Queue-priority optimized algorithm: a novel task scheduling for runtime systems of application integration platforms

Daniela L. Freire, UNIJUI University, Rua do Comércio, 3000, Ijuí, 98700-000, Brazil

Rafael Z. Frantz, UNIJUI University, Rua do Comércio, 3000, Ijuí, 98700-000, Brazil

Fabricia Roos-Frantz, UNIJUI University, Rua do Comércio, 3000, Ijuí, 98700-000, Brazil

Vitor Basto-Fernandes, University Institute of Lisbon (ISTAR-IUL), Av. das Forças Armadas, 376, 1649-026, Lisbon, Portugal

Abstract
The need for integration of applications and services in business processes from enterprises has increased with the advancement of cloud and mobile applications. Enterprises started dealing with high volumes of data from the cloud and from mobile applications, besides their own. This is the reason why integration tools must adapt themselves to handle with high volumes of data, and to exploit the scalability of cloud computational resources without increasing enterprise operations costs. Integration platforms are tools that integrate enterprises' applications through integration processes, which are nothing but workflows composed of a set of atomic tasks connected through communication channels. Many integration platforms schedule tasks to be executed by computational resources through the First- in-first-out heuristic. This article proposes a Queue-priority algorithm that uses a novel heuristic and tackles high volumes of data in the task scheduling of integration processes. This heuristic is optimized by the Particle Swarm Optimization computational method. The results of our experiments were confirmed by statistical tests, and validated the proposal as a feasible alternative to improve integration platforms in the execution of integration processes under a high volume of data.

Keywords: Task scheduling algorithm; System integration; Application integration; Optimization; Heuristic

1 Introduction
Applications that compose the enterprise software ecosystems [38] must work together, in order to aggregate value for business processes. Enterprise Application Integration (EAI) is the foundation for software ecosystems which are characterized by heterogeneity and distribution [45]. The relevance of EAI increased due to the fast-emerging of cloud and mobile applications, which are now part of companies' daily business affairs [35], and especially by the advent of the pay-as-you-go charging model adopted by cloud providers to offer software as services, and where the payment is proportional to the consumption of computational resources [12]. Enterprise Application Integration is the research field that provides methodologies, techniques, and tools to develop integration processes. It enables applications to share data and functionality, to meet the requirements of business processes [21]. Integration

tools need to be re-engineered to better explore the model introduced by cloud computing, so that they can take advantage of the scalability and reduce costs by optimising computational resources usage [36].
The conceptual model of an integration process is a workflow formed by distinct atomic tasks connected by communication channels. A task performs a specific operation such as transforming, filtering, splitting, joining, routing or any modification can be applied a one or more messages. Integration platforms are EAI tools that allow software engineers to design, implement, run, and monitor integration processes. A set of conceptual integration patterns, which initially was documented by [29] and lately extended by [44], have become a reference cookbook for the integration community. Such patterns have also inspired the development of opensource integration platforms such as ServiceMix [32], Petals [53], Jitterbit [48], and Guaraná [21]. These open-source integration platforms follow the Pipes-and-Filters architecture [2], which allows the uncoupling of tasks from integration processes. Pipes represent communication channels, while filters represent atomic tasks in an integration process; they both implement a concrete integration pattern to process encapsulated data in messages. Tasks are executed by computational resources known as threads. A thread is the smallest unit of a computational program that can be managed by integration platforms.
Such platforms usually provide a domain-specific language, a development toolkit, monitoring tools, and a runtime system. The domain-specific language enables the description of conceptual models for integration processes. The development toolkit is a set of software tools that allow the implementation of integration processes, i.e., it transforms a conceptual model into executable code. Monitoring tools are used to detect failures that may occur during the execution of an integration process. The runtime system is responsible for running all integration processes, and its primary function is the task scheduling [25, 27]. Task scheduling schedules the execution of tasks and the allocation of computational resources and it concerns with the time of task execution and with the computational resources that perform the task.
Messages coming from the cloud and mobile application common in the contemporaneous environments combined with messages coming from traditional enterprise environments have to be integrated and processed while guaranteeing high throughput [43]. 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 [33, 59]. Enterprises also face the challenge to suit and integrate their applications, together with the optimization of resource usage to save costs [23, 25, 26].
Our survey [22] identified research directions regarding fair execution of tasks of integration processes by optimization of the task scheduling and the resource allocation of the runtime systems of the integration platforms.
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 that due to the pay-asyou-go charging model [22]. In the context of cloud, task scheduling concerns with

costs reduction [22], with the handling of high workloads [50], and with software quality in terms of flexibility and response time [17]. Besides, it is necessary to optimize the use of computational resources because the processing of a large amount of data requires more allocation of resources [17, 50]. In overload situations due to thigh and continuous input rate, the scheduling of tasks of the workflow tends to concentrate on tasks in the begging of the workflow. We consider that an integration process is in overload situation when the number of remained message is greater than the number of processed message. This concentrating of initial tasks causes the threads to execute them more frequently, to the detriment of the other tasks. This behavior impacts negatively on the performance of the integration process.
Most open-source integration platforms adopt the First-in-first-out (FIFO) heuristic, especially when scaling requirements are not known. This is because FIFO is the simplest heuristic and achieves consistent results in most application integration scenarios [22]. However, the performance of this heuristic depends on many factors and is rarely optimal for all situations [42], like in the case of overload. In those situations, the scheduling by FIFO tends to concentrate on the workflow initial tasks, and it ends up degrading execution performance of integration processes. It becomes necessary a fair task scheduling, which provides optimal resources allocation to maximise performance [8, 40]. Such allocation of optimal resources requires an active and dynamic task scheduling heuristic [51], which is capable of increasing the number of processed messages per time unit.
There are many studies and proposals of algorithms regarding task scheduling, but none of them deals singularly with integration processes scheduling. The integration processes scheduling has quite specific characteristics, such as unknown message arrival rate, variable task processing time, unpredictable path of the workflow traced by messages, elastic resource provisioning. The integration processes scheduling has quite specific characteristics, such as unknown message arrival rate, variable task processing time, unpredictable path of the workflow traced by messages, elastic resource 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 re-engineering of the integration platforms, it is not possible to ensure to suit that they can deal with contemporaneous environments. Besides, it is not possible to ensure that the enterprises take advantage of the scalability provided by cloud computing or optimize computational resource usage.
In this article, we propose a queue-priority optimized (qPrior) algorithm, which addresses a new task scheduling heuristic for the execution of integration processes under high workloads. In this heuristic, there are multiple prioritized queues to store task instances; these tasks wait for available threads to execute them. Each queue stores task instances from an integration process. Threads check the queue according to this priority and execute a predefined number of tasks at each checking. This number of tasks is found by the Particle Swarm Optimization (PSO) meta-heuristic in order to maximise the number of processed messages per time unit. To validate our proposal, we performed two experiments. First, we compared the task scheduling performance with and without the PSO optimization. Then, we simulated the execution of a classic integration process and compared performances between qPrior-optimized and FIFO heuristic. We evaluated the executions of integration

processes by the FIFO heuristic. We verified that with this heuristic, as the input rate increases, the number of messages processed decreases until no message can be processed. These results show that the qPrior algorithm handles better with the overload in the processing of messages than FIFO because qPrior responds more rapidly, producing outbound messages in a shorter time than FIFO, besides keeping the production of outbound messages. We validated our proposal by performing two experiments, and their results confirmed statistically with ANOVA and Scott & Knoot statistical tests.
The rest of this article is organized as follows: Section 2 discusses works related to optimization of task scheduling using PSO; Sect. 3 provides background information over integration processes and their task scheduling and formulates the problem; Sect. 4 exposes the proposed qPrior-optimized algorithm; Sect. 5 reports experiments and statistics to validate the proposal; and, Sect. 6 presents our conclusions.

