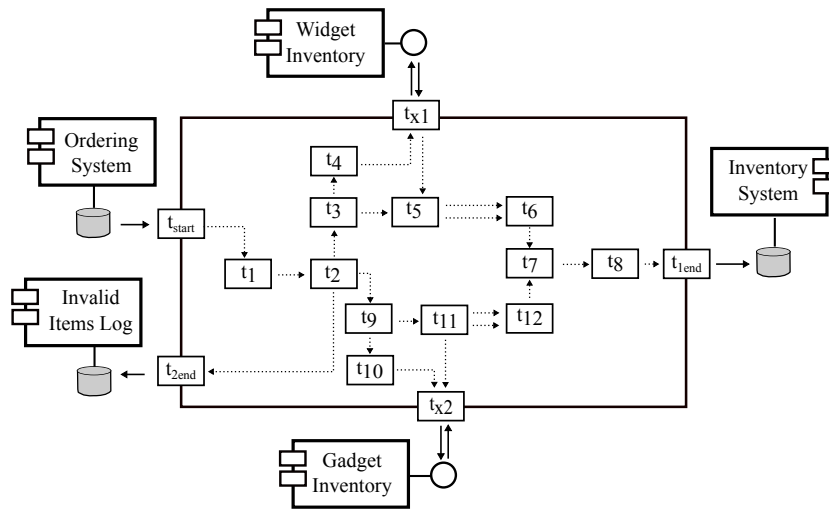
Ref.	Name	Research field
Buyya and Murshed (2002)	GridSim	Grid Systems
Calheiros et al. (2011)	CloudSim	Cloud Computing
Kanagaraj and Swamynathan (2016)	WorkflowSim	Cloud Computing
Cervin and Årzén (2018)	TrueTime	Embedded Control Systems
Fernández-Cerero et al. (2018)	SCORE	Cloud Computing
Fernández-Cerero et al. (2019)	Sphere	Edge Computing
Fernández-Cerero et al. (2019)	GAME-SCORE	Cloud Computing
[ Our Proposal ]	IPS	Enterprise Application Integration

### 3 Background

In this section, we present an integration process that use the integration patterns documented by Hohpe and Woolf (2004) and the architectural style Pipes-and-Filters (Alexander et al., 1977). We define the elements of the task scheduling and describe the task-based model, approached in this article.

An integration process is a computational program that connects applications, synchronising the exchange of data and functionalities amongst them. An example of an integration process, proposed by Hohpe (2005), is the «Processing Order», depicted in Figure 1. The «Processing Order» integration process connects five applications: «Ordering System», «Widget Inventory», «Gadget Inventory», «Invalid Items Log», and «Inventory System». «Ordering System» represents a source application that delivers the data of the new orders to the integration process. Each order is wrapped inside a «message» that contains data of order. A message with a new order is split into individual messages, each of which must contain only one item. Messages are 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. Generally, there are many jobs been processed at a particular point in time, so there are many job instances.

In the workflow of an integration process, a «path» is the set of uncoupled tasks connected by communication channels whereby an inbound message flows through, from source applications up to sink applications. An integration process is a «workflow» composed of several segments of tasks arranged sequentially, in parallel, or both. Parallel segments are parts of path which can be executed in parallel. The inbound message processing corresponds to the execution of all



**Fig. 1** Processing Order conceptual model.

tasks of the path through which the message flows from source application to the destination application.

A task implements an integration pattern, and every pattern represents an atomic and specific operation on message processing, such as transforming, filtering, splitting, joining, or routing. Depending on the integration pattern that a task implements, it can have one or more inputs and one or more outputs. There is an order of dependence for the tasks that determines their order of execution. So, a message can only be processed by a task after every predecessor tasks have processed this message. An outbound message of a task is written to the communication channel that connects this task with the next successor task in the path.

Runtime systems supply services to applications implemented on top of them (Appel, 1990). The «scheduler» is the key element that manages and orchestrates the activities of the elements of a runtime system, allowing the accomplishment of an inbound message processing. The computational resources responsible for the execution of the tasks are managed by the scheduler. In runtime systems of integration platforms, these resources are «threads» and, usually they are grouped in «thread pools» to avoid the creation of consecutive threads and allow quickly handle requests of tasks (Jeon and Jung, 2018). A thread is the smallest unit of a computational program that can be managed by the runtime system and is an abstraction of a piece of a physical and independent processing unit, or a central processing unit (CPU) core.

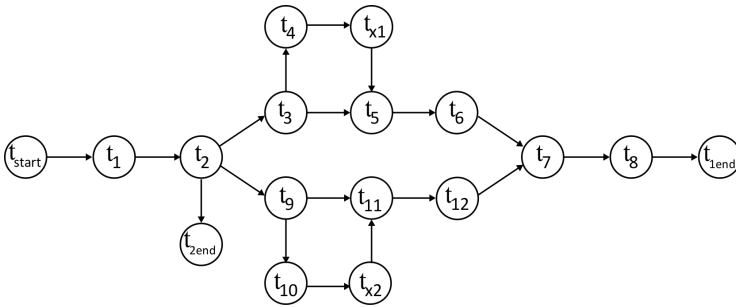
The execution model of runtime systems establishes how they must execute tasks and allocate threads during the processing of messages in an integration process (Freire et al., 2019a). In this article, we approach the task-based model, as it allows treating task instances, that is, tasks assigned to execution by a thread. In this model, a task is considered ready to be executed when there are messages in all communication channels that are inputs to that task. The request of the execution of a ready task is annotated in a waiting queue. Ready tasks wait in

the queue until there is an available thread to execute them. An available thread selects a task of the queue, following an execution policy, e.g. FIFO.

The scheduler creates, manages, and releases threads and can configure the pool by determining parameters, such as initial threads number, the maximum number of threads, 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 successor 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 of the tasks in the path. Tasks in sequential segments are executed sequentially, whereas tasks in parallel segments can be executed in parallel because there is no dependency between them.

#### 4 Problem Formulation

In this section, we represent the task scheduling problem of the «Processing Order» integration process by means of a Directed Acyclic Graph (DAG), and so we formulate the mathematical model of the problem. A DAG can represent the task scheduling model and the constraints of the execution of the tasks. 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  $T_i = \{t_{i,1}, t_{i,2}, \dots, t_{i,k}\}$  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. Every edge has a weight, which represents the waiting time of the task in the queue (Saifullah et al., 2013). The «Processing Order» integration process is represented by a DAG in Figure 2.



**Fig. 2** Processing Order represented in a DAG task model.

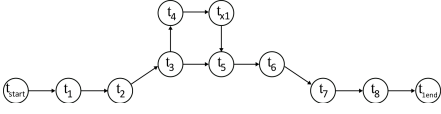
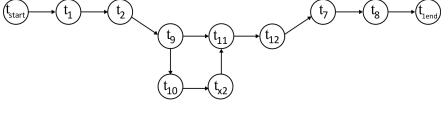
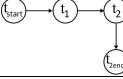
There are 17 nodes, which represent the 16 tasks:  $t_{start}$ ,  $t_{1end}$ ,  $t_{2end}$ ,  $t_1$ ,  $t_2$ ,  $t_3$ ,  $t_4$ ,  $t_5$ ,  $t_6$ ,  $t_7$ ,  $t_8$ ,  $t_9$ ,  $t_{10}$ ,  $t_{11}$ ,  $t_{12}$ ,  $t_{x_1}$ ,  $t_{x_2}$ , where  $t_{start}$  is a starting dummy node, and  $t_{1end}$  and  $t_{2end}$  are ending nodes. The nodes  $t_{x_1}$ ,  $t_{x_2}$  represent tasks that send and receive information to/from applications. There are 19 edges, which represent the 19 communication channels.

There is one input task represented by  $t_{start}$ , which has no predecessor task, and two output tasks represented by  $t_{1end}$  and  $t_{2end}$ , which have no successor



task. The nodes  $t_{x1}$  and  $t_{x2}$  represent the tasks that exchange messages with applications during runtime. 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 one of the inventories, «Widget Inventory» or «Gadget Inventory», or to none. The path for a unitary item that belongs to «Widget Inventory» is the task segment that equals to  $s_1 = \{t_{start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_7, t_8, t_{1end}\}$ ; for a unitary item that belongs to «Gadget Inventory» is the task segment that equals to  $s_2 = \{t_{start}, t_1, t_2, t_9, t_{10}, t_{x2}, t_{11}, t_{12}, t_7, t_8, t_{1end}\}$ ; and for a unitary item that does not belong to any inventory is the task segment that equals to  $s_3 = \{t_{start}, t_1, t_2, t_{2end}\}$ . Examples of tasks that can be executed in parallel in «Processing Order» integration process are  $[t_3, t_9]$ ,  $[t_4, t_{10}]$ ,  $[t_{x1}, t_{x2}]$ ,  $[t_5, t_{11}]$ ,  $[t_6, t_{12}]$ . Table 2 shows the paths and classification of segments of «Processing Order» integration process.

