Task scheduling characterisation in enterprise application integration

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

Abstract
Cloud computing allows enterprises to incorporate applications and computational resources as services, and thus, enterprises can concentrate on their business processes, without concerning the development, configuration and maintenance of these applications and resources. Integration platforms are one of these services that allow enterprises to integrate applications in order to reduce the maintenance costs and operations of the integration of on-premises platforms. However, high performance on resources offered by the cloud, demands improvement in task scheduling of integration platforms. Our literature review has identified a lack of studies in the field of enterprise application integration, focusing on specificities and vulnerabilities of the task scheduling of integration processes. This is a pioneer work regarding the characterisation of the scheduling of tasks of integration processes. We propose a ranking according to their conceptual models and apply this ranking to five integration processes. Then, we have statistically analysed the influence of each component of their conceptual models on the performance of the execution of these integration processes. We characterise the task scheduling of integration processes and presented a mathematical equation for the makespan as a function of the components of this characterisation. This study can guide software engineers in the optimal task scheduling for integration processes, which can improve the performance runtime systems regarding using the computational resources and result in minimisation of costs of companies.

1 Introduction

Contemporary business processes are involved in environments that deal with large, fast and varied amounts of data from several different sources and types of devices [73]. These processes are supported by a set of on-premise applications and cloud services that compose the software ecosystem of enterprises [60]. Cloud computing services, such as Software-as-a-Service (SaaS) and Infrastructure-as-a-Service (IaaS), have brought up several advantages for enterprises. SaaS allows the incorporation of applications, and IaaS enables the expansion of computational power, reducing the efforts of enterprises with development, deployment, maintenance of applications and computational resources to execute these applications. Besides, in cloud computing, billing is frequently based on the pay-as-you-go model, which allows enterprises to pay only for the services that they consume [15]. Internet-of-Things (IoT) is another contemporary advantage. IoT improves overall business possibilities, but it implies in dealing with a large amount of business data to obtain exact information on business assets and to analyse business requirements [82]. In this context, it is a challenge to make these on-premise applications and cloud services interoperate to exchange data and share functionalities.

Enterprise application integration (EAI) is the field of research that enables applications and services to exchange data and share functionality to respond to demands of the business processes efficiently. EAI provides methodologies, techniques and tools for the integration of applications. The integration platform is a crucial tool to develop integration processes. Integration platforms are traditionally deployed on-premise, but recently, they have become of interest to be deployed and provided on the cloud as Integration-Platform-as-a-Service (iPaaS). iPaaS represents a suitable solution, especially for small- and medium-sized enterprises because it reduces the maintenance costs and operations of the integration platforms when compared to deploying integration platforms on-premise [13]. The integration platforms enable software engineers to design, run and monitor integration processes. An integration process implements a workflow composed by distinct atomic tasks that process messages that flow into it. Many of the open-source integration platforms have been supporting a set of conceptual integration patterns documented by Hohpe and Woolf [44], as well as following the Pipes-and-Filters architectural style [1]. Some of these open-source integration platforms are ServiceMix [49], Petals [84], Jitterbit [78] and Guaraná [30, 31]. Pipes represent message channels and filters represent atomic tasks that implement a particular integration pattern to process information wrapped in messages. The runtime system is the component of the platform that is responsible for the execution of integration processes [30], and it plays a central role in task scheduling [39, 42].

Task scheduling concerns the time of task execution and the computational resources that perform the task. 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 [35]. Besides, it is necessary to optimise the use of computational resources because processing a large amount of data from IoT requires more allocation of resources [25, 82]. In situations of high input rate, the scheduling of tasks at the workflow tends to focus on tasks in the beginning of the workflow, since the most common scheduling heuristic used in integration platforms execution engines is FIFO. This focus on initial tasks causes the threads to execute them more frequently, in detriment of the other tasks. This behaviour impacts negatively on the performance of the integration process. To measure the performance, we adopt the makespan, a known metric, defined as the total execution time of the integration process for a given message [16, 19].

Messages coming from the cloud and mobile applications, common in contemporary environments, combined with messages coming from traditional enterprise environments, have to be integrated and processed while guaranteeing high throughput [74]. The current integration platforms present design challenges for providing near-real-time responsiveness. These platforms manage petabyte-scale data, distributing integration processes across contemporary environments ranging from traditional on-premise servers to cloud systems and mobile devices [52, 89]. Enterprises also face the challenge to suit and integrate their applications, together with the optimisation of resource usage to save costs [37, 39, 41]. A recent study in EAI identified research directions regarding fair execution of tasks of integration processes by optimising the task scheduling and the resource allocation of the runtime systems of the integration platforms [32].

There are many studies and proposals of algorithms [14, 39, 46, 59, 63, 67, 72, 76, 83, 86, 89, 90, 93, 95] regarding task scheduling, but none of them deals singularly with scheduling integration, such as tasks and computational resources are unknown, unpredictable paths to message processing, the task processing time variable. The scheduling of integration processes has quite specific characteristics, such as unknown message arrival rate, variable task processing time, unpredictable path at 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 concurrent execution in different resources. Without re-engineering the integration platforms, it is not possible to ensure their suitability when dealing with contemporary environments. Besides, it is not possible to ensure that the enterprises will take advantage of the scalability provided by cloud computing or optimise computational resource usage. The cloud scalability can increase their productivity, and the optimisation of computational resource usage can reduce their costs [56]. In the cloud context, the efficiency of runtime systems is fundamental because the performance of the execution of integration processes directly impacts the financial costs of enterprises. This relation between efficiency and cost occurs due to the pay-as-you-go model adopted by the cloud. The re-engineering of the runtime systems points to the need for improvement in task scheduling of integration processes since this is a critical activity in runtime systems.