2 Related work

This section gathers works in the subject of task scheduling and allocation of computational resources from different research fields in order to increase the systems performance. The works related to performance optimization by PSO meta-heuristic or related to proposals of task scheduling algorithms are summarized in Table 1.
This article differs from others, once it proposes an algorithm to maximise throughput from the execution of integration processes under high workloads, through a new heuristic for task scheduling, carried out by run-time systems of integration platforms.
Rodriguez and Buyya [46] proposed a PSO-based algorithm to minimise the overall execution cost and to meet deadline constraints for scheduling a scientific workflow application in a cloud environment. Their goal was to deal with the dynamic provisioning and heterogeneity of unlimited computational resources and the performance variation of virtual machines. Jian et al. [31] proposed a PSO-based algorithm to schedule tasks to cloud resource suppliers, taking into account how reliable such resource providers were and the network data transmission among suppliers. A mathematical model defined a reliability measure, which was used to evaluate the task to be run, and also the reliability degree in data transmission. Verma and Kaushal [55] proposed a task scheduling over available cloud resources by minimising the execution cost, considering at the same time the constraints of deadline and budget. Aron et al. [6] proposed a PSO-based hyper-heuristic method that minimises time, cost, and the optimized utilization from resources in the grid environment, without violating any security norms. Ghosh and Das [24] proposed a hybrid algorithm which combined Extreme Optimization and PSO. Their algorithm aimed to simultaneously minimise makespan, processing cost and job failure rate, while maximising the resource utilization of computational Grid systems. Blythe et al. [9] propose a discrete version of the PSO algorithm, applied on a single-objective and on a multi-objective optimization scenario. Rodriguez and Buyya [47] proposed the scheduling for resource provisioning and a scheduling strategy by a dynamic heuristic-based algorithm. The goal was to minimize the overall cost of leasing infrastructure resources without compromising workflows deadlines. Anwar and Deng [5] proposed an algorithm based on the Bag of Tasks technique, which groups the workflow into bags of tasks by the criterion of data dependency and priority constraints.

The algorithm also optimizes the allocation of elastic, heterogeneous and dynamically provisioned cloud resources. Sun et al. [52] proposed a flexible online scheduling framework for big data streaming applications (E-Stream). The authors were based on the priority-based "earliest finish time first" and scheduled multiple graphs by a max-min fairness strategy.
Wang et al. [56] proposed a pattern classification algorithm, based on a hybrid PSO. The approach was to train the scheduler and switch scheduling strategies in real-time. Manasrah and Ali [37] proposed a hybrid algorithm based on PSO and GA meta-heuristics. The authors concerned efficiency of the allocation of tasks to the resources workflow applications in cloud computing environments. Ghafouri et al. [23] proposed an algorithm based on back-tracking heuristic combined with the scheduling of critical and non-critical tasks. The goal was to minimise makespan and the execution cost from workflow applications. Xie et al. [59] proposed an algorithm based on PSO that employs non-linear inertia weight with selection and mutation operations by a directional search process. The goal was the efficient allocation of tasks to the different resources for meeting the real-time requirement of the cloud and edge computing environments. Pietri et al. [40] proposed an algorithm that seeks to identify Pareto-optimal trade-offs between overall execution time, monetary cost and fairness, efficiently exploring the solution space. The authors approached the scheduling multiple dataflows on heterogeneous clouds.

3 Problem formulation

In this section, we address task scheduling for integration processes and the main components and the execution model approached in this article. After, we describe the environment of an integration processes, based on the research work by [19] and a mathematical modelling for our problem. We first explain the software system, then we represent the integration process by a Directed Acyclic Graph (DAG). After that, we formulate the mathematical problem and our fitness function.

3.1 Task scheduling for integration processes

A conceptual model of an integration process is a workflow composed of tasks, which communicates through channels. Integration processes receive data from source applications and deliver them to destination applications. Data are wrapped in messages that have a header, containing custom properties, and a body providing payload data. A message flows through an integration process and it is modified by tasks processing. Transforming, filtering, splitting, joining, or routing any modification can apply, depending on the specific operation that each task perform. Therefore, a message can be split into one or more messages in the workflow, or even two or more messages can be merged into a unique message. We identify a «path» as a specific set of tasks and communication channels where a message is completely processed in an integration process. A dependence order determines the sequence in which tasks must be executed. So, a task processes a message, whenever this message has already been processed by each of its predecessor tasks. An outbound message of a task is written to the communication channel that connects this task with

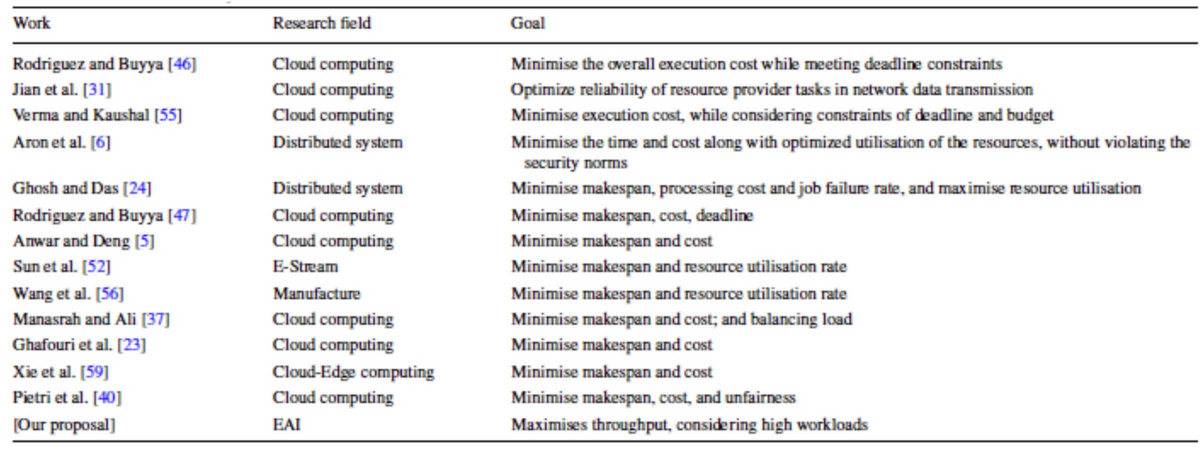
the next one in the workflow path. The «scheduler» is the main element of runtime systems during the execution of integration processes. The scheduler manages activities from the message processing, and also the use of computational resources for the execution of tasks. Computational resources that execute tasks are called «threads», which are usually grouped in «thread pools».
The runtime system execution model defines procedures for task execution and for thread allocation, during the processing of messages in integration processes. Two execution models are found in the literature, namely: process-based and taskbased [3, 9, 10, 20]. In the process-based model, a thread executes every task of the workflow over an inbound message, so that this message can flow throughout the process. After every task in the workflow has been executed, the thread is released. In the task-based model, a 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; this channel connects the current task to the next one 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 the referred task. We approached the task-based model in this article.
In the task-based model, a ready task is the one that is on the verge of being executed, once there are messages in all communication channels that are inputs to this very task. The solicitation of the execution of a ready task is recorded in a waiting queue. The waiting queue stores ready tasks that wait to be executed by the available threads. An available thread selects a task from the waiting queue, following a Firstin-first-out policy, in which the task that was recorded first is scheduled first to be executed. One of the roles of a scheduler is to create, manage, and release threads. Additionally, it is possible to configure the pool with some parameters, such as: the initial thread 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 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 among them.