**Table 2** Processing Order path characterisation.

ID	Path	Segment	
		Sequential	Parallel
$s_1$		$t_{start}, t_1, t_2$ $t_6, t_7, t_8, t_{1end}$	$t_3, t_4, t_{x1}, t_5$
$s_2$		$t_{start}, t_1, t_2$ $t_{12}, t_7, t_8, t_{1end}$	$t_9, t_{10}, t_{x2}, t_{11}$
$s_3$		$t_{start}, t_1, t_{2end}$	

Makespan is a well-known by the integration community metric of performance and it is defined as the total execution time of the integration process for a given message (Canon and Jeannot, 2007; Chirkin et al., 2017). It corresponds to the total execution time of all the tasks of the path through which the message must flow for its complete processing. It includes all the times involved, such as the total CPU time, the waiting time of the tasks in a queue, and the waiting time of the task in request and response operations with external applications. We also assume that the range of the execution time of a task  $t_k$  is defined as  $[te_{t_k_{ini}}, te_{t_k_{fin}}]$ . In this article, we compute the total execution time of an integration process by the elapse of time between the start time of the first message entered the workflow  $ST_{m_1}$  and the end time of the last message leaving the workflow  $ET_{m_n p}$ .

Throughput is a performance measurement 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 measurements (Wood and Forman, 1971; Thakur and Kumar., 2018). In case of execution of integration processes, throughput corresponds to the number of messages processed per time unit and it is calculated as the division of the total number of processed messages  $np$  by the total execution time  $TET$ , cf. Equation 1.

$$Throughput = \frac{np}{ET_{m_{np}} - ST_{m_1}} \quad (1)$$

## 5 Proposal

In this section, we describe the Integration Process Simulator, a simulator of heuristics for task scheduling for integration processes. Our simulator implements the following heuristics: First-In-First-Out (FIFO), Multi-queue Round Robin (MqRR), and Query-Priority (qPrior). FIFO is a simple heuristic and is adopted in most integration platforms (Freire et al., 2019b), where messages are processed in the same order they enter the workflow. Round Robin (RR) heuristic is popularly known by its simplicity and considered an efficient and effective scheduling technique in computing (Zhang et al., 2018b), where messages are processed according to circular order. Here, we propose two novel heuristics: MqRR and qPrior, both based on RR. These heuristics seek to increase the performance of the execution of the integration processes in overload situation, facing the dynamic environment of the task scheduling of applications integration. An overload situation happens when the number of messages accumulated into communication channels is higher than the number of messages processed in a given time interval.

In scheduling by FIFO heuristic, there is a single task queue that maintains the instances of all the tasks. Tasks are maintained in a queue in descendant order of arrival time. In the head of the queue there is the task that has the longest arrival time, and in the tail of the queue there is the task that arrived most recently. Available threads recurrently poll the queue for task and, if there are pending tasks, the threads execute them in head-to-tail order.

In scheduling with MqRR heuristic, there are multiple task queues, and each queue maintains the instances of a task. Thus, the number of queues equals the number of tasks of the integration process. However, tasks belonging to parallel segments can be maintained in the same queue to execute them in parallel. Tasks are maintained in queues in decreasing order of arrival time, and available threads recurrently poll the queue for task and execute the existing tasks in order from head to tail. Threads recurrently poll the task queues in a circular order, executing a predefined number of tasks of every queue, each time.

With qPrior heuristic, also there are multiple task queues, but in this case, threads recurrently poll the queues following order of priority. A task that has more predecessor tasks has more priority in its execution.

### 5.1 Algorithms

Figure 6 in Appendices A.2 presents the flowchart of our simulator. **Start** is responsible for receiving the simulation parameters and calling the heuristics to use in the simulation. **Integration Process Profile** contains the integration

process information used as parameters in the other algorithms. **FIFO** provides task scheduling using the First-In-First-Out heuristic. **MqRR** uses Multi-Queue Round Robin heuristic. **qPrior** uses Queue Priority heuristic. **Allocate Thread** is responsible for managing and allocating the threads that execute the tasks of integration processes. **Next Task** is responsible for sending the tasks to the next task queue. **Operation** simulates the task execution. **Queue Add** adds tasks in a queue. **Queue Add Input** adds tasks in the first queue, simulating the arrival of messages in the integration process. The pseudo-codes of these algorithms are presented and detailed as follow.

### 5.1.1 Start

**Start** coordinates the simulation of integration processes, cf. Activity 1. It receives the number of simulations, the maximum duration of the simulation, maximum number of messages, initial number of inbound messages, the heuristic, and the number of tasks performed at a time (we called this number of preemption), in case of MqRR and qPrior heuristics. The last input parameter is the number of tasks that must be executed every time the threads check a queue. **Start** returns the throughput, the number of processed messages, and the number of remaining messages. Firstly, **Start** creates a vector for the starting time of messages, a vector for the final time of messages and the creation of task queues. If the heuristic is FIFO, it creates a single queue, whereas if the heuristic is RR, it creates a queue for every task of the integration process. First, it adds the first tasks in the queues, and then, it calls the algorithm that corresponds to the chosen heuristic. Finally, it calculates the throughput and records the metrics in a text file.

### 5.1.2 Integration Process Profile

**Integration Process Profile** is responsible for the profiles of the integration processes, cf. Activity 2. It receives the vector of identification of tasks, the vector of parallel tasks, the vector of the execution time of the tasks, the vector of operation of the tasks, the vector of the next tasks, and the vector of last tasks. It calculates the number of tasks of the integration process by measuring the length of the vector of identification of tasks.

### 5.1.3 First-In-First-Out

This algorithm carries out the task scheduling by the FIFO heuristic, cf. Algorithm 3. It receives the task queue, the maximum duration of the simulation, and the starting time of the first task in the queue. The algorithm starts by initialising the auxiliary variable: *totsize* that corresponds to queue total size. After, it recursively calls **Allocate Thread** activity to allocate threads to perform tasks until the simulation time expires or up to the moment there is no task in the queue.

### 5.1.4 Multi-Queue Round Robin

This algorithm carries out the task scheduling by the MqRR heuristic, cf. Algorithm 4. It receives the task queues, the maximum duration of the simulation,

---

**Activity 1 Start**

---

**Input:** Number of simulations: *numsimulation*  
**Input:** Maximum duration of the simulation: *maxduration*  
**Input:** Number of inbound messages: *messinbound*  
**Input:** Heuristic: *policy*  
**Input:** Number of tasks performed at a time: *preempttask*  
**Output:** Throughput : *throughput*  
**Output:** Number of processed messages: *messproc*  
**Output:** Number of remained messages: *messrem*

```

1: starttime[]                                ▷ Creates a starting time vector
2: endtime[]                                  ▷ Creates an end time vector
3: while count ≤ numsimulation do          ▷ Execution of the simulations
4:   if policy = FIFO then
5:     queues[0]                               ▷ Creates a single queue
6:   else
7:     for [i] = 1 to numtasks do
8:       queues[i]                             ▷ Creates a queue to each task
9:     end for
10:  end if
11:  for [i] = 1 to messinbound do          ▷ Creates first tasks in queues
12:    starttime[i] ← current.Time           ▷ Adds starting times of the messages
13:    if policy = FIFO then
14:      queues[0] ← add(1)
15:    else
16:      queues[1] ← add(1)
17:    end if
18:  end for
19:  switch policy do
20:    case FIFO
21:      simulationFile ← SimulationFIFO.txt
22:      Fifo()                                ▷ Executes FIFO heuristic
23:    case MqRR
24:      simulationFile ← SimulationMqRR.txt
25:      MqRR( )                               ▷ Executes MqRR heuristic
26:    case qPrior
27:      simulationFile ← SimulationqPrior.txt
28:      qPrior( )                             ▷ Executes qPrior heuristic
29:    count ← count + 1
30:  end while
31: messproc ← endtime.size();
32: if messproc > 0 then
33:   totmakespan ← (endtime[messproc] – starttime[1])  ▷ Calculates total total
   execution time
34:   throughput ← (messproc/(endtime[messproc] – starttime[1]))
35: end if
36: Record Archive(simulationFile)           ▷ Records metrics in a text file