In "Survey on the run-time systems of enterprise application integration platforms focusing on performance" [32], we provided a comparison framework for integration platforms and then used it to evaluate nine open-source integration platforms. Besides, we pointed open research directions regarding the performance of integration platforms to the context of cloud computing, amongst them, task scheduling.

This article is the first step towards dynamic scheduling for the execution of integration processes. Our goal is to contribute to a better understanding of task scheduling of integration processes by researchers and software engineers, to the platforms to be endorsed with functionality to deal with the new challenges of the cloud computing environment, as well as to take more advantage of the computational power of this environment. As far as we know, this is the first work that deals with the characterisation of the scheduling of tasks of integration processes. This characterisation allows abstract the complexity and diversity of the workflows.

We reviewed the current approaches regarding task scheduling by analysing their models, classification, problems and methods used. Based on this review, we have characterised the task scheduling of integration processes carried out by runtime systems. Our first contribution was the adaptation of the representation of integration processes found in literature [75]. We have posed a representation that typifies the tasks of integration processes according to their type of logic operation. Then, we have applied this representation to five integration processes. The second contribution was a classification of the most common types of disturbance of events related to integration models, messages, computational resources, the tasks and queues. The third contribution is a mathematical model for the makespan as a function of the elements from this characterisation, obtained by step-wise multiple linear regression statistic technique.

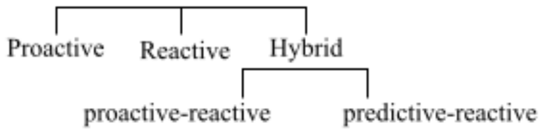
Based on the findings of this article, we propose a new task scheduling heuristic for the execution of integration processes under high workloads in another article entitled "Queue-priority optimised algorithm: a novel task scheduling for run-time systems of application integration platforms" [35].

The rest of this article is organised as follows: Sect. 2 provides background information on task scheduling and its challenges and guidelines; Sect. 3 discusses the work related to task scheduling; Sect. 4 presents the problem formulation; Sect. 5 applies the proposed representation for tasks of integration processes; Sect. 6 presents an experiment to define a mathematical model for makespan; and Sect. 7 presents our conclusions and future work.

2 Background

In this section, we introduce the main concepts, classifications, methods and approaches regarding task scheduling. First, we define the concept of scheduling and outline the dynamic scheduling, which is usually found in real-world problems [18]. After, we approach the challenges and guidelines to improve task scheduling. Lastly, we introduce the main concepts concerning information of application integration.

Fig. 1 Dynamic scheduling approaches



2.1 Scheduling definition

Scheduling is the decision-making process, in which it is necessary to answer «what » , «when » and «where » to perform a job [65]. «What » refers to the decision about the set of tasks that must be carried out to complete a job. «When » refers to the decision about the set of time intervals associated with tasks that must be carried out. «Where » refers to the decision about the set of resources used to perform the tasks.

Scheduling covers two problems: «resource allocation » and «task sequencing and timing » . The former precedes the latter. Resource allocation addresses the decision about which resources perform which tasks, and according to Pinedo [68] and Blazewicz et al. [10], it can be described as follows:

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

Task sequencing and timing addresses the decision about how to distribute the execution of the tasks over time. Sequencing addresses the order in which tasks must be executed, whereas timing addresses the initial time that each task must be executed at. According to Baker and Trietsch [6], task sequencing and timing can be described as follows:

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

It is possible to find optimal solutions to resource allocation problems through mathematical models; however, the combinatorial nature of task sequencing and timing problems makes them complicated, and consequently, they are not efficiently solved through mathematical models [6, 10, 68].

2.2 Scheduling approaches

In most real-world environments, the schedule needs to fit the presence of a variety of unexpected factors, such as workload oscillations, task delays and resource overload. The use of static scheduling-based approaches becomes unfeasible and the near-optimal schedules based on approximated data, antiquate for practical use [85].

The «proactive » and «reactive » scheduling approaches are more appropriated to deal with environmental uncertainties [2, 22, 61]. Chaari et al. [18] classify the dynamic scheduling as «proactive » , «reactive » and «hybrid » , cf. Fig. 1.

«Proactive » scheduling deals with uncertainties in design time. It produces one or more scheduling endowed with flexibility that becomes responsive to uncertainties. Thus, this approach is also called robust scheduling [55]. «Reactive » scheduling is used in highly disturbed environments where uncertainties are both frequent and abundant; thus, decisions are quickly made during runtime. «Hybrid » scheduling can be «proactive-reactive » or «predictive-reactive » . The «proactive-reactive » approach produces a set of static schedules and adopts one of them during runtime. Then, the decision-making is not made during runtime, but the scheduling can change to a previously defined scheduling capable of managing the current disturbance. The «predictive-reactive » approach has a deterministic schedule and adaptive scheduling. While the former produces schedules in design time, the latter makes decisions during runtime, reacting to disturbances.

2.3 Challenges and guidelines

Parunak [65] outlines the main challenges in scheduling: desirability, stochasticity, tractability, chaos and decidability. Desirability means finding a schedule that provides results that are closer to the ones expected by the company. Mathematical and dynamic programming [57] and evaluation [50] are the most used approaches to address this challenge. Mathematical and dynamic programming seeks optimal scheduling using interacting constraints; however, in complex environments, this can demand excessive computational resources. Evaluation techniques experiment and evaluate several solutions by simulations or by simplification of the behaviour of the environment. However, in complex environments, the exhaustive search of the set of solutions to evaluate can demand excessive time.