3.2 Environment of an integration process

The Directed Acyclic Graph (DAG) represents task models for real-time scheduling, allowing the description of constraints on tasks execution [49]. 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 (Ei, Ti) , where
Ti = ti,1, ti,2,..., ti,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 (ti, tj) represents a dependence between ti and tj , in which ti is the predecessor node of tj and tj is the successor node of ti . 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.
The Coffee shop integration process is our case study (CS). CS is a benchmark of an integration process introduced by [28]. Its conceptual model is depicted in Fig. 1. In the CS, 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.

Table 1 Related works using optimization by the PSO



Work	Research field	Goal
Rodriguez and Buyya [46]	Cloud computing	Minimise the overall execution cost while meeting deadline constraints
Jian et al. [31]	Cloud computing	Optimize reliability of resource provider tasks in network data transmission
Verma and Kaushal [55]	Cloud computing	Minimise execution cost, while considering constraints of deadline and budget
Aron et al. [6]	Distributed system	Minimise the time and cost along with optimized utilisation of the resources, without violating the security norms
Ghosh and Das [24]	Distributed system	Minimise makespan, processing cost and job failure rate, and maximise resource utilisation
Rodriguez and Buyya [47]	Cloud computing	Minimise makespan, cost, deadline
Anwar and Deng [5]	Cloud computing	Minimise makespan and cost
Sun et al. [52]	E-Stream	Minimise makespan and resource utilisation rate
Wang et al. [56]	Manufacture	Minimise makespan and resource utilisation rate
Manasrah and Ali [37]	Cloud computing	Minimise makespan and cost; and balancing load
Ghafouri et al. [23]	Cloud computing	Minimise makespan and cost
Xie et al. [59]	Cloud-Edge computing	Minimise makespan and cost
Pietri et al. [40]	Cloud computing	Minimise makespan, cost, and unfairness
[Our proposal]	EAI	Maximises throughput, considering high workloads

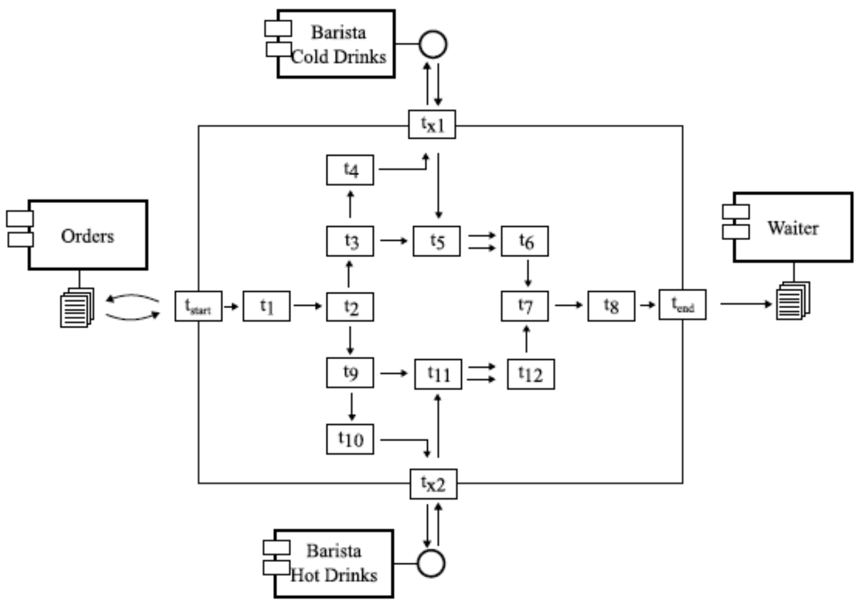


Fig. 1 Conceptual model of Coffee case study

The CS 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 tstart , and one output task represented by tend . Tasks that exchange messages with applications during runtime are represented by tx1 and tx2 . Intermediary tasks are represented by ti , 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 CS 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 [t3, t9] , [t4, t10] , [tx1, tx2] , [t5, t11] , [t6, t12].
In CS integration operation typed graph, there are 16 nodes, which represent the tasks, and there are 19 edges, which represent the channels. Node tstart is a starting node, that represents a task of the type «start». The nodes t3 and t9 represent tasks of the type «and». The t2 represents a task of the type «or». The nodes t5 , t7 , and t11 represent tasks of the type join; Nodes t1 , t4 , t6 , t8 , t10 , and t12 represent tasks of the type « message processor »; The nodes tx1 and tx2 represent tasks that send and receive information to/from applications that are tasks of the type external call ; and, node tend is a ending node that represent a task of the type «end».

3.3 Mathematical formulation

In this section, we present the mathematical modelling of integration processes, with focus on time-related performance metrics, commonly used for scheduling performance benchmarking. Makespan is a well-known metric for performance within

the integration community. Several researchers use the makespan arithmetic average as a performance metric for scheduling algorithms [1, 13, 34].
In integration processes, we defined the total processing time of a message « TPmi » as the elapsed time interval between the time a message entered and the time it leaves the integration process. « TPmi » is the sum of the execution time of all the tasks of the path by which the message must flow for its complete processing. We assume that the execution time of a task, « ETtk » 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 « tk » is defined as « [ETtk ini , ETtkfin]».
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 « STm1 » and the end time of the last message that leaves the workflow « ETmnp ». 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 [54, 58]. 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. Eq. 1. This formulation is represented by the objective function: max{Throughput}.

$$\text{Throughput} = \frac{\text{np}}{\text{ET}_{m_{\text{np}}} - \text{ST}_{m_1}} \qquad (1)$$

Time optimization becomes fundamental in situations in which the process execution duration must meet specific constraints or deadlines, because their violations increase the business processes costs [39]. 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.

4 Our algorithm

In this section, we propose our heuristic to tackle task scheduling of integration processes. We named this heuristic as Queue Priority (qPrior). It deals with overload situations caused by workloads of large volumes of data (of the order of millions).
We implement qPrior by a lightweight algorithm to increase the performance of the execution of the integration processes. The source code developed and used in this experiment is publicly available for download (http:// github. com/ gca- resea rch- group/ simul ation- queue priory- pso).