```

---

**Activity 2 Integration Process Profile**

---

**Input:** Vector of identification of tasks : *IdTask*[ ]  
**Input:** Vector of parallel tasks : *ParallelTask*[ ]  
**Input:** Vector of the execution time of the tasks : *TimeExec*[ ]  
**Input:** Vector of operation of the tasks: *OperTask*[ ]  
**Input:** Vector of the next tasks : *NextTask*[ ]  
**Input:** Vector of last tasks: *LastTask*[ ]

```
1: NumTasks ← IdTask.length
```

---

**Algorithm 3** *First-In-First-Out*


---

**Input:** Task queue: *queues*[0]  
**Input:** Maximum duration of the simulation: *maxduration*  
**Input:** Starting time of the simulation: *start*

```

1: totsize  $\leftarrow$  queues[0].size                                 $\triangleright$  Initialises the queue size
2: while totsize > 0 and duration < maxduration do  $\triangleright$  Executes the tasks in the queue
3:   if queues[i]  $\neq$   $\emptyset$  then
4:     Allocate Thread(queues[0], totsize)                     $\triangleright$  Allocates threads to tasks in the queue
5:   end if
6:   duration  $\leftarrow$  current.Time - start                   $\triangleright$  Updates the duration of the simulation
7:   totsize  $\leftarrow$  queues[0].size                           $\triangleright$  Updates the task queue total size
8: end while

```

---

and the starting time 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. The algorithm starts by initialising two auxiliary variables: *totsize* and *preempt*. The former corresponds to queue total size and the latter to the number of tasks performed at a time (preemption). The algorithm checks the queues from the the queue of the first task until the queue of the last task, and it continues to check them in a circular order. When the queue size is smaller than *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 **Allocate Thread** activity to allocate threads to execute tasks, while there are tasks in the queue and the simulation duration is lower than the input parameter that stipulates maximum duration.

### 5.1.5 Queue Priority

This algorithm carries out the task scheduling by the qPrior heuristic, cf. Algorithm 5. It receives the task queues, the maximum duration of the simulation, and the starting 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. The algorithm starts by initialising two auxiliary variables: *totsize* and *preempt*. The former corresponds to queue total size and the latter the latter to the number of tasks performed at a time (preemption). The algorithm checks the queues from the highest priority task queue until the lowest priority task queue. When the queue size is smaller than *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 **Allocate Thread** activity to allocate threads to execute tasks, while there are tasks in the queue and the simulation duration is lower than the input parameter that stipulates maximum duration.

### 5.1.6 Allocate thread

**Allocate Thread** manages and allocates threads to the execution of tasks of a queue, cf. Activity6. It receives a task queue, the maximum duration of the simulation, the maximum number of messages, the number of tasks performed at a

**Algorithm 4** *Multi-Queue Round Robin*


---

**Input:** Task queues: *queues* [ ]  
**Input:** Maximum duration of the simulation: *maxduration*  
**Input:** Starting time of the simulation: *start*  
**Input:** Total number of tasks: *numtasks*  
**Input:** Number of tasks performed at a time: *preempttask*

```

1: totsize  $\leftarrow$  1 ▷ Initialises the total queue sizes
2: preemp  $\leftarrow$  preempttask ▷ Initialises the variable preempt
3: while totsize > 0 and duration < maxduration do ▷ Execution of tasks in the queue
4:   for [i] = 1 to numtasks do
5:     if queues[i]  $\neq$   $\emptyset$  then
6:       ▷ Checks whether there is preemption and compares with queue size
7:       if (preempt = 0) or (queues[i].size < preempt) then
8:         preempt  $\leftarrow$  queues[i].size
9:       else
10:        preempt  $\leftarrow$  preempttask
11:      end if
12:      Allocate Thread(queues[i], preempt) ▷ Allocates threads to tasks in the queue
13:    end if
14:  end for
15:  totsize  $\leftarrow$  0
16:  for [i] = 1 to numtasks do ▷ Updates the task queues total size
17:    totsize  $\leftarrow$  totsize + queues[i].size
18:  end for
19:  duration  $\leftarrow$  current.Time - start ▷ Updates the duration of the simulation
20: end while

```

---

**Algorithm 5** *Queue Priority*


---

**Input:** Task queues: *queues* [ ]  
**Input:** Maximum duration of the simulation: *maxduration*  
**Input:** Starting time of the simulation: *start*  
**Input:** Total number of tasks: *numtasks*  
**Input:** Number of tasks performed at a time: *preempttask*

```

1: totsize  $\leftarrow$  1 ▷ Initialises the total queue sizes
2: preemp  $\leftarrow$  preempttask ▷ Initialises the variable preempt
3: qprior  $\leftarrow$  numtasks ▷ Initialises the variable qprior
4: while totsize > 0 and duration < maxduration do ▷ Execution tasks in the queue
5:   for [i] = numtasks to 1 step-1 do ▷ Selects queue of higher priority
6:     if queues[i]  $\neq$   $\emptyset$  then
7:       qprior  $\leftarrow$  i
8:       i  $\leftarrow$  1
9:     end if
10:  end for
11:  if (preempt = 0) or (queues[qprior].size < preempt) then
12:    preempt  $\leftarrow$  queues[qprior].size
13:  else
14:    preempt  $\leftarrow$  preempttask
15:  end if
16:  Allocate Thread(queues[qprior], preempt) ▷ Allocates threads to tasks in the queue
17: end while
18: totsize  $\leftarrow$  0
19: for [i] = 1 to numtasks do ▷ Updates the task queues total size
20:  totsize  $\leftarrow$  totsize + queues[i].size
21: end for
22: duration  $\leftarrow$  current.Time - start ▷ Updates the duration of the simulation

```

---

time, and a vector containing the ending tasks. It starts by initialising the auxiliary variable *preempt*, and then it begins with the creation of a thread pool, which must be elastic, which allow variable size, and take advantage of the multicore CPU. **Allocate Thread** selects tasks in the queue, from head to tail, until it achieves the number of tasks equals to size of the preemption variable (*preempt*). If the queue size is smaller than the number of tasks, the algorithm assigns the queue size to size of the preemption variable. It submits the task to the thread pool and calls **Operation** activity, responsible for task operation. After the execution, **Allocate Thread** removes the task from the current queue. Then, it checks if the task belongs to the vector containing the ending tasks. If the task is not an end task and the heuristic used has multiple task queue, **Allocate thread** activity calls **Next queue** activity that is responsible by storing the task in the next queue, according to the logic of the integration process. Finally, it destroys the thread pool.

---

### Activity 6 *Allocate Thread*

---

**Input:** Task queue: *queues[i]*

**Input:** Maximum duration of the simulation: *maxduration*

**Input:** Maximum number of messages: *maxmessages*

**Input:** Number of tasks performed at a time: *preempt*

**Input:** Vector of last tasks: *LastTask[ ]*

```

1: preempt ← preempttask                                ▷ Initialises the variable preempt
2: Creates elastic thread pool
3: for [j] = 1 to preempt do
4:   if duration < maxduration then
5:     task ← queues.head                                ▷ Selects the task from the head of the queue
6:     if task ≠ null then
7:       Submits Operation(task) to a thread pool        ▷ Executes task
8:       for [j] = 0 to LastTask[ ].length do
9:         if task ≠ LastTask[j] then lasttask = 1
10:        end if
11:      end for
12:      if lasttask = 0 then
13:        Next queue(task)                                ▷ Stores task in next queue
14:      else
15:        endtime[] ← current.Time
16:      end if
17:      Removes task of the queue[i]                      ▷ Removes task of the queue
18:      Shutdown thread pool
19:    end if                                                ▷ Compares queue size with the preemption
20:    if queue[i].size < preempt then
21:      preempt ← queue[i].size
22:    else
23:      preempt ← preempttask
24:    end if
25:  else
26:    i ← preempt + 1
27:  end if
28:  duration ← current.Time – start
29:  if maxmessages > starttime.size() then
30:    Queue Input Add()                                ▷ Adds messages in first queue every 100 tasks executed
31:  else
32:    duration ← maxduration
33:  end if
34: end for

```

---

### 5.1.7 Next queue

**Next Task** is responsible for sending the task to its next queue, in case of heuristics that use multiple queue, cf. Activity 7. It receives a task, a vector containing the next task queues, and a vector containing the logic operations of the tasks. It starts by initialising the auxiliary variable *operleng* that corresponds to the length of the vector containing the logic operations of the tasks. Length of this vector equals to zero means that the task is sequential and must be sent to only one queue. Otherwise, the task must be sent to one or more queues, depending on logic operation. The logic operations of task can be **AND**, **OR\***, and **OR**. If the operation equals **AND**, the task must be sent to all the next queues. If the operation equals **OR\***, the task must be sent to only one next queue. The operation **OR** acts as both **AND** and **OR\***, i.e., the task can be sent to one or more queues. **Next Task** sends the task to the queues according to the vector of the next tasks, which contains the information regarding next task of integration process.