Stochasticity concerns the deviation of the real parameters defined in the computational model that is used for scheduling. Such deviation arises from unexpected events, such as breakdowns, overload, bottlenecks, variation in processing times, and interference. Simulation techniques can efficiently provide mechanisms to describe pseudo-random samples from a variety of standard statistical distributions, but the number of required simulations to encompass a statistically significant number of events can become prohibitive. Rescheduling [7], deferred commitment [92] and tweaking [38] are the standard techniques for this challenge. Rescheduling monitors the deviation, changing the schedule when this deviation exceeds a given constraint; however, this technique is only feasible when the time required to schedule is short enough to accompany changes in the environment. Deferred commitment scheduling designs high-level decisions and during runtime, refines these decisions based on the current status of the environment. Tweaking analyses the deviation of a mathematical model of the scheduling within certain limits to decide when to reschedule.

Tractability concerns the computational difficulty to analyse complex environments behaviours in reasonable time and cost. Usually, heuristic techniques are used to find a near-optimal solution, such as dispatching rules [4], constraint propagation [91], stochastic search [64] and predictive performance modelling [87]. Dispatching rules select the next task to be executed at a resource, obeying predefined rules. Constraint propagation deals directly with predefined constraints, identifying interactions and performing changes to reduce disparity amongst them. The stochastic search uses metaheuristics that allow the search in a wider solution space from an initial set of random solutions. Predictive performance modelling extrapolates historical observations to predictions of future situations without requiring detailed information about the workload.

Chaos concerns an environment where the conditions never repeat themselves and small differences in the initial conditions can result in exponential divergences in the environment during runtime, so it is not possible to predict its behaviour. The techniques suggested are bidding interval [23], information uncertainty [17] and structure of chaos [45]. Bidding interval seeks a threshold below which the environment is not chaotic. Usually, this value is determined by simulation. Information uncertainty inserts noise to the environment to block chaotic behaviour. Although this strategy conflicts with the problem of stochasticity, it can be used as a component in a compromised solution. Structure of chaos identifies configurations of the environment favourable to chaos in order to avoid them.

Decidability concerns the capacity of analysing environmental behaviour in order to identify undecidable problems, i.e. problems for which there is no computationally feasible solution in a reasonable time. A strategy used in such problems is the participatory scheduling, which involves job processing as a partner in the scheduling process. Techniques, as neural models [24], allow the computational adaptation or learning, adjusting the model to the characteristics of the problem domain. Participatory scheduling also addresses other challenges, and it deals with the complexity through heuristic search, the stochasticity through real-time distributed computation and the desirability through adaptive behaviour.

2.4 Integration definitions

An integration process is a workflow composed by segments of tasks connected through communication channels. These segments can be of sequential or parallel tasks, or both. Messages flow through tasks to their entire processing. A message wrappers data of user requests and has a header and a body. The header contains custom properties, and the body has the payload data. A message can be split into one or more messages in the workflow, as well as two or more messages can be merged into a unique message. A path is a specific segment connecting starting tasks to ending tasks, in which a message is entirely processed in an integration process. There is a dependence order for tasks at a path in which they must be executed, so that a message can only be processed by a task after this message has been processed by every predecessor task. However, parallel segments contain tasks that can be executed in parallel. A task implements an integration pattern, which represents an atomic operation, such as transforming, filtering, splitting, joining or routing. An outbound message of a task is written to the communication channel that connects this task with the next successor task at the workflow path.

The runtime system is the element of the integration platforms responsible for the execution of the integration processes. Usually, runtime systems have a scheduler and a thread pool. The scheduler orchestrates the activities and computational resources required for the accomplishment of the message processing. Thus, it is the central element of the runtime system. Threads are computational resources that execute the tasks and are usually grouped in a thread pool. A thread is the smallest sequence of a computational program that can be managed by the runtime system. The execution model of runtime systems establishes how they must execute tasks and allocate threads during the processing of messages in an integration process [33]. Regarding execution of integration processes, there are two models in the literature: process-based and task-based [3, 11, 12, 29]. In the process-based model, a thread executes every task of the workflow over an inbound message that flow throughout the process. The thread is released when it finishes the execution of every task in the workflow. In the task-based model, a thread executes the task over the inbound message that is being processed by 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. In this article, we address the task-based execution model, in which the scheduling of tasks of the integration process follows a first-in-first-out (FIFO) policy.

3 Related work

This section provides a literature review of the recent tasks scheduling approaches that deal with dynamic environments. Most of the related works are included in two primary goals. One of them manages the applications in multi-tenant cloud computing, and another deals with machine breakdowns and energy consumption in manufacturing systems. The goal of this research has a distinct purpose: the characterisation of task scheduling to application integration on the cloud. The integration-Platform-as-a-Service became imperative to handle the current amount of data and to save computational resource use. In 2017, two-thirds of application integration projects were projected by Cloud integration [66], being iPaaS the favourite deployment for the integration platform. According to "Magic quadrant for enterprise integration platform as a service 2017", the annual yield for iPaaS increases higher than the implementation of on-premise integration [40, 81]. The same task scheduling can be used by both on-premise and cloud implementations. However, in on-premise integration, the cost by computational resources consumption does not vary, whereas, on the cloud, this cost is proportional to using time. So, it was the cloud environment that justified the new approach of task scheduling.

We have structured this review, classifying the articles by research field, scheduling type and the method used to address the scheduling problem. We have discussed each article separately and summarised them in Table 1.