4.1 qPrior 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 the number of tasks that are executed as preemption. Tasks are allocated to threads till the tasks are entirely executed. In other words, the preemption will be the number of threads allocated by one time to every execution. So the preemption optimal has to be determined previously.
We used the PSO meta-heuristic for this purpose. PSO is widely in scheduling algorithms [55] and presents better computational performance for high dimensional nonlinear optimization problems with continuous variables and has fewer parameters to adjust than other algorithms, facilitating their implementation [4,11].
To present the qPrior heuristic, we present four possible iterations of the heuristic illustrated in Fig. 2. 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 t3 . 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 t2 . The queue of low priority maintains the annotations for executions of the instances of task t1 . 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 t3.

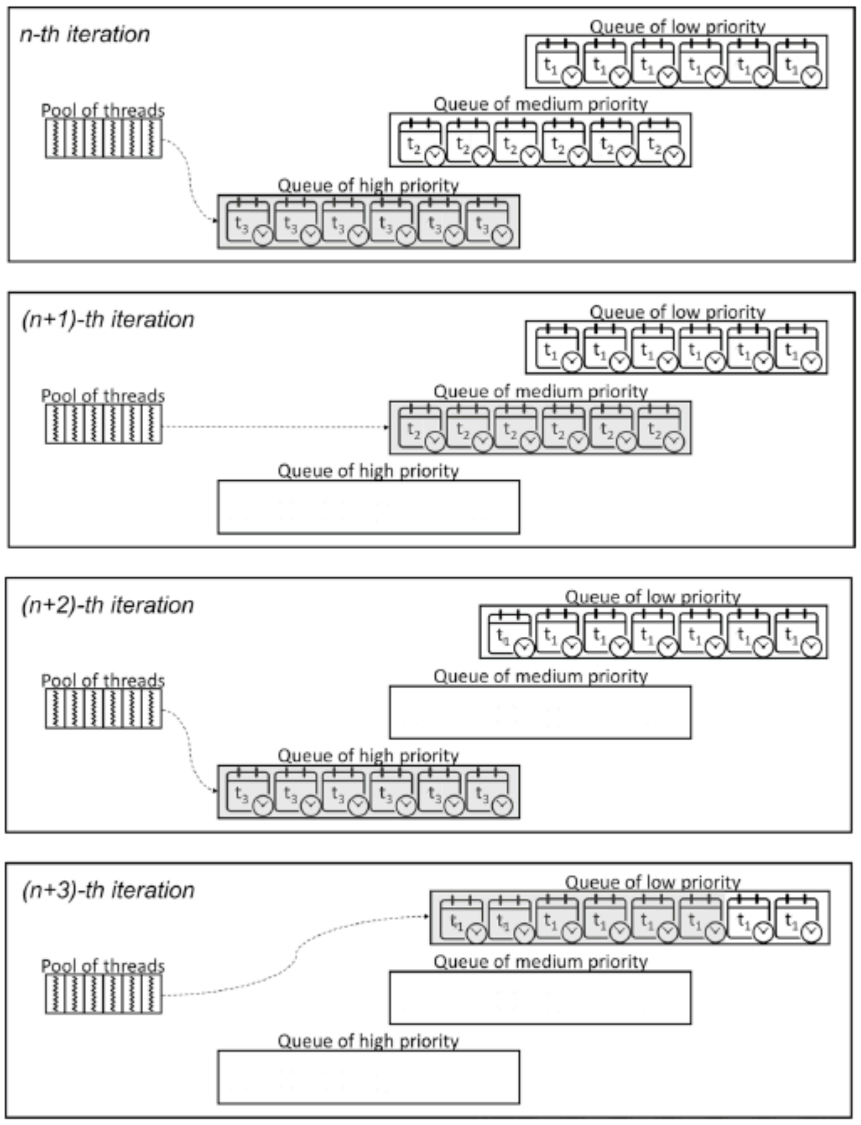


Fig. 2 Iterations of the qPrior heuristic

[(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 t2 of this queue. After, the executions of tasks t2 produce outbound messages that are inbound messages for successor task, t3 . 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 t1.

Table 2 PSO parameters

Parameter	Value
ω	1
ϕ_p	1
ϕ_g	1
Particle number	20
Particle dimension	2
Range	[25,000, 50,000], [50,000, 75,000]
Number of iteration	100

[(n+2)-th iteration:] threads poll the queue of high priority and, then, they execute six tasks of queue of tasks t3 . While new tasks continue being annotated in the queue that maintains the tasks t1.
[(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 t1 of this queue. After, the executions of tasks t1 produce outbound messages that are inbound messages for successor task, t2 . So, the scheduler annotates the executions in the queue of tasks t2.
This process iteratively continues till up all queues are empty or till up it is interrupted.

4.2 qPrior algorithm

We have implemented qPrior algorithm that the software engineer to perform the Queue Priority heuristic. The pseudo-code of the algorithm that implements the qPrior heuristic is shown in Algorithm 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 execute 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. Our proposal is computationally tractable since handling tasks runs in a low time. This low complexity of the algorithm provides good performance, more straightforward execution, debug and maintenance.

Algorithm 1 *qPrior*

```
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): $preemptTask$

1:  $totalSize \leftarrow 1$
2:  $preeemp \leftarrow preemptTask$
3:  $qPrior \leftarrow numTasks$
4:  while $totalSize > 0$ and $duration < maxDuration)$ do
5:      for $i = numTasks$ to 1 with step$-1$ do
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:     $Allocate\ Thread(queues[qPrior], preempt)$
17: end while
18: $totalSize \leftarrow 0$
19: for $i = 1$ to $numTasks$ do
20:     $totalSize \leftarrow totalSize + queues[i].size$
21: end for
22: $duration \leftarrow$ current.Time $-start$
```

4.3 PSO modeling

PSO is a meta-heuristic proposed by Eberhart and Kennedy [16] based on the behavior of animal flocks, such as insects, fish, and birds. Such animals find the best regions for nourishing through the iterative adjustment of their positions in the search space, taking into account their best individual positions and the group's best general position. The algorithm responsible for implementing PSO is widely researched and utilized. It presents better computational performance in highdimensional nonlinear optimization problems with continuous variables, and it has fewer parameters to adjust than other algorithms; such facts turn its implementation easier [4, 11].

When there is a high arrival rate of messages, the queues accumulate more annotations of the initial tasks and threads keep busy in the 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 engineer 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 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 2 summarises the PSO parameters for a total workload of 2,000,000 messages.