---

#### Activity 7 Next queue

---

**Input:** Task : *task*

**Input:** Vector of next tasks : *NextTask*[ ]

**Input:** Vector of operation tasks: *OperTask*[ ]

```

1: operleng ← OperTask[task].length    ▷ Initialises the auxiliary variable with vector size
2: switch operleng do
3:   case 0                                ▷ Sequential task
4:     nexttask ← NextTask[task][1]
5:     Queue Add(nexttask)                ▷ Adds the task in its next queue
6:   case 1 or 2                            ▷ Fork task
7:     if operleng = 2 then
8:       operrad = random OperTask[task][ ] ▷ Randomly selects one of the operations
9:     else
10:      operrad = OperTask[task]           ▷ Selects the task operation
11:    end if
12:    if operrad = or then                 ▷ Can choose one of the next task
13:      nexttask = random NextTask[task][1] ▷ Randomly selects one of the next task
14:      Queue Add(nexttask)                ▷ Adds the task in its next queue
15:    else
16:      if operrad = and then               ▷ Send to all are next tasks
17:        for [i] = 1 to NextTask[task].length do
18:          nexttask = NextTask[task][i]
19:          Queue Add(nexttask)            ▷ Adds the task in next queue
20:        end for
21:      end if
22:    end if

```

---

### 5.1.8 Operation

**Operation** simulates the processing of messages by a task, cf. Activity 8. It receives a task and the vector of tasks execution time. It randomly selects an execution time contained in this vector and waits during this time. The operation of processing message can be to split, translate, filter, aggregate, route, so on.



---

**Activity 8** *Operation*

---

**Input:** Task : *task***Input:** Vector of execution time tasks : *TimeExec*[ ]

- 1: *time*  $\leftarrow$  random *TimeExec*[*task*][ ]  $\triangleright$  Randomly selects one of the execution time to the task
  - 2: Waits *time*  $\triangleright$  Simulates the execution of the task
- 

**5.1.9** *Queue Add*

**Queue Add** is responsible for adding a task in a queue, cf. Activity 9. It receives a task and the vector of parallel tasks. The latter input indicates when a task can be executed in parallel with another task. **Queue Add** checks the heuristic, and if the heuristic is FIFO, it adds the task in the FIFO queue, otherwise, it checks the vector of parallel tasks. If there is a parallel task to the task, it adds the task in the queue of parallel task, otherwise, it adds the task in its queue.

---

**Activity 9** *Queue Add*

---

**Input:** Task : *task***Input:** Vector of parallel tasks : *ParallelTask*[ ]

- 1: **if** *policy* = FIFO **then**
  - 2:     *queues*[0]  $\leftarrow$  *add(task)*  $\triangleright$  Adds tasks in FIFO queue
  - 3: **else**  $\triangleright$  Checks if the tasks are parallel
  - 4:     **if** *ParallelTask*[*task*]  $\neq$  0 **then**
  - 5:         *parallel* = *ParallelTask*[*task*]
  - 6:         *queues*[*parallel*]  $\leftarrow$  *add(task)*  $\triangleright$  Adds tasks in the queue of the parallel task
  - 7:     **else**
  - 8:         *queues*[*task*]  $\leftarrow$  *add(task)*  $\triangleright$  Adds tasks in the respective queue
  - 9:     **end if**
  - 10: **end if**
- 

**5.1.10** *Queue Input Add*

**Queue Input Add** is responsible for adding first tasks in the queue, cf. Activity 10. It receives the number of input tasks, then, it checks the heuristic, and if the heuristic is FIFO, it adds the tasks in the FIFO queue. Otherwise, it adds the task in the queue of the first task.

**6 Validation**

In this section, we use the IPS simulator to compare the performance of the heuristics FIFO, MqRR, and qPrior in the execution of integration processes under high workloads (above 1,000,000 messages). We used performance metrics such as throughput, the number of processed messages and number of remained messages. The integration processes used in the simulations are Processing Order (IP1), Huelva's County Council (IP2), and Real Estate (IP3). Their profiles are described in Appendices A.1. The Processing Order problem is a classic example

**Activity 10** *Queue Input Add***Input:** Number of input tasks : *numinputtask*


---

```

1: for [i] = 1 to numinputtask do                                ▷ Creates first tasks in the queue
2:   starttime[] ← current.Time
3:   if policy = FIFO then
4:     queues[0] ← add(1)
5:   else
6:     queues[1] ← add(1)
7:   end if
8: end for

```

---

of an integration process, introduced by Hohpe (2005), which conceptual model is depicted in Figure 1. The Huelva’s County Council problem is a real-world integration process (Frantz et al., 2016) that consists of the automatization of the user registration into a central repository (Huelva, Spain). The Real State problem is a real-world integration process (Freire et al., 2019a) that consists of the automatization of the real state tax management system in the city of Ijuí, (Brazil).

Our simulation is classified as termination simulation because the output is a function of the initial conditions, in which we followed a protocol based on Jedlitschka and Pfahl (2005), Wohlin et al. (2012), and Basili et al. (2007), with procedures for controlled experiments in the field of software engineering. The literature suggests that statistical analysis must be done of the data from simulations regarding performance (Georges et al., 2007), because this type of simulation deals with non-determinism in computational systems (Frantz et al., 2011). We used the ANOVA test to verify the influence of random factors, called errors, in the measurements of the dependent variables. The comparison of averages by the Scott & Knott test groups the averages of the dependent variables to check if there is a statistical difference between the results of the heuristics. The source code and data sets used in the simulations is publicly available for download <sup>1</sup>.

### 6.1 Research Question and Hypothesis

This experiment aims to answer the following research question:

RQ: Is it possible to extract performance metrics by IPS that allow the evaluation of heuristics of task scheduling in the execution of an integration process under high workloads?

Our hypothesis to this research question is that:

H: It is possible to simulate heuristics of task scheduling of an integration process with IPS simulator and to find throughput, number of processed messages, and performance metrics in the execution of an integration process under high workloads.

---

<sup>1</sup> <https://github.com/gca-research-group/Simulation-qPrior>

## 6.2 Variables

The independent variables are:

**Heuristic.** The heuristic used to task scheduling. The values tested for this variable were FIFO, MqRR, and qPrior.

**Integration process.** The conceptual model of the integration process. The value tested for this variable were IP1, IP2, and IP3.

**Duration.** The time interval that the algorithm keeps performing. The value tested for this variable was 60 seconds.

**Total workload.** The total number of inbound messages. The value tested for this variable was 2,000,000 messages.

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

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

The dependent variables are:

**Throughput.** This variable corresponds to the average of processed messages by millisecond, cf. Equation 1.

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

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

## 6.3 Environment and Supporting Tools

The simulations were carried out 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. The programming language Java, version 8.0 update 152, was used to implement and execute the IPS simulator. The Genes (Cruz, 2006) software, version 2015.5.0, was used to process the descriptive statistic, ANOVA test, and Scott & Knoot test were used for the performance metrics in this study.

## 6.4 Execution and Data Collection

The experiment was conducted using the IPS simulator, which simulates the execution of the IP1, IP2, IP3 integration processes. Table 3 describes the main differences between the integration processes tested.

The simulation starts with a workload of 1,000 inbound messages 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 MqRR and qPrior heuristics, the preemption variable was set to 100 tasks. We configured the simulation time to 60 seconds, after this time, the simulation is stopped. Then, the simulator collects the dependent variables and stores them in a text file. Afterwards, we handled and analysed the data, and then applied the statistical tests.

**Table 3** Integration process characterisation.

ID	DAG	Nodes number	Starting nodes	Ending nodes
IP1		17	1	2
IP2		19	2	2
IP3		17	3	2

The literature suggests that 20-30 executions are sufficient to obtain a population average (Sargent, 2013). In our experiment, each heuristic was simulated 25 times for each integration process, resulting in the 225 different scenarios, summarised in Table 4.