Huang and Süer [46] aimed to handle conflicts amongst different performance measures in a manufacturing system. Their strategy was to identify the combinations of dispatching rules that lead to scheduling that more accurately meet the expectations of the decision makers of the company. They adopt the predictive scheduling approach and proposed a dispatching rule-based genetic algorithm (GA) with fuzzy satisfaction levels. The performance metrics used were makespan, average flow time, maximum tardiness and total tardiness. Guo et al. [39] focused on the mapping

Table 1 Related works summary

References	Goal	Field	Type	Method
Huang and Süer [46]	Minimise makespan, average flow time, maximal tardiness and total tardiness	Manufact.	Predictive	GA and Fuzzy
Guo et al. [39]	Minimise makespan	Cloud	Reactive	Fuzzy
Nouiri et al. [63]	Minimise makespan	Manufact.	Predictive	PSO
Rimal and Maier [72]	Minimise makespan, cost, and tardiness	Cloud	Reactive	Dynamic heuristic
Zhou et al. [95]	Maximise completion ratio and minimise bandwidth consumption	Broadcast	Predictive	EDS
Wang et al. [86]	Minimise makespan and resource utilisation rate	Manufact	Reactive	Exact method and local search
Zaourar et al. [93]	Minimise makespan and energy consumption	Manufact.	Reactive	PSO
Manasrah and Ali [59]	Minimise makespan and cost; and balancing load	Cloud	Predictive	GA and PSO
Yahouni et al. [90]	Minimise tardiness	Manufact.	Predictive	Greedy
Rodriguez and Buyya [76]	Minimise makespan, cost, %deadline, and VM	Cloud	Reactive	Dynamic heuristic
Anwar and Deng [5]	Minimise makespan and cost	Cloud	Reactive	BoTs and MIP
Sun et al. [83]	Minimise makespan and resource utilisation rate	E-Stream	Reactive	EFT and max-min fairness
Buddala and Mahapatra [14]	Minimise makespan	Manufact.	Reactive	Teaching-learning
Ghafouri et al. [37]	Minimise makespan and cost	Cloud	Reactive	Back-tracking and non-critical tasks
Xie et al. [89]	Minimise makespan and cost	Cloud-Edge	Predictive	PSO
Pietri et al. [67]	Minimise makespan, cost, and unfairness	Cloud	Predictive	Pareto
[Our Proposal]	Minimise makespan	EAI	Predictive	RR

of tasks to resources of precedence constrained workflow applications in cloud computing environments. They adopt the predictive scheduling approach for task mapping and proposed an algorithm based on the fuzzy clustering of resources. The goal was to minimise the makespan of workflow applications.

Nouiri et al. [63] aimed to minimise the effect of machine breakdowns in the schedule in manufacturing environments. They adopted a predefined schedule and proposed an algorithm, based on particle swarm optimisation (PSO) meta-heuristic, to solve the flexible job-shop scheduling problem under uncertainty. As the goal function, they used the minimisation of the makespan, i.e. the minimisation of the completion time of the last job [71]. Rimal and Maier [72] addressed resource management in the context of compute-intensive workflow applications in multi-tenant cloud computing environments. They adopted the reactive scheduling approach for resource provisioning and scheduling strategy by a dynamic heuristic-based algorithm. The goal was to minimise the makespan, the cost of execution of the workflow and the tardiness while maximising the resource utilisation within a given deadline. The authors defined makespan as a measure of the throughput of the system and tardiness as a measure of the completion time, exceeding the expected time. Zhou et al. [95] investigated the broadcast scheduling problem for disseminating data to periodic continuous queries. They adopted the predictive scheduling approach and proposed a variant of the standard EDF scheduling algorithm that uses dynamic priority for preemptive real-time broadcast to design scheduling algorithms to use in data broadcast environments. The performance metrics used were: completion ratio and bandwidth consumption.

Wang et al. [86] aimed to solve the problem of poor performance and inefficiency feature selection methods in robotic cells of manufacturing systems. Their strategy was to train the scheduler and switch scheduling strategies in real time. They adopted the reactive scheduling approach and proposed a pattern classification algorithm, based on a hybrid PSO. The performance index used was the makespan. Additionally, the total robot utilisation rate was measured to reflect the average busyness of the robot. Zaourar et al. [93] aimed to increase the performance concerning energy consumption in the heterogeneous computing systems. They adopted a reactive approach and proposed an algorithm based on an exact solving method and on a local search optimisation technique, which takes into account the heterogeneity both on the computing and communication sides. As the objective function, they sought to minimise the makespan and energy consumption. Manasrah and Ali [59] aimed the efficient allocation of tasks to the resources workflow applications in cloud computing environments. They adopted the predictive scheduling approach and proposed a hybrid algorithm based on PSO and GA meta-heuristics. The goal was to reduce the makespan and balance the load over virtual machines (VM) with a minimum total monetary cost.

Yahouni et al. [90] aimed to prevent disturbances in the execution of a manufacturing schedule previously planned. Their strategy was to use "groups of permutable operations" methods that proposed a family of schedules instead of only one. They adopted the predictive scheduling approach and a decision-aid algorithm, based on a greedy heuristic, to select the schedule that fits best the real state of the shop. The performance metric used was the maximum tardiness. Rodriguez and Buyya [76] addressed resource utilisation inefficiencies on a Workflow as a Service environment. They adopted the reactive scheduling approach for resource provisioning and scheduling strategy by a dynamic heuristic-based algorithm. The goal was to minimise the overall cost of leasing the infrastructure resources without compromising the deadline of workflows. The performance metrics used were: costs, percentages of deadlines, the total number of virtual machines leased and their average utilisation, and the average makespan. Anwar and Deng [5] aimed to find a map scheduling that optimises costs under an user-defined approach as a Mixed Integer Programming (MIP) problem and proposed an algorithm based on the Bag of Tasks (BoTs) technique, which groups the workflow into bags of tasks by the criterion of data dependency and priority constraints. After, the algorithm optimises the allocation of elastic, heterogeneous and dynamically provisioned cloud resources. The authors defined a Normalised deadline, an Improvement rate, which are metrics related to makespan and cost.