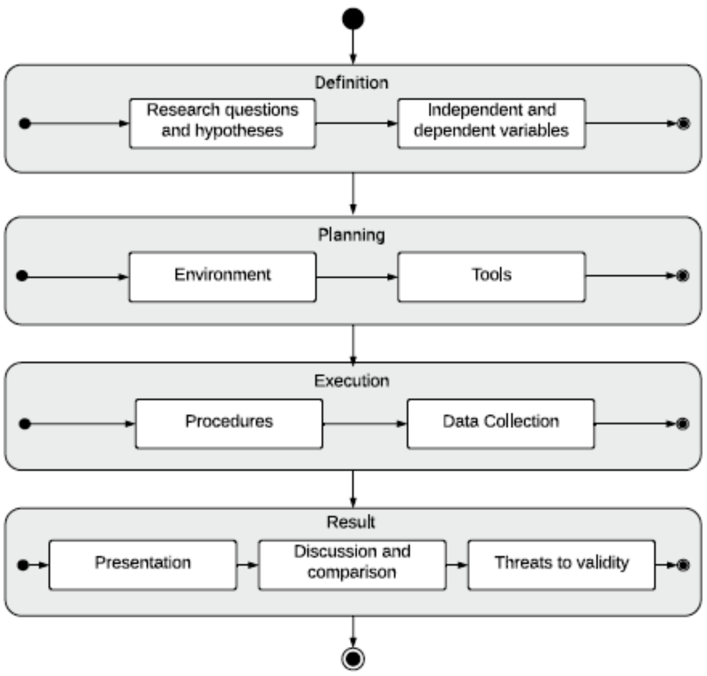


Fig. 3 Experimental protocol

5 Performance evaluation

In this section, we describe the evaluation environment, the supporting tools, and
the general information about execution and data collection related to two qPrioroptimized
performance experiments in Sects. 5.3 and 5.4. The former evaluates
PSO optimization and the importance of choice of the proper preemption. The
latter compares qPrior-optimized with FIFO heuristic.
The experiments followed a protocol based on [7, 30, 57], 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
Fig. 3 and, in following, detail them.
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.

5.1 Threats to validity

In this section, 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 [15, 57]. Four types of threats to validity are discussed here, they are: constructor validity, conclusion validity, internal validity, and external validity.

Constructor validity discusses whether the planning and execution of this research are well adequate to answer the research question. We planned the experiments according to procedures from empirical software engineering [7, 30, 57].

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.

As reported by [57], 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 aims to ensure that the treatment caused the outcome, mitigating effects of other uncertain factors or not measured [18]. 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 [41].

Then, first execution was 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 focuses on the generalisation of the results outside the scope of our article [18]. This research is generalized for integration platforms that adopt the integration patterns by [29], the style Pipes-and-Filters, and task-based execution model. We reported this article following a practical guideline [57], 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.

5.3 Evaluation of the Preemption

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.

5.3.1 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.

5.3.2 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 Coffee case study.
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 1000 msg.
Rate of inbound messages. The number of inbound messages added periodically to the integration process. The value tested for this variable was 1000 msg.
Preemption. Number of tasks executed at each time queue checking. The values tested for this variable were 100, 500, 1000, 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.

5.3.3 Execution and data collection

The experiment was conducted using an java simulator, which simulates the execution of the case studies aforementioned. The simulation starts with a workload of 1000 msg and receives 1000 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 sizes for preemption. For each one of them, we repeated 25 times the execution of the qPrior heuristic, resulting in 300 scenarios, summarized in Table 3.

We tested the PSO algorithm with the range for search space [50,000; 55,000].
For our case study, the best throughput was 30,528 msg/s, found the near-optimal preemption equals to 50,000 tasks, and the execution time of PSO algorithm took 7203 s. The output of the PSO algorithm is shown in Table 4.

5.3.4 Results

The average throughput obtained in the 25 repetitions of the simulation for every value of the preemption is shown in Fig. 4, where the x-axis represents preemption, and the y-axis represents the average throughput in messages per seconds (msg/s).

Table 3 Scenarios for preemption evaluation

Integration process	Coffee case study (CS)	1
Total workload	2,000,000 msg	1
Rate of inbound messages	1000 msg	1
Initial workload	1000 msg	1
Preemption	100, 500, 1000, 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 × 1 × 1 × 1 × 12 × 25	300

Table 4 Output of PSO algorithm

Test function	:	qPriorPSO
Number of particles	:	20
Iterations	:	100
Global best position	:	[50,000; 55,000]
Global best value	:	0.000032755755697
Preemption	:	55,000
Throughput (msg/s)	:	30,528
Duration qPrior(s)	:	60
f(25,000, 75,000)	:	0.000032755755697

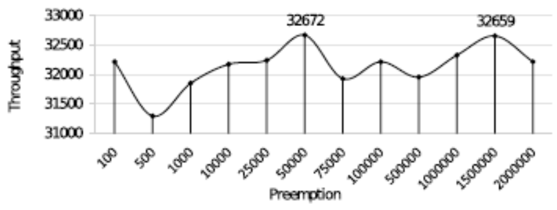
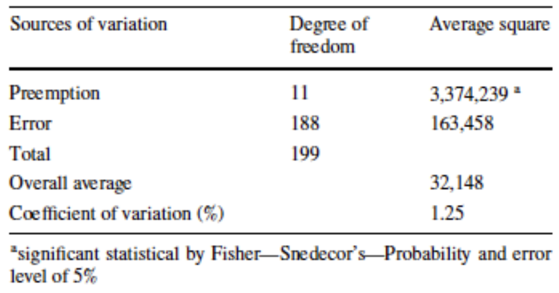


Fig. 4 Average throughput regarding preemption (with 95% confidence interval)

In this figure, we outline the values in 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 Coffee case study under a workload of 2,000,000 msg is shown in Table 5, with 95% confidence interval. 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 6. First column represents 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 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 1000, 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.

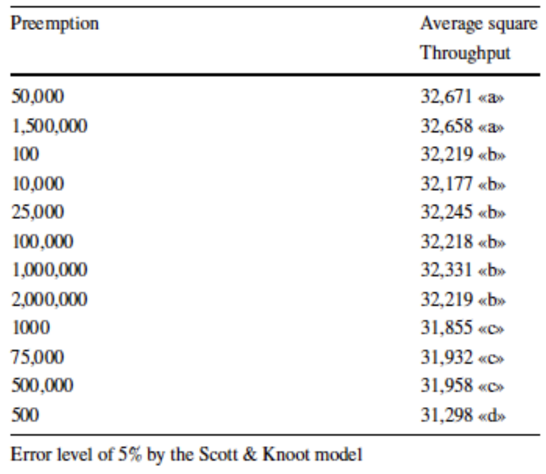
Table 5 Variance analysis of the throughput for preemption evaluation



Sources of variation	Degree of freedom	Average square
Preemption	11	3,374,239 [a]
Error	188	163,458
Total	199	
Overall average		32,148
Coefficient of variation (%)		1.25

[a]significant statistical by Fisher—Snedecor's—Probability and error level of 5%

Table 6 Scott and Knott test for preemption



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»
1000	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

5.3.5 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/s, 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 (60 s). 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 5, 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 6. 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.

5.4 Evaluation of qPrior
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.

5.4.1 Research question and hypothesis
This experiment aims to answer the following research question:
RQ: What is the performance behaviour 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.