**Table 4** Scenarios of the simulations.

Heuristics	FIFO, MqRR, qPrior	3
Integration Process	IP1, IP2, IP3	1
Duration	60 seconds	1
Total workload	2,000,000	1
Initial workload	1000	1
Rate of inbound messages	1000	1
Repetitions		25
<b>Scenarios</b>	<b>3 x 3 x 1 x 1 x 1 x 1 x 1 x 25</b>	<b>225</b>

### 6.5 Results

The average values of dependent variables obtained in the 25 independent runs of the simulation for every type of heuristic are shown in scatter charts, cf. Figures: 3, 4, and 5. In these charts, the x-axis represents the heuristics and the

y-axis represents the average number of throughput, average number of processed messages, and average number of remained messages, respectively.

In the simulation of IP1, the throughput achieved (in messages per milliseconds) was 8.44 mess/ms with FIFO; 9,455.17 mess/ms with MqRR; and 28,076.67 mess/ms with qPrior. The average number of processed messages was equal to 502.40 messages with FIFO; 22,607 messages with MqRR; and 199,800 messages with qPrior. Lastly, the average number of remained messages was equal to 1,999,498 messages with FIFO; 1,977,393 messages with MqRR; and 1,800,200 messages with qPrior. So, qPrior was most efficient and FIFO was least efficient.

In the simulation of IP2, the throughput achieved (in messages per milliseconds) was 64.26 mess/ms with FIFO; 10,250.02 mess/ms with MqRR; and 27,001.52 mess/ms with qPrior. The average number of processed messages was equal to 3,821.52 messages with FIFO; 25,515 messages with MqRR; and 199,800 messages with qPrior. Lastly, the average number of remained messages was equal to 1,996,178 messages with FIFO; 1,974,485 messages with MqRR; and 1,800,200 messages with qPrior. So, qPrior was most efficient and FIFO was least efficient.

In the simulation of IP3, the throughput achieved (in messages per milliseconds) was 8.49 mess/ms with FIFO; 9,381.09 mess/ms with MqRR; and 27,001.52 mess/ms with qPrior. The average number of processed messages was equal to 506.28 messages with FIFO; 22,607 messages with MqRR; and 199,800 messages with qPrior. Lastly, the average number of remained messages was equal to 1,999,494 messages with FIFO; 1,977,393 messages with MqRR; and 1,800,200 messages with qPrior. So, qPrior was most efficient and FIFO was least efficient.

Table 5 shows the analysis of variance for IP1, an average square of the throughput, an average square was 509,9275,548 for the heuristics and 1,704,696 for error. The overall average was equal to 12,513.42, and the coefficient of variation was 10.43 %. In the analysis of variance of the number of processed messages, an average square was 298,355,654,464 for the heuristics and 3,415 for error. The overall average was equal to 1,925,696.70, and the coefficient of variation was 3.03 %. Lastly, in the analysis of variance of the number of remained messages, an average square was 298,355,654,464 for the heuristics and 3,415 for error. The overall average was equal to 74,303.29, and the coefficient of variation was 0.07 %. So, there was a relation between the treatments (heuristics) and the result of an experiment (performance).

Table 6 shows the analysis of variance for IP2, an average square of the throughput, an average square was 4,638,534,208 for the heuristics and 2,231,170.0 for error. The overall average was equal to 12,451, and the coefficient of variation was 11.99 %. In the analysis of variance of the number of processed messages, an average square was 288,556,580,878 for the heuristics and 182,858 for error. The overall average was equal to 1,923,621.32, and the coefficient of variation was 2.22 %. Lastly, in the analysis of variance of the number of remained messages, an average square was 288,556,580,878 for the heuristics and 182,858 for error. The overall average was equal to 76,378.68, and the coefficient of variation was 0.55 %. So, there was a relation between the treatments and the result of an experiment.

Table 7 shows the analysis of variance for IP3, an average square of the throughput, an average square was 4,695,619,987 for the heuristics and 3,282,183 for error. The overall average was equal to 12,130.36, and the coefficient of variation was 14.93 %. In the analysis of variance of the number of processed messages, an average square was 298,348,495,903 for the heuristics and 2,398 for error. The

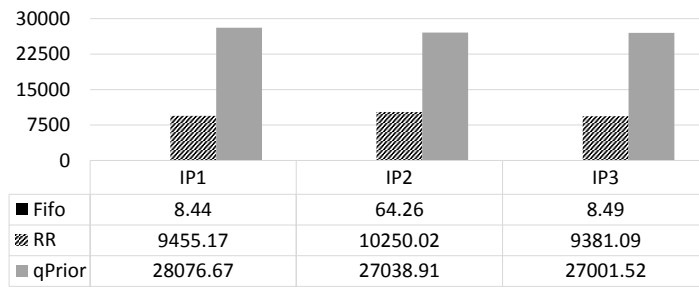


Fig. 3 Average throughput

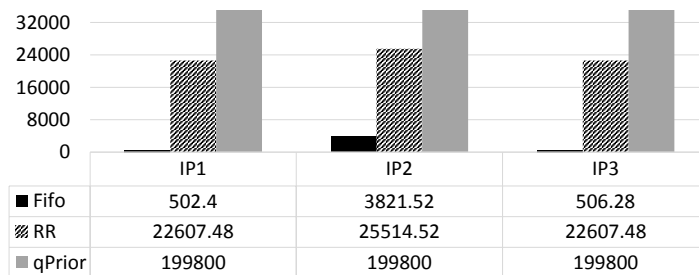


Fig. 4 Average number of processed messages

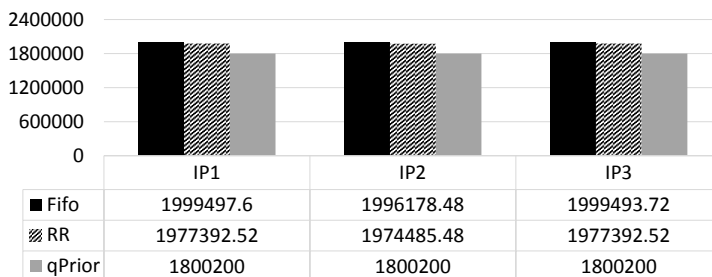


Fig. 5 Average number of remained messages

overall average was equal to 1,925,695.41, and the coefficient of variation was 2.54 %. Lastly, in the analysis of variance of the number of remained messages, an average square was 298,348,495,903 for the heuristics and 2,398 for error. The overall average was equal to 74,304, and the coefficient of variation was 6.59 %. So, there was a relation between the treatments and the result of an experiment.

The results of the Scott & Knott test are shown in Tables 8, 9, and 10, for IP1, IP2, and IP3, respectively. The heuristics identified in the first column. For each dependent variable, there is a column for the average and a column for the group of Scott & Knott. There were three groups: «a», «b», and «c». The «a» group refers to the heuristic with the highest average, the «b» group refers to the

**Table 5** ANOVA test for IP1.

Sources of variation	Degree of freedom	Average square		
		Throughput	Processed messages	Remained messages
Heuristics	2	5099275548 †	298355654464 †	298355654464 †
Error	72	1704696	3415	3415
Total	74			
Overall average		12513.42	1925696.70	74303.29
Coefficient of variation (%)		10.43	3.03	0.07

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

**Table 6** ANOVA test for IP2.

Sources of variation	Degree of freedom	Average square		
		Throughput	Processed messages	Remained messages
Heuristics	2	4638534208 †	288556580878 †	288556580878 †
Error	72	2231170	182858	182858
Total	74			
Overall average		12451	1923621.32	76378.68
Coefficient of variation (%)		11.99	2.22	0.55

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

**Table 7** ANOVA test for IP3.

Sources of variation	Degree of freedom	Average square		
		Throughput	Processed messages	Remained messages
Heuristics	2	4695619987 †	298348495903 †	298348495903 †
Error	72	3282183	2398	2398
Total	74			
Overall average		12130.36	1925695.41	74304
Coefficient of variation (%)		14.93	2.54	6.59

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

heuristic with the second-highest average, and «c» refers to the heuristic with the lowest average.

The results of the Scott & Knoot test for IP1, regarding throughput, show that FIFO was in «c» group with the lowest average 8.44 mess/ms, MqRR in «b» group with the average 9,455.17, and qPrior in «a» group with the highest average 28,076,67 mess/ms; regarding the number of processed messages, FIFO was in «c» group with the lowest average 502 messages, MqRR in «b» group with the average 22,607 messages, and qPrior in «a» group with the highest average 199,800 messages; and regarding the number of remained messages, FIFO was in «a» group with the highest average 1,999,498 messages, MqRR in «b» group with the average 1,977,393, and qPrior in «c» group with the lowest average 1,800,200, cf. Table 8. So, there was statistical difference between the three treatments.