Sun et al. [83] aimed at a flexible online scheduling framework for big data streaming applications (E-Stream). They adopted an online scheduling strategy based on priority-based earliest finish time (EFT) first and schedule multiple graphs by a max-min fairness strategy. After, the algorithm optimises the allocation of elastic, heterogeneous and dynamically provisioned cloud resources. The performance metrics were the makespan and the fairness degree. The latter refers to the resource utilisation rate. Buddala and Mahapatra [14] aimed to minimise the effect of machine breakdowns in the scheduling in manufacturing environments. They adopted a reactive approach and proposed an optimisation algorithm based on teaching-learning to solve the flexible job-shop scheduling problem under uncertainty. As an objective function, they sought to minimise the makespan. Ghafouri et al. [37] tackled resource management in the context of workflow applications in cloud computing environments. They adopted the reactive scheduling approach for resource provisioning and proposed an algorithm based on back-tracking heuristic combined with the scheduling of critical and non-critical tasks. The goal was to minimise the makespan and the execution cost of workflow applications. Xie et al. [89] aimed at the efficient allocation of tasks to the different resources for meeting the real-time requirement of the cloud and edge computing environments. In these environments, the application data is transmission-intensive in business workflows. They adopted the predictive scheduling approach and proposed an algorithm based on PSO that employs nonlinear inertia weight with selection and mutation operations by a directional search process. The goal was to reduce makespan and cost. Pietri et al. [67] addressed the problem of scheduling multiple dataflows on heterogeneous clouds. They adopted the predictive scheduling approach and 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 goal was to minimise the makespan, cost and unfairness of workflow applications executions.

We proposed a task scheduling characterisation that takes into account the specific features of the integration processes execution. The approach to this kind of task scheduling is predictive, preferably those that use low complexity heuristic to deal with dynamic environments and make quick decisions based on the current environment status without increasing the response time of the integration processes.

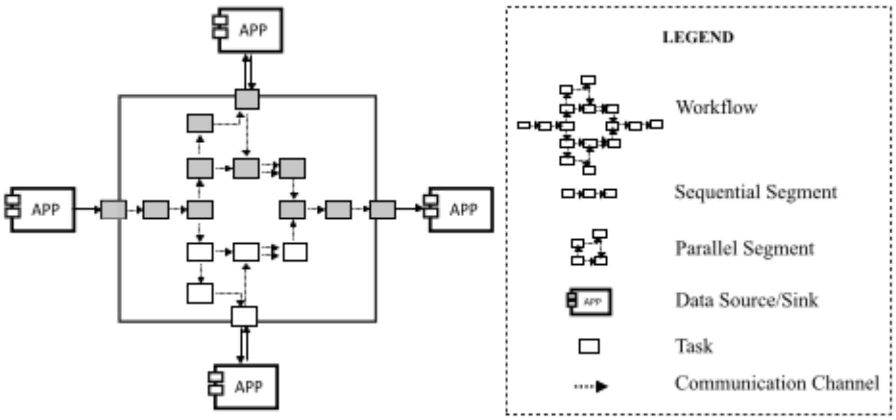


Fig. 2 Integration process conceptual model

We proposed a task scheduling characterisation that takes into account the specific features of the integration processes execution. The approach to this kind of task scheduling is predictive, preferably those that use low complexity heuristic to deal with dynamic environments and make quick decisions based on the current environment status without increasing the response time of the integration processes.

4 Problem formulation

In this section, we first describe the terminology and then formulate the problem. The terminology of the task scheduling problem in integration processes is based on the classic real-time scheduling theory and general-purpose parallel systems. We characterise three integration processes, according to this terminology. In the mathematical formulation, we introduce the problem definition and objective function. The former is the modelling and codification of the problem and the latter measures the adequacy of the heuristic in order to maximise the number of messages processed and to minimise the makespan.

4.1 Terminology

A conceptual model of an integration process is a workflow and is a set of segments, which in turn are composed of tasks uncoupled and connected by communication channels. An example of an integration process is depicted in Fig. 2, in which the larger rectangle represents the integration process. Small rectangles inside it represent tasks and arrows connecting tasks represent communication channels. Rectangles outside the integration process represent applications that are being integrated. The highlighted segment represents a possible path for a message in the workflow. We use the following terminology:

- Task is a computational code that implements an atomic operation.
- Communication channel is the means whereby messages pass from a task to another.
- Workflow is a set of atomic chained tasks by communication channels inside an integration process.
- Segment is a piece of a workflow that can be composed of tasks arranged sequentially, in parallel, or both.
- Path refers to a specific set of tasks, by which a message is entirely processed in an integration process.

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

The accomplishment of a job corresponds to receiving of one or more messages from one or more source applications, the processing of these messages by tasks of a path of an integration process, up to sending of one or more messages to one or more deliver applications. Generally, several jobs are processed at a particular point in time; that is, many service requests are fulfilled at a particular point in time. The execution model of runtime systems establishes how they must execute tasks of an integration process and allocate threads during the processing of messages [34]. The task scheduling of integration processes can be represented as a set of tasks on computational resources of the same capability, consisting of m threads.

4.2 Directed acyclic graph

The directed acyclic graph (DAG) represents task models for real-time scheduling, allowing the description of constraints on tasks execution [79]. In the DAG model, an integration process is described as a workflow W composed of k tasks, being an extension of the DAGs with weighted vertices (E_i, T_i), where $T_i = \left\{t_{i,1}, t_{i,2}, \cdots, t_{i,k}\right\}$ 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 it indicates precedence constraints between tasks. Every edge has a weight, which represents the waiting time of the task in the queue.