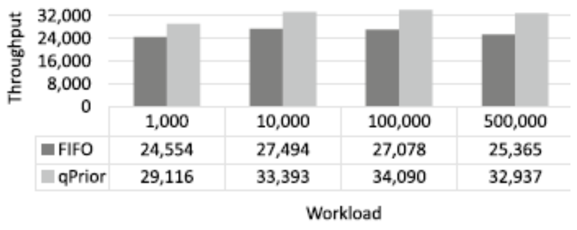
5.4.2 Variables
The independent variables controlled in the execution of the algorithm are:
Heuristic. The heuristic used to task scheduling. The heuristics tested were: FIFO and qPrior.
Integration process. The conceptual model of the integration process. The model tested was the Coffee case study.
Workload. The number of inbound messages. The value tested for this variable were 1000, 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 1000 msg.
Rate of inbound messages. The number of inbound messages added periodically to the integration process. The value tested for this variable was 1000 msg.
Preemption. The number of task executed at each time queue checking. The values tested for this variable was 50,000 tasks.
The dependent variable measured in the execution of the algorithm is:
Throughput. The number of processed messages by time unit.

Table 7 Scenarios for comparison of heuristics regarding workloads

Heuristics	FIFO and qPrior	2
Integration process	Coffee case study (CS)	1
Total workloads	1000, 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	1000 msg	1
Initial workload	1000 msg	1
Preemption	50,000 tasks	1
Repetitions	1...25	25
Scenarios	$2\times 1\times 8\times 1\times 1\times 1\times 25$	400

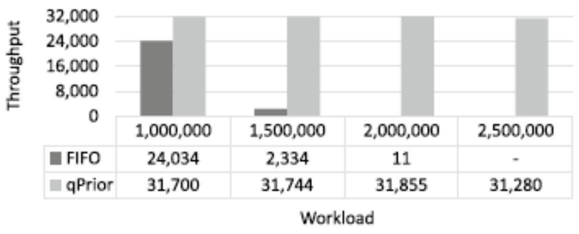
5.4.3 Execution and data collection

The experiment was conducted using a java simulator, which simulates the execution of the case studies aforementioned. The simulation starts with a workload of 1000 msg and receives 1000 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 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 60 s. Thus, the simulator stops the generation of messages when it reaches this maximal number of messages and stops the running after 60 s. Then, the simulator stores the workload and the throughput in a text file. After, we handled and statistically analysed these metrics. We tested the execution using 8 different workloads and 1 case study. For each one of them, we repeated 25 times the execution using qPrior and 25 times using FIFO, resulting in 400 scenarios, summarized in Table 7.



	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

Fig. 5 Average throughput—low workload (with 95% confidence interval)



	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

Fig. 6 Average throughput—high workload (with 95% confidence interval)

5.4.4 Results

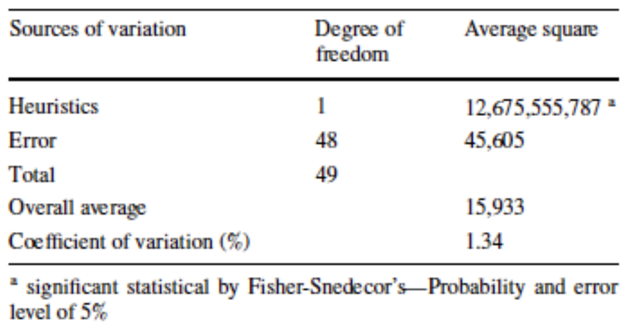
The average values of dependent variables obtained in the 25 repetitions of the simulation for every value of the workload are shown in scatter charts. The x-axis represents the workload for every heuristic. The y-axis represents the average throughput

measured. We consider 1000, 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. Figure 5 represents the average throughput with low and
Fig. 5 high workloads.

In the simulation with a workload of 1000 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.

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 (Fig. 6).

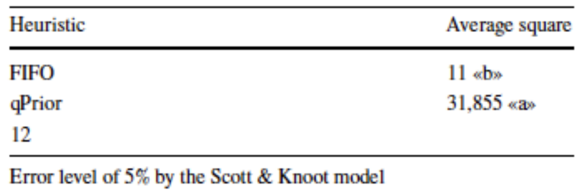
Table 8 ANOVA test



Sources of variation	Degree of freedom	Average square
Heuristics	1	12,675,555,787 [a]
Error	48	45,605
Total	49	
Overall average		15,933
Coefficient of variation (%)		1.34

[a] significant statistical by Fisher-Snedecor's—Probability and error level of 5%

Table 9 Scott & Knott test



Heuristic	Average square
FIFO	11 «b»
qPrior	31,855 «a»
12	

Error level of 5% by the Scott & Knoot model

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 variable, for both heuristics, were different of zero. The workload of 2,000,000 msg for CS and the analysis of variance is shown in Table 8, with 95% confidence interval. Since there were statistical differences between the results with different scenarios, we follow the comparison of averages of the dependent variable by Scott & Knott test. The results of the Scott & Knoot test is shown in Table 9. In this table, the heuristics are in the first column.

For the dependent variable, there is a column for the average and a column for the group of Scott & Knott. Two groups were found: «a» and «b». Group «a» refers to the heuristic with the highest average of throughput and group «b» refers to the heuristic with the lowest.
The analysis of variance was applied for the workload of 2,000,000 msg, cf. Table 8. 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%.
The results of the Scott & Knoot test for workload of 2,000,000 msg are shown in Table 9. Regarding 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.

5.4.5 Discussion
In the simulation of the execution of the Coffee case study 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.
As the comparison of the two heuristics running the experiment over a period of time and measure the work done on the said, considering a continuous flow of messages arriving in the integration process. The performance of the FIFO heuristic degrades as the message arrival rate becomes higher than the system's capacity to process them.
The averages of the throughput in execution of integration processes were higher when using qPrior than FIFO. ANOVA test confirmed that the use of different heuristics generates a significant difference in the throughput, cf. Table 8. The coefficient of variation was 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. Table 9.

6 Conclusion
The challenge of enterprises to integrate their applications increases with large volumes of data from the cloud and with mobile applications. In face of that, tools for integration processes like integration platforms need to be improved. Integration platforms are tools that model and implement integration processes. An integration process is a computational program that allows applications to exchange data and functionalities. It can be represented by a workflow composed of tasks and communication

channels. Each task of an integration process carries out a specific computational operation that is executed by threads present in the runtime system of integration platforms.

This article contributes to the adaptation of integration platforms in the context of high volumes of data, by proposing a new task scheduling heuristic of integration processes. The qPrior heuristic separates tasks that await for available threads in multiple prioritized queues; the threads select the task to execute according to the priority of the queue. Every time a thread verifies a queue, it selects a predefined number of tasks to execute. We used the Particle Swarm Optimisation (PSO) meta-heuristic to find the near-optimal number of tasks that must be selected to reach better performance in the execution of integration processes. We implemented qPrior heuristic and simulated the execution of a classic integration process under high workloads. After that, we compared it with FIFO heuristic, which is traditionally used in large part of the open-source integration platforms. The results from executions showed that qPrior provides higher throughput than FIFO in integration processes overload cases. Those results were statistically analysed, and they showed there was significant improvement in the integration process execution performance when the task scheduling was performed by qPrior heuristic.