The results of the Scott & Knoot test for IP2, regarding throughput, show that FIFO was in «c» group with the lowest average 64.26 mess/ms, MqRR in «b» group with the average 10,250.02, and qPrior in «a» group with the highest average 27,038.91 mess/ms; regarding the number of processed messages, FIFO was in «c» group with the lowest average 3,821 messages, MqRR in «b» group with the average 25,515 messages, and qPrior in «a» group with the highest average 199,800 messages; and regarding the number of remained messages, FIFO was in «a» group with the highest average 1,996,178 messages, MqRR in «b» group with

the average 1,974,485, and qPrior in «c» group with the lowest average 1,800,200, cf. Table 9. So, there was statistical difference between the three treatments.

The results of the Scott & Knott test for IP3, regarding throughput, show that FIFO was in «c» group with the lowest average 8.49 mess/ms, MqRR in «b» group with the average 9,381.09, and qPrior in «a» group with the highest average 27,001,52 mess/ms; regarding the number of processed messages, FIFO was in «c» group with the lowest average 506 messages, MqRR in «b» group with the average 22,607 messages, and qPrior in «a» group with the highest average 199,800 messages; and regarding the number of remained messages, FIFO was in «a» group with the highest average 1,999,4948 messages, MqRR in «b» group with the average 1,977,393, and qPrior in «c» group with the lowest average 1,800,200, cf. Table 10. So, there was statistical difference between the three treatments.

**Table 8** Scott & Knott test for IP1.

Heuristic	Throughput		Processed messages		Remained messages	
	Average	Group	Average	Group	Average	Group
FIFO	8.44	c	502	c	1999498	a
MqRR	9455.17	b	22607	b	1977393	b
qPrior	28076.67	a	199800	a	1800200	c

*Error level of 5% by the Scott & Knott model.*

**Table 9** Scott & Knott test for IP2.

Heuristic	Throughput		Processed messages		Remained messages	
	Average	Group	Average	Group	Average	Group
FIFO	64.26	c	3821	c	1996178	a
MqRR	10250.02	b	25515	b	1974485	b
qPrior	27038.91	a	199800	a	1800200	c

*Error level of 5% by the Scott & Knott model.*

**Table 10** Scott & Knott test for IP3.

Heuristic	Throughput		Processed messages		Remained messages	
	Average	Group	Average	Group	Average	Group
FIFO	8.49	c	506	c	1999494	a
MqRR	9381.09	b	22607	b	1977393	b
qPrior	27001.52	a	199800	a	1800200	c

*Error level of 5% by the Scott & Knott model.*

## 6.6 Summary

In the simulation of IP1, the best average throughput was 28076.67 mess/ms with qPrior. The worst-case scenario was the throughput of 8.44 mess/ms, using FIFO. The qPrior heuristic was one that managed to process more messages,



reaching the average of 199,800 messages and the number of remained messages was 1,800,200 messages. The worst case of the processed messages happened using FIFO, in which only 502.40 messages were processed and 1,999,498 messages remained unprocessed.

In the simulation of execution of the IP2, the best average throughput was 27,038.91 mess/ms with qPrior. The worst-case scenario was the throughput of 64.26 mess/ms, using FIFO. The qPrior heuristic was one that managed to process more messages, reaching the average of 199,800 messages and the number of remained messages was g 1,800,200 messages. The worst case of the processed messages happened using FIFO, where only 3,821.52 messages were processed and 1,996,178 messages remained unprocessed.

In the simulation of execution of the IP3, the best average throughput was 27,001.52 mess/ms with qPrior. The worst-case scenario was the throughput of 8.49 mess/ms, using FIFO. The qPrior heuristic was one that managed to process more messages, reaching the average of 199,800 messages and the number of remained messages was 1,800,200 messages. The worst case of the processed messages happened using FIFO, in which only 506.28 messages were processed and 1,999,494 messages remained unprocessed.

The heuristics MqRR and qPrior provide stability in the average number of processed messages in three integration processes evaluated in this study. This stability is indicative that these heuristics allow predictability of number of processed messages, independently of the integration process. The heuristic qPrior resulted in the best average throughput in all integration processes, and FIFO resulted in the worst in all integration processes.

The use of different heuristics generates a significant difference in the average values of the throughput, number of processed messages, and number of remained messages, cf. Table 5. Low coefficients of variation indicate the adequacy and reliability of the simulations. The results of the Scott & Knott averages comparison test show that there were three different groups for throughput, number of processed messages, and number of remained messages, cf. Table 8. The statistical differences between the three heuristics resulting from the simulations, provide support to answer our research question:

- **RQ:** Is it possible to extract performance metrics by IPS that allow the evaluation of heuristics of task scheduling in the execution of an integration process under high workloads?

By the experiment, we confirm our hypothesis for the research question:

- **H:** It is possible to simulate heuristics of task scheduling of an integration process with IPS simulator and to find throughput, number of processed messages, and performance metrics in the execution of an integration process under high workloads.

## 6.7 Threats to Validity

Threats to validity are present in any empirical research (Cruzes and ben Othman, 2017), in addition to that, there are some of them, specific to optimisation studies (Wohlin et al., 2012). Next, we evaluated the factors that could influence the results of the experiments and suggest how to mitigate them.

### 6.7.1 Constructor Validity

Constructor validity discusses whether the planning and execution of the study are adequate and able to answer the research question. We planned the experiment according to procedures from empirical software engineering (Jedlitschka and Pfahl, 2005; Basili et al., 2007; Wohlin et al., 2012). Firstly, we defined our research question, formulated our hypothesis, and defined the independent and dependent variables. After, we provided information about the execution environment, supporting tools, execution algorithms and data collection. Then, we performed our simulation in seventy-five different scenarios and used statistical techniques to evaluate the results.

### 6.7.2 Conclusion Validity

As reported by Wohlin et al. (2012), 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”. We used statistical techniques 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.

### 6.7.3 Internal Validity

Internal validity aims to ensure that the treatment leading to the outcome, mitigating effects of other uncertain or not measured factors (Feldt and Magazinius, 2010). In our case, the treatments are the heuristics. All our experiments were performed in the same machine, on security mode, with minimal features and disconnected from the Internet, in order to minimise their influence on the execution time of the algorithms. We implemented our algorithms in Java and run a few executions before starting the experiments, to let the virtual machine to stabilise and eventually perform code optimisation (Pinto et al., 2014). Additionally, we accurately inspected the procedures and used statistical tests to validate the used performance metrics. .

### 6.7.4 External Validity

External validity focuses on the generalisation of the results outside the scope of our study (Feldt and Magazinius, 2010). This study is generalised for integration platforms that adopt the integration patterns by Hohpe and Woolf (2004), the style Pipes-and-Filters, and task-based model. We reported this study following a practical guideline (Wohlin et al., 2012), so that exact repetition is possible. The experiment is valid to test other parameters, such as integration processes, message arrival rate, and simulation duration. As future work, we intend to experiment with other integration processes in order to further evaluate the generalisation of the results.

## 7 Conclusion

Many business possibilities exist for enterprises in the big-data era, but to deal with a large volume of data, and extracting exact information on the business is

still a challenge (Shoukry et al., 2019). Enterprise Application Integration (EAI) is a field of study that faces the complex task of integrating these data, providing new methodologies, techniques, and tools for the design and implementation of integration processes (Frantz et al., 2016; Ritter et al., 2019).

In this article, we proposed a tool for simulation of task scheduling algorithms, the Integration Process Simulator (IPS). Using IPS we evaluated three heuristics: First-In-First-Out (FIFO), Multi-queue Round Robin (MqRR), and Queue Priority (qPrior), but other scheduling algorithms might easily be implemented in IPS. Performance metrics, such as throughput, number of processed messages, and number of remained messages can be obtained from IPS. It also allows to configure the time of simulation, initial workload of messages, total workload of messages, rate of inbound messages, and the integration process. We tested the simulator with three benchmark integration processes. The results of the simulations showed that qPrior was the best heuristic, providing the highest throughput in high workloads, on the other hand, FIFO was the worst heuristic, providing the lowest throughput. The statistical analysis confirmed the above and showed that there is a significant difference in performance metrics when task scheduling is performed with FIFO, MqRR, and qPrior. In the future, we intend to include more metrics and more heuristics into the simulator, and to extend the simulations, testing other integration processes and comparing other workload parameters. Regarding the research questions of the experiment: *our simulator is able to evaluate heuristics used to task scheduling of integration processes. For the three integration processes tested, the qPrior heuristic was the one that processed more messages per time unit.*

## Acknowledgements

This work was supported by the Coordination for the Improvement of Higher Education Personnel (CAPES), under grants 88881.119518/2016-01 and 88881.136207/2017-01; and, by the Research Support Foundation of the State of Rio Grande do Sul (FAPERGS) under grant 17/2551-0001206-2.