Ritter et al. [75] represented the integration process as a directed graph, called Integration Pattern Typed Graph (IPTG). IPTG was defined as a set of nodes T and a set of edges $E \subseteq T$ x T added to the function $type : T \rightarrow F$, where $F = \{start, end, message\ processor, fork, join, condition, merge, external\ call\}$. For a node $t \in T$, $\cdot t = \left\{t' \in T | (t' \cdot t) \in E\right\}$ for the set of direct predecessors of t, and $t\cdot = \left\{t'' \in T | (t \cdot t'') \in E\right\}$ for the set of direct successors of t.

The function *type* records what type of task each node represents. The first correctness condition claims that an integration pattern has at least one input and one output; the second condition indicates the cardinality of the tasks involved, i.e. the in-degrees and out-degrees of a node. The last condition states, «the graph (T, E) is connected and acyclic » , indicating that a graph represents only a task and its relation with its predecessor and successor tasks and that messages do not loop back to previous tasks. From the IPTG representation, we adopt the condition of verification, the classification by task cardinality and some terminologies, such as *type start*, *end*, *join*, *message processor*, and *external call*. However, as we have taken into consideration the logic operation of the task, we add *and*, *or*, and *or** function *types*. We named our representation as Integration Operation Typed Graph (IOTG).

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

- $\exists\, t_1, t_2 \in T$ with $type\left(t_1\right) = start$ and $type\left(t_2\right) = end$;
- if $type\ (t) \in \{and\}$ then $|\cdot t| = 1$ and $|t\cdot| = n$ and must produce messages to all n outputs;
- if $type\ (t) \in \{or\}$ then $|\cdot t| = 1$ and $|t\cdot| = n$ and produce message in at least one of its outputs;
- if $type\ (t) \in \{or^*\}$ then $|\cdot t| = 1$ and $|t\cdot| = n$ and produce message in by only one of its outputs;
- if $type\ (t) \in \{join\}$ then $|\cdot t| = n$ and $|t\cdot| = 1$;
- if $type\ (t) \in \{message\ processor\}$ then $|\cdot t| = 1$ and $|t\cdot| = 1$;
- if $type\ (t) \in \{external\ call\}$ then $|\cdot t| = 1$ and $|t\cdot| = 2$;
- the graph (T, E) is connected and acyclic.

4.3 Integration environment

In real-world problems, many events disturb the environment and degrade the performance of the execution of integration processes. We grouped the most common types of disturbance into five groups of events related to the integration model, to the messages, to the computational resources, to the tasks and to the queues, cf. Table 2. These events are also divided into internal and external. The former refers to events caused by internal elements that can be managed or monitored by the runtime system. The latter is caused by external elements, such as application, user requirements or data source.

Model-related events are caused by factors related to the integration process design. Some internal events are bottlenecks and dynamic routing. Regarding the bottlenecks, there are task arrangement patterns in integration process models, which, subject to certain conditions, indicate obstruction or delay the message processing [77]. The dynamic routing refers to the unpredictability of the path in which a message is processed. Usually, the integration process includes tasks that filter or route some specific messages; thus, the tasks in which a message will be processed are uncertain. Also, there are external events such as I/O delay and constraint change. I/O delay refers to the interruptions or delays caused by applications, database or any other component in which the message is processed. Constraint change

Table 2 Common disturbances in integration environment

Group	Event	
	Internal	External
Model-related	Bottleneck	I/O delay
	Dynamic routing	Constraint change
Message-related	–	Priority change
		Workload peak
		Random arrival rate
Resource-related	Unavailability	Constraint change
	Idleness	
	Deadlock	
Task-related	Starvation	–
	Random processing time	
Queue-related	Overload	Constraint change

refers to adjustments to meet new business requirements, such as a change in the message processing maximum time or message processing minimum rate.

Message-related events are caused by unexpected external events, such as change of priorities, workload peak and the random arrival rate. Priority change refers to a change in the order of message processing or task execution. Scheduling based on the priority of task specifies a decision criterion to select the task to be executed first, for example, the earliest finish time [20, 94]. Peak load refers to the simultaneous arrival of a large number of messages. Random arrival rate refers to the unpredictability of the arriving message number per unit of time.

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

Task-related refers to events related to the integration tasks that compose an integration process. Some internal events are starvation, random processing time and high processing time. Starvation refers to tasks that wait for a long time to be executed; for example, when a task has low priority and threads are always busy with high priority tasks. Random processing time refers to environments that deal with *Big Data* or *Internet of Things* [70], in which data vary in volume, variety, velocity and variability causing variation in the processing time of the tasks, which becomes unpredictable.

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

5 Study cases

In this section, we review five integration processes in the literature, in order to discuss their conceptual models, describe their integration logic and characterise their tasks according to function *types* of Integration Operation Typed Graph. We selected conceptual models designed in a single language in order to standardise the way the integration pattern was implemented. Initially, the models were designed using the Guaraná domain-specific language [28].

For the five integration processes, a starting task is represented by t_{start} and means that this task does not have predecessor tasks in the order of dependence. An ending task is represented by t_{end} and means that this task does not have successor tasks. It is possible to have one or more starting tasks and one or more ending task. A task that exchanges messages with applications during runtime is represented by t_x and an intermediate task is represented by t. There are several possible paths for the processing of a message, which were defined during design time. However, the path through which a message will flow depends on the integration process logic, so this path is only acknowledged during runtime. In DAGs, an edge between (t_i, t_j) represents a dependence between t_i and t_j, in which t_i is the predecessor node of t_j and t_j is the successor node of t_i.