Regarding the research questions from the experiments:

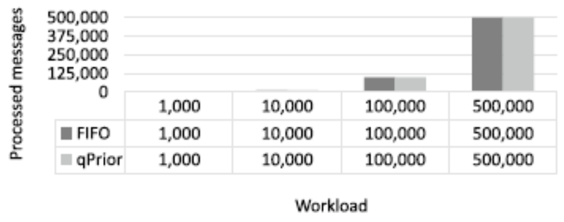
- RQ1: The proper choice for the number of tasks, which must be executed at each thread checking, impacts positively the performance of the task scheduling by qPrior heuristic. This number can be found by PSO meta-heuristic.
- RQ2: The qPrior algorithm provided much higher performance over throughput than FIFO for the execution of integration processes under overload.

Appendix 1: Additional metrics

The averages of processed messages in the execution of the Coffee shop integration process is our case study (CS) are shown in Figs. 7 and 8. The averages of remained messages in the execution of the CS are shown in Figs. 9 and 10. The makespan in the execution of the CS is shown in Figs. 11 and 12.

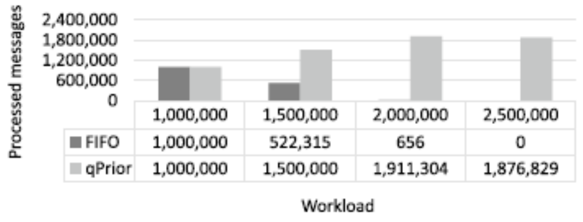
As for the number of messages processed, the FIFO heuristic performance degrades from 1,500,000 messages when it only processes approximately 35% of the total messages (Fig. 8). Figure 10 shows that 1,359,833 messages remained accumulated in the integration process, without processing, in the case of the FIFO heuristic. Figure 12 shows that from a workload of 1,500,000 messages arriving in a continuous flow in the integration process, the heuristic consumes all the simulation time and cannot fully process the messages that entered the integration process.

The most important metric is the throughput, which shows the processing rate of the heuristics, cf. Fig. 6. For this metric, in all workloads, the qPrior heuristic performed better than FIFO, and with the workload of 1,500,000 messages, FIFO's performance was only 7% of the performance obtained by qPrior. Thus, the qPrior heuristic is an option for scenarios of low workloads (<1,500,000 messages) and which continues to perform well for high volumes of messages ($\geq$ 1, 500, 000 messages). At the same time, FIFO is only an option for low workloads and still under-performing.



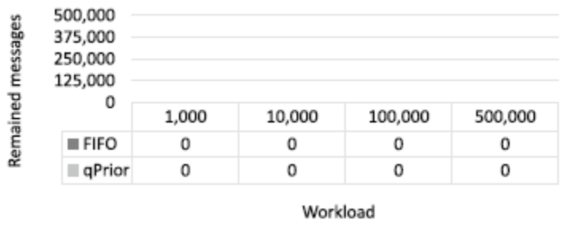
	1,000	10,000	100,000	500,000
▨ FIFO	1,000	10,000	100,000	500,000
▨ qPrior	1,000	10,000	100,000	500,000

Fig. 7 Average processed messages—low workload



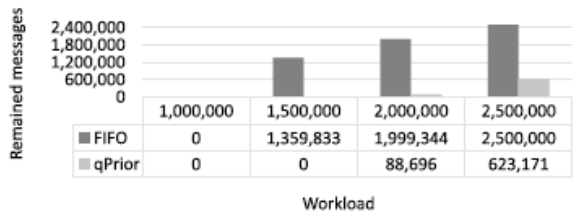
	1,000,000	1,500,000	2,000,000	2,500,000
▨ FIFO	1,000,000	522,315	656	0
▨ qPrior	1,000,000	1,500,000	1,911,304	1,876,829

Fig. 8 Average processed messages—high workload



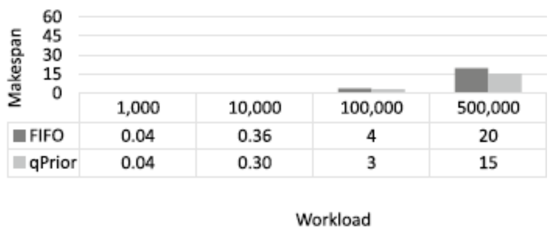
	1,000	10,000	100,000	500,000
▨ FIFO	0	0	0	0
▨ qPrior	0	0	0	0

Fig. 9 Average remained messages—low workload



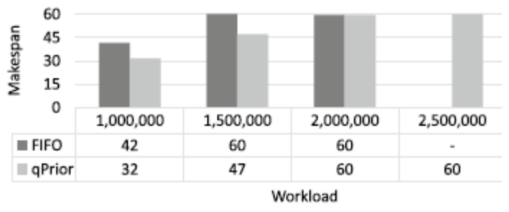
	1,000,000	1,500,000	2,000,000	2,500,000
▨ FIFO	0	1,359,833	1,999,344	2,500,000
▨ qPrior	0	0	88,696	623,171

Fig. 10 Average remained messages—high workload



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

Fig. 11 Average makespan—low workload



	1,000,000	1,500,000	2,000,000	2,500,000
▨ FIFO	42	60	60	-
▨ qPrior	32	47	60	60

Fig. 12 Average makespan—high workload

References

1. Abdulhamid S, Shafie AL, Idris I (2014) Tasks scheduling technique using league championship algorithm for Makespan minimization in IaaS cloud. J Eng Appl Sci 9(1):2528–2533
2. Alexander C, Ishikawa S, Silvertein M (1977) A pattern language: towns, buildings, construction. Oxford University Press, Oxford
3. 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. J Syst Softw 113:1–26
4. AlRashidi MR, El-Hawary ME (2009) A survey of particle swarm optimization applications in electric power systems. IEEE Trans Evolut Comput 13(4):913–918
5. Anwar N, Deng H (2018) Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. Future Int 10(5):1–23
6. Aron R, Chana I, Abraham A (2015) A hyper-heuristic approach for resource provisioning-based scheduling in grid environment. J Supercomput 71(4):1427–1450
7. Basili VR, Rombach D, Kitchenham KSB, Selby D, Pfahl RW (2007) Empirical software engineering issues. Springer, Berlin
8. Basu S, Karuppiah M, Selvakumar K, Li KC, Islam SKH, Hassan MM, Bhuiyan MZA (2018) An intelligentcognitive model of task scheduling for IoT applications in cloud computing environment. Future Gener Comput Syst 88:254–261
9. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K (2005) Task scheduling strategies for workflow-based applications in grids. Int Symp Cluster Comput Grid 2:759–767
10. Boehm M, Habich D, Preissler S, Lehner W, Wloka U (2011) Cost-based vectorization of instancebased integration processes. Inf Syst 36(1):3–29
11. Bratton D, Kennedy J (2007) Defining a standard for particle swarm optimization. In: Swarm intelligence Symposium, pp 120–127
12. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616
13. Chhabra A, Oshin (2018) Hybrid psacga algorithm for job scheduling to minimize makespan in heterogeneous grids. In: Industry Interactive Innovations in Science, Engineering and Technology, pp 107–120
14. Cruz CD (2006) Programa genes: estatística experimental e matrizes. Editora Universidade Federal de Viçosa
15. Cruzes DS, Ben Othman L (2017) Threats to validity in empirical software security research. In: Empirical Research for Sof. Security, pp 295–320
16. Eberhart R, Kennedy J (1995) Particle swarm optimization. In: Proceedings of the IEEE International Joint Conference on Neural Networks, pp 1942–1948
17. Fan K, Zhai Y, Li X, Wang M (2018) Review and classification of hybrid shop scheduling. Prod Eng 12(5):597–609
18. Feldt R, Magazinius A (2010) Validity threats in empirical software engineering research-an initial survey. In: International Conference on Software Engineering and Knowledge Engineering, pp 374–379
19. Frantz RZ (2012) Enterprise application integration: an easy-to-maintain model-driven engineering approach. PhD thesis, University of Seville