## References

- Aazam M, Huh EN, St-Hilaire M, Lung CH, Lambadaris I (2016) Cloud of Things: Integration of IoT with Cloud Computing, Springer International Publishing, pp 77–940
- Alexander C, Ishikawa S, Silverstein M (1977) A pattern language: towns, buildings, construction. Oxford University Press
- Alkhanak EN, Lee SP, Rezaei R, Parizi RM (2016) Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *Journal of Systems and Software* 113:1–26
- Appel AW (1990) A runtime system. *LISP and Symbolic Computation* 3(4):343–380
- Basili VR, Rombach D, Kitchenham KSB, Selby D, Pfahl RW (2007) *Empirical Software Engineering Issues*. Springer Berlin/Heidelberg

- Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K (2005) Task scheduling strategies for workflow-based applications in grids. In: IEEE International Symposium on Cluster Computing and the Grid (CCGrid), vol 2, pp 759–767
- Boehm M, Habich D, Preissler S, Lehner W, Wloka U (2011) Cost-based vectorization of instance-based integration processes. *Information Systems* 36(1):3–29
- Buyya R, Murshed M (2002) Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency Computation Practice and Experience* 14(13–15):1175–1220
- Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R (2011) Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software - Practice and Experience* 41(1):23–50
- Canon LC, Jeannot E (2007) A comparison of robustness metrics for scheduling DAGs on heterogeneous systems. In: *Int. Conf. on Cluster Computing (IEEE Cluster)*, pp 558–567
- Cervin A, Arzén KE (2018) Truetime: Simulation tool for performance analysis of real-time embedded systems. In: *Model-Based Design for Embedded Systems*, CRC Press, pp 169–200
- Chirkin AM, Belloum ASZ, Kovalchuk SV, Makkes MX, Melnik MA, Visheratin AA, Nasonov DA (2017) Execution time estimation for workflow scheduling. *Future Generation Computer Systems* 75:376–387
- Cruz CD (2006) Programa Genes: estatística experimental e matrizes. Editora Universidade Federal de Viçosa
- Cruzes DS, ben Othman L (2017) Threats to validity in empirical software security research. In: *Empirical Research for Sof. Security*, pp 295–320
- Eker J, Cervin A (1999) A matlab toolbox for real-time and control systems co-design. In: *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. (RTCSA)*, IEEE, pp 320–327
- Fan K, Zhai Y, Li X, Wang M (2018) Review and classification of hybrid shop scheduling. *Production Engineering* 12(5):597–609
- Feldt R, Magazinius A (2010) Validity threats in empirical software engineering research—an initial survey. In: *Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, pp 374–379
- Fernández-Cerero D, Jakóbič A, Fernández-Montes A, Kołodziej J (2019) GAME-SCORE: Game-based energy-aware cloud scheduler and simulator for computational clouds. *Simulation Modelling Practice and Theory* 93:3–20
- Fernández-Cerero D, Fernández-Montes A, Jakóbič A, Kołodziej J, Toro M (2018) Score: Simulator for cloud optimization of resources and energy consumption. *Simulation Modelling Practice and Theory* 82:160–173
- Fernández-Cerero D, Fernández-Montes A, Ortega FJ, Jakóbič A, Widlak A (2019) Sphere: Simulator of edge infrastructures for the optimization of performance and resources energy consumption. *Simulation Modelling Practice and Theory*
- Frantz RZ, Corchuelo R, Arjona JL (2011) An efficient orchestration engine for the cloud. In: *International Conference on Cloud Computing Technology and Science (CloudCom)*, pp 711–716
- Frantz RZ, Corchuelo R, Molina-Jiménez C (2012) A proposal to detect errors in enterprise application integration solutions. *Journal of Systems and Software*

- 85(3):480–497
- Frantz RZ, Corchuelo R, Roos-Frantz F (2016) On the design of a maintainable software development kit to implement integration solutions. *Journal of Systems and Software* 111:89–104
- Freire DL, Frantz RZ, Roos-Frantz F (2019a) Towards optimal thread pool configuration for run-time systems of integration platforms. *International Journal of Computer Applications in Technology* XX(in-press):1–18
- Freire DL, Frantz RZ, Roos-Frantz F, Sawicki S (2019b) Survey on the run-time systems of enterprise application integration platforms focusing on performance. *Software: Practice and Experience* 49(3):341–360
- Georges A, Buytaert D, Eeckhout L (2007) Statistically rigorous java performance evaluation. *ACM SIGPLAN Notices* 42(10):57–76
- Guo F, Yu L, Tian S, Yu J (2015) A workflow task scheduling algorithm based on the resources' fuzzy clustering in cloud computing environment. *International Journal of Communication Systems* 28(6):1053–1067
- Gupta I, Gupta S, Choudhary A, Jana PK (2019) A hybrid meta-heuristic approach for load balanced workflow scheduling in iaas cloud. In: *International Conference on Distributed Computing and Internet Technology(ICDCIT)* , pp 73–89
- Haugg IG, Frantz RZ, Roos-Frantz F, Sawicki S, Zucolotto B (2019) 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
- Hilman MH, Rodriguez MA, Buyya R (2018) Multiple workflows scheduling in multi-tenant distributed systems: A taxonomy and future directions. *ACM Computing Surveys* 1(1):1–33
- Hohpe G (2005) Your coffee shop doesn't use two-phase commit [asynchronous messaging architecture]. *IEEE software* 22(2):64–66
- Hohpe G, Woolf B (2004) *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional
- Howell F, McNab R (1998) Simjava: A discrete event simulation package for java with applications in computer systems modelling. *Proceedings of the First International Conference on Web-based Modelling and Simulation* pp 51–56
- Jedlitschka A, Pfahl D (2005) Reporting guidelines for controlled experiments in software engineering. In: *International Symposium on Empirical Software Engineering (ESEM)*, pp 95–104
- Jeon S, Jung I (2018) Experimental evaluation of improved IoT middleware for flexible performance and efficient connectivity. *Ad Hoc Networks* 70:61–72
- Kanagaraj K, Swamynathan S (2016) A study on performance of dominant scheduling algorithms on standard workflow systems in cloud. In: *International Conference on Informatics and Analytics (ICIA)*, pp 45:1–45:6
- Pinto G, Castor F, Liu YD (2014) Understanding energy behaviors of thread management constructs. In: *ACM SIGPLAN Notices*, vol 49, pp 345–360
- Qureshi K, Shah SMH, Manuel P (2011) Empirical performance evaluation of schedulers for cluster of workstations. *Cluster computing* 14(2):101–113
- Riaz R, Kazmi SH, Kazmi ZH, Shah SA (2018) Randomized dynamic quantum cpu scheduling algorithm. *Journal of Information Communication Technologies and Robotic Applications* pp 19–27

- Ritter D, May N, Rinderle-Ma S (2017) Patterns for emerging application integration scenarios: A survey. *Information Systems* 67:36–57
- Ritter D, Rinderle-Ma S, Montali M, Rivkin A (2019) Formal foundations for responsible application integration. *Information Systems* p 101439
- Saifullah A, Li J, Agrawal K, Lu C, Gill C (2013) Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems* 49(4):404–435
- Sargent RG (2013) Verification and validation of simulation models. *Journal of simulation* 7(1):12–24
- Schwarzkopf M, Konwinski A, Abd-El-Malek M, Wilkes J (2013) Omega: Flexible, scalable schedulers for large compute clusters. *Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys 2013* pp 351–364
- Shoukry A, Khader J, Gani S (2019) Improving business process and functionality using IoT based E3-value business model. *Electronic Markets* 1:1–10
- Thakur V, Kumar S (2018) A pragmatic study and analysis of load balancing techniques in parallel computing. In: *Information and Decision Sciences*, pp 447–45400
- Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) *Experimentation in software engineering*. Springer Science & Business Media
- Wood DC, Forman EH (1971) Throughput measurement using a synthetic job stream. In: *Fall Joint Computer Conference*, pp 51–56
- Zhang T, Pota H, Chu CC, Gadh R (2018a) Real-time renewable energy incentive system for electric vehicles using prioritization and cryptocurrency. *Applied Energy* 226:582–594
- Zhang Y, Shen ZJM, Song S (2018b) Exact algorithms for distributionally  $\beta$ -robust machine scheduling with uncertain processing times. *INFORMS Journal on Computing* 30(4):662–676