These studies of the case are distinguished by types of tasks, types of segments and the number of tasks and communication channels. The studies of cases 1 and 5 have the same type of task (start, and, or, join, message processor, external call and end), but the first has only one start task and one end task, whereas the fifth has three start tasks and two end tasks. The studies of cases 2 and 3 have the same type of task (start, and, or*, join, message processor, external call and end), but the first has only one start task, whereas the third has two start tasks. The fourth study case has the only task of the types: start and join, message processor, external call, and end, but it is the single case that has three end tasks.

The primary benefit of enabling task logic modelling using logic operators is to abstract the complexity and variety of the computational operation of tasks such as transforming, filtering, splitting, joining, routing. This abstraction allows processes integration as directed acyclic graph to develop a mathematical model to predict performance metrics. The mathematical model is the function of a feature of every process integration, namely the number of every type of task and the number of sequential and parallel.

5.1 Study of case 1 (`SC1`): coffee shop integration process

The Coffee Shop problem is a benchmark of an integration process, introduced by Hohpe [43], whose conceptual model is depicted in Fig. 3. This process is an example of an integration process with three different paths for messages, where

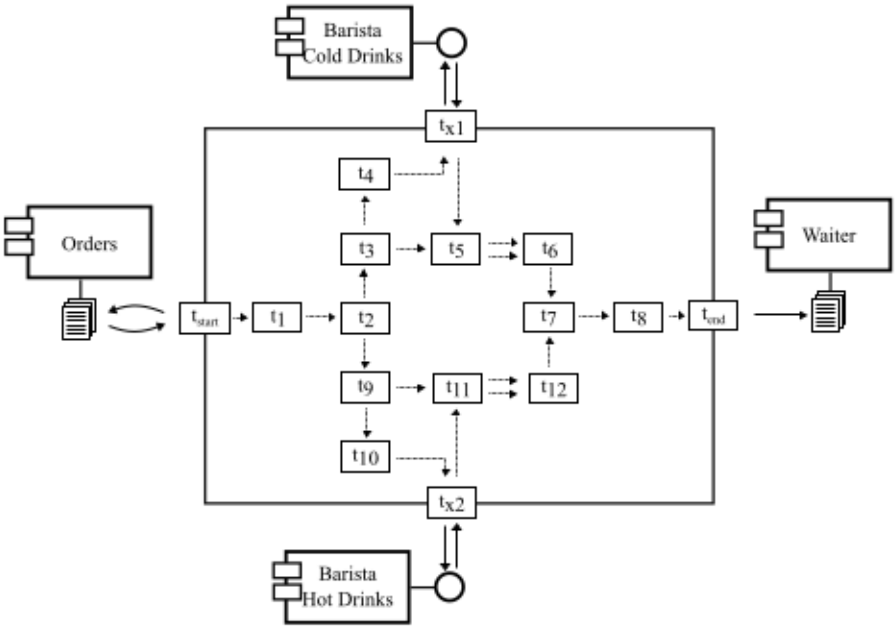


Fig. 3 Coffee shop conceptual model

the following types of tasks can be found: *start*, *and*, *or*, *join*, *message processor*, *external call* and *end*.

In the `SC1` integration process, the integrated applications are: «Orders » , «Barista Cold Drinks » , «Barista Hot Drinks » and «Waiter » . «Orders » represents the source application that delivers the data from the customer orders to the integration process. The data from the customer orders are wrapped inside messages. An order may include either hot or 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, which means that one or more instances of the job can be 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.

There is one input task represented by t_{start} and one output task represented by t_{end}. Tasks that exchange messages with applications during runtime are represented by t_{x1} and t_{x2}. Intermediary tasks are represented by t_i, where i ranges from 1 to 12. In the integration logic of this conceptual model, there may be customer orders containing only cold drinks, hot drinks or orders containing both cold and hot drinks. For each one of these types of customer orders, there is a path

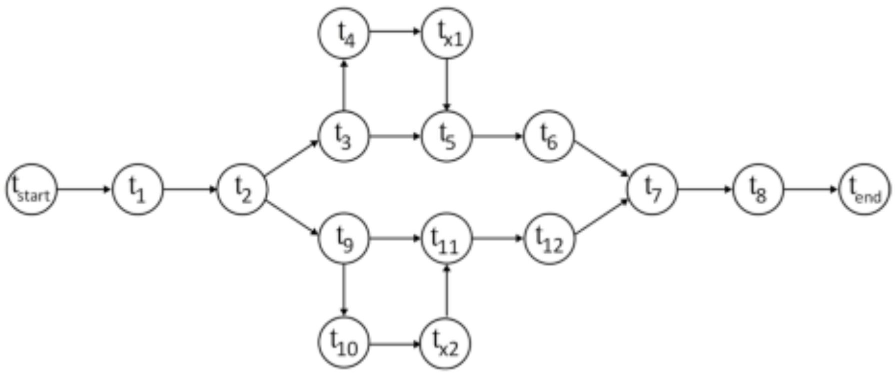


Fig. 4 Coffee shop represented in a DAG task model

Table 3 Coffee shop path characterisation

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

through which messages are processed. The possible paths were defined during design time by the `SC1` 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 at the Coffee Shop integration process are $[t_3, t_9]$, $[t_4, t_{10}]$, $[t_{x1}, t_{x2}]$, $[t_5, t_{11}]$, $[t_6, t_{12}]$. The Coffee Shop integration process is represented by a DAG in Fig. 4.