20. Frantz RZ, Corchuelo R, Molina-Jiménez C (2012) A proposal to detect errors in enterprise application integration solutions. J Syst Softw 85(3):480–497
21. Frantz RZ, Corchuelo R, Roos-Frantz F (2016) On the design of a maintainable software development kit to implement integration solutions. J Syst Softw 111:89–104
22. Freire DL, Frantz RZ, Roos-Frantz F, Sawicki S (2019) Survey on the run-time systems of enterprise application integration platforms focusing on performance. Softw Pract Exp 49(3):341–360
23. Ghafouri R, Movaghar A, Mohsenzadeh M (2019) A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds. Peer-to-Peer Netw Appl 12(1):241–268
24. Ghosh TK, Das S (2018) Job scheduling in computational grid using a hybrid algorithm based on particle swarm optimization and extremal optimization. J Inf Technol Res 11(4):72–86
25. Guo F, Yu L, Tian S, Yu J (2015) A workflow task scheduling algorithm based on the resources' fuzzy clustering in cloud computing environment. Int J Commun Syst 28(6):1053–1067
26. Harman M, Lakhotia K, Singer J, White DR, Yoo S (2013) Cloud engineering is search based software engineering too. J Syst Softw 86(9):2225–2241
27. Hilman MH, Rodriguez MA, Buyya R (2018) Multiple workflows scheduling in multi-tenant distributed systems: a taxonomy and future directions. ACM Comput Surv 1(1):1–33
28. Hohpe G (2005) Your coffee shop doesn't use two-phase commit [asynchronous messaging architecture]. IEEE Softw 22(2):64–66
29. Hohpe G, Woolf B (2004) Enterprise integration patterns: designing, building, and deploying messaging solutions. Addison-Wesley Professional, Boston
30. Jedlitschka A, Pfahl D (2005) Reporting guidelines for controlled experiments in software engineering. In: International Symposium on Empirical Software Engineering, pp 95–104
31. Jian C, Tao M, Wang Y (2014) A particle swarm optimisation algorithm for cloud-oriented workflow scheduling based on reliability. Int J Comput Appl Technol 50(3–4):220–225
32. Konsek H (2013) Instant Apache ServiceMix How-to. Packt Publishing
33. Kuhn R, Hanafee B, Allen J (2017) Reactive design patterns. Manning Publications Company
34. Lin SW, Ying KC (2019) Makespan optimization in a no-wait flowline manufacturing cell with sequence-dependent family setup times. Comput Ind Eng 128(1):1–7
35. Linthicum DS (2000) Enterprise application integration. Addison-Wesley Professional, Boston
36. Linthicum DS (2017) Cloud computing changes data integration forever: what's needed right now. IEEE Cloud Comput 4(3):50–53
37. Manasrah AM, Ali HB (2018) Workflow scheduling using hybrid GA-PSO algorithm in cloud computing. Wirel Commun Mobile Comput 2018:1–16
38. Manikas K (2016) Revisiting software ecosystems research: a longitudinal literature study. J Syst Softw 117:84–103
39. Pereira JL, o Varajão J, (2019) The temporal dimension of business processes: requirements and challenges. Int J Comput Appl Technol 59(1):74–81
40. Pietri I, Chronis Y, Ioannidis Y (2019) Fairness in dataflow scheduling in the cloud. Inf Syst 83:118–125
41. Pinto G, Castor F, Liu YD (2014) Understanding energy behaviors of thread management constructs. ACM SIGPLAN Not 49:345–360

42. Qureshi K, Shah SMH, Manuel P (2011) Empirical performance evaluation of schedulers for cluster of workstations. Cluster Comput 14(2):101–113
43. Ritter D, May N, Rinderle-Ma S (2017) Patterns for emerging application integration scenarios: a survey. Inf Syst 67:36–57
44. Ritter D, Rinderle-Ma S, Montali M, Rivkin A, Sinha A (2018) Formalizing application integration patterns. In: International Enterprise Distributed Object Computing Conference, pp 11–20
45. Ritter D, Rinderle-Ma S, Montali M, Rivkin A (2019) Formal foundations for responsible application integration. Inf Syst, p 101439
46. Rodriguez MA, Buyya R (2014) Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. Trans Cloud Comput 2(2):222–235
47. Rodriguez MA, Buyya R (2018) Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Future Gener Comput Syst 79:739–750
48. Russell J, Cohn R (2012) Jitterbit integration server. Book on Demand
49. Saifullah A, Li J, Agrawal K, Lu C, Gill C (2013) Multi-core real-time scheduling for generalized parallel task models. Real-Time Syst 49(4):404–435
50. Shoukry A, Khader J, Gani S (2019) Improving business process and functionality using IoT based E3-value business model. Electron Mark 1:1–10
51. Stavrinides GL, Karatza HD (2018) A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments. Multimed Tools Appl 78:24639–24655
52. Sun D, Yan H, Gao S, Liu X, Buyya R (2018) Rethinking elastic online scheduling of big data streaming applications over high-velocity continuous data streams. J Supercomput 74(2):615–636
53. Surhone LM, Timpledon MT, Marseken SF (2010) Petals ESB. Betascript Publishing
54. 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
55. Verma A, Kaushal S (2015) Cost minimized pso based workflow scheduling plan for cloud computing. Int J Inf Technol Comput Sci 7(8):37–43
56. Wang C, Zhang L, Liu C (2018) Adaptive scheduling method for dynamic robotic cell based on pattern classification algorithm. Int J Model Simul Sci Comput 9(5):1850040–1–1850040–18
57. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in soft. engineering. Springer, Berlin
58. Wood DC, Forman EH (1971) Throughput measurement using a synthetic job stream. In: Fall Joint Computer Conference, pp 51–56
59. Xie Y, Zhu Y, Wang Y, Cheng Y, Xu R, Sani AS, Yuan D, Yang Y (2019) A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment. Future Gener Comput Syst 97:36–378