# Appendices

## A Profiles of the integration processes

### A.1 Profiles of integration processes

#### A.1.1 Processing Order (IP1)

- Identification of the integration process tasks:  
 $\text{VectorIdTask} = \{1,2,3,4,5,6,4,8,9,10,11,12,13,14,15,16,17\};$
- Identification of the next task of each task of the integration process:  
 $\text{VetorNextTask} = \{\{2\}, \{3\}, \{4,12,17\}, \{5,7\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \{11\}, \{\}, \{13,15\}, \{14\}, \{15\}, \{16\}, \{9\}, \{\}\};$
- Identification of the execution time range of task of each task of the integration process:  
 $\text{VetorTimeExec} = \{\{1,2\}, \{2,3\}, \{2,3\}, \{2,3\}, \{1,2\}, \{1,2\}, \{3,4\}, \{1,2\}, \{3,4\}, \{1,2\}, \{1,2\}, \{2,3\}, \{1,2\}, \{1,2\}, \{3,4\}, \{1,2\}, \{1,2\}\};$

- Identification of the logic operation type of task of each task of the integration process:  
 $\text{VetorOper} = \{\{\}, \{\}, \{\text{or}\}, \{\text{and}\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\text{and}\}, \{\}, \{\}, \{\}, \{\}, \{\}\};$
- Identification of the parallel tasks of the integration process:  
 $\text{VetorParallelTask} = \{0,0,0,0,0,0,0,0,0,0,0,4,5,6,7,8,4\};$
- Identification of the last tasks of the integration process:  
 $\text{LastTask} = \{11,17\}.$

#### A.1.2 Huelva's County Council (IP2)

- Identification of the integration process tasks:  
 $\text{VectorIdTask} = \{1,2,3,4,5,6,4,8,9,10,11,12,13,14,15,16,17,18,19\};$
- Identification of the next task of each task of the integration process:  
 $\text{VetorNextTask} = \{\{3\}, \{3\}, \{4\}, \{5,7\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10,18\}, \{11\}, \{12,14\}, \{13\}, \{14\}, \{15\}, \{16\}, \{17\}, \{\}, \{19\}, \{\}\};$
- Identification of the execution time range of task of each task of the integration process:  
 $\text{VetorTimeExec} = \{\{1,2\}, \{1,2\}, \{3,4\}, \{2,3\}, \{1,2\}, \{1,2\}, \{3,4\}, \{1,2\}, \{2,3\}, \{1,2\}, \{2,3\}, \{1,2\}, \{1,2\}, \{3,4\}, \{1,2\}, \{1,2\}, \{1,2\}, \{1,2\}, \{1,2\}\};$
- Identification of the logic operation type of task of each task of the integration process:  
 $\text{VetorOper} = \{\{\}, \{\}, \{\}, \{\text{and}\}, \{\}, \{\}, \{\}, \{\}, \{\text{or}\}, \{\}, \{\text{and}\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}, \{\}\};$
- Identification of the parallel tasks of the integration process:  
 $\text{VetorParallelTask} = \{0,1,0,0,0,0,5,0,0,0,0,0,0,12,0,0,0,16,17\};$
- Identification of the last tasks of the integration process:  
 $\text{LastTask} = \{17,19\}.$

#### A.1.3 Real Estate (IP3)

- Identification of the integration process tasks:  
 $\text{VectorIdTask} = \{1,2,3,4,5,6,4,8,9,10,11,12,13,14,15,16,17\};$
- Identification of the next task of each task of the integration process:  
 $\text{VetorNextTask} = \{\{5\}, \{5\}, \{4\}, \{5\}, \{6\}, \{7,8\}, \{8\}, \{9\}, \{10,11\}, \{11\}, \{12\}, \{13,16\}, \{14\}, \{15\}, \{\}, \{17\}, \{\}\};$
- Identification of the execution time range of task of each task of the integration process:  
 $\text{VetorTimeExec} = \{\{1,2\}, \{1,2\}, \{1,2\}, \{1,2\}, \{3,4\}, \{2,3\}, \{1,2\}, \{3,4\}, \{2,3\}, \{1,2\}, \{1,2\}, \{1,2\}, \{1,2\}, \{3,4\}, \{2,3\}, \{1,2\}, \{3,4\}, \{2,3\}, \{1,2\}, \{1,2\}\};$

---

$\{1,2\}, \{3,4\}, \{2,3\}, \{1,2\}, \{1,2\}, \{1,2\}, \{1,2\}, \{1,2\}$ ;

- Identification of the logic operation type of task of each task of the integration process:

VetorOper =  $\{\{\}, \{\}, \{\}, \{\}, \{\}, \{\text{and}\}, \{\}, \{\}, \{\text{and}\}, \{\}, \{\}, \{\text{or}\}, \{\}, \{\}, \{\}, \{\}, \{\}\}$ ;

- Identification of the parallel tasks of the integration process:

VetorParallelTask =  $\{0,1,1,0,0,0,0,0,0,0,0,0,0,0,13,14\}$ ;

- Identification of the last tasks of the integration process:

LastTask =  $\{15,17\}$ .



A.2 Flowchart of the simulator

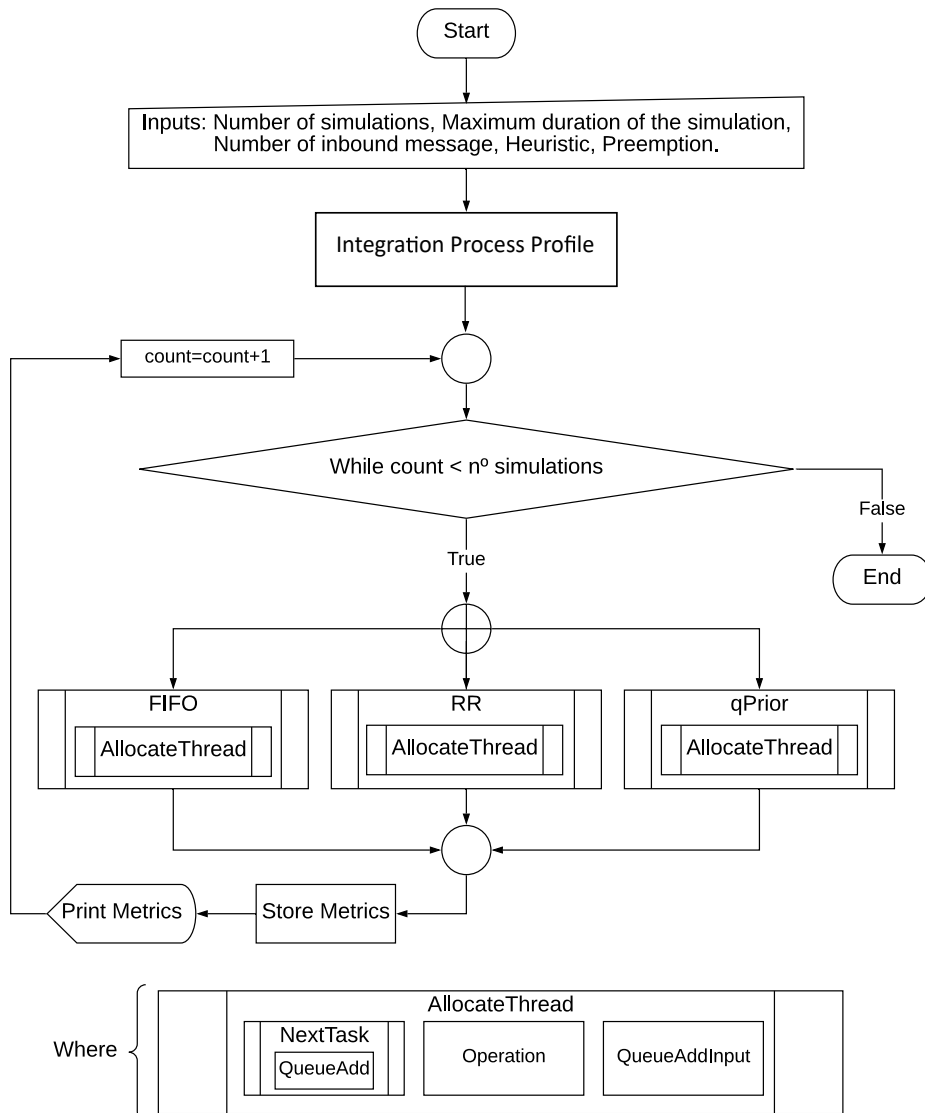


Fig. 6 Flowchart of the IPS.