In `SC1` Integration Operation Typed Graph, there are 16 nodes, which represent the tasks, and there are 18 edges, which represent the channels. Node t_{start} is a starting node that represents a task of the type *start*. The nodes t_3 and t_9 represent tasks of the type *and*. The t_2 represents a task of the type *or*. The nodes t_5, t_7, and t_{11} represent tasks of the type *join*; nodes t_1, t_4, t_6, t_8, t_{10}, and t_{12} represent tasks of the type *message processor*; the nodes t_{x1} and t_{x2} represent tasks that send and receive information to/from applications and are tasks of the type *external call*; and, node t_{end} is an ending node that represents a task of the type *end*. The possible paths of the conceptual model `SC1` and their segments of tasks are shown in Table 3.

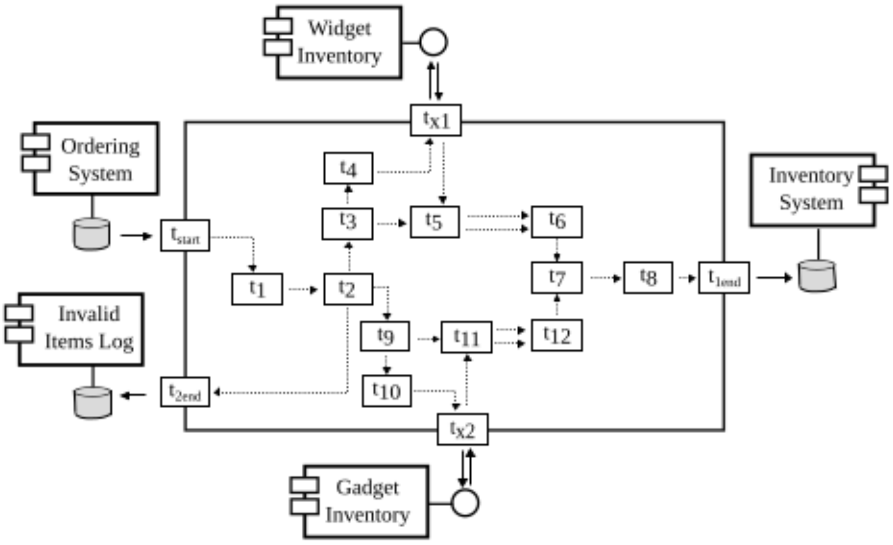


Fig. 5 Processing order conceptual model

5.2 Study of case 2 (SC2) : processing order integration process

The Processing Order problem is another benchmark of an integration process, introduced by Hohpe [43], whose conceptual model is depicted in Fig. 5. This process is an example of an integration process with three different paths for messages, where the following types of tasks can be found: *start*, *and*, *or**, *join*, *message processor*, *external call* and *end*. The SC2 has a different integration logic from the SC1 and, besides, in SC2 there are two ending tasks and one task of the type *or**.

In the SC2 integration process, the integrated applications are: «Ordering System » , «Widget Inventory » , «Gadget Inventory » , «Invalid Items Log » and «Inventory System » . «Ordering System » represents the source application that delivers the data of the new orders to the integration process. The data of the orders are wrapped inside messages. Every message with a new order is split into individual messages, each of which containing only one item. A message is routed to «Widget Inventory » or «Gadget Inventory » depending on their contents. Messages with items that do not belong to any of these inventories are routed to «Invalid Items Log » . The «Inventory System » application represents a final data sink that responds considering the availability of items. The processing of one order corresponds to one job instance.

There is one input task represented by t_{start} and two output tasks represented by t_{1end} and t_{2end}. The tasks that exchange messages with applications during runtime are represented by t_{x1} and t_{x2}. The intermediary tasks are represented by t_i, where i ranges from 1 to 12. In the integration logic of this conceptual model, an order contains several items. An order is split into unitary items, which can belong exclusively to one of the inventories, «Widget Inventory » and «Gadget Inventory » , or to none. There is a path for a unitary item that belongs to «Widget Inventory » ;

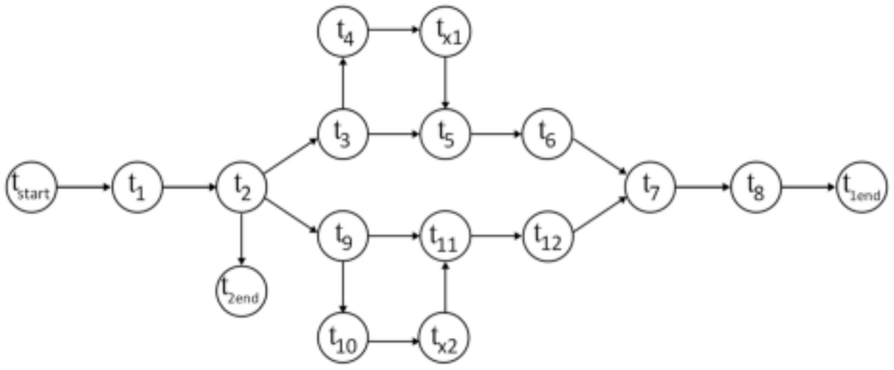


Fig. 6 Processing order represented in a DAG task model

Table 4 Processing order path characterisation

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

another path for a unitary item that belongs to «Gadget Inventory » ; and another path for a unitary item that does not belong to any inventory. Examples of tasks that can be executed in parallel in the `SC2` integration process are $[t_3, t_9]$, $[t_4, t_{10}]$, $[t_{x1}, t_{x2}]$, $[t_5, t_{11}]$, $[t_6, t_{12}]$. The `SC2` integration process is represented by a DAG in Fig. 6.

In the `SC2` Integration Operation Typed Graph, there are 17 nodes, which represent the tasks, and there are 19 edges, which represent the channels. Node t_{start} is a starting node that represents a task of the type *start*. The nodes t_3 and t_9 represent tasks of the type *and*. The t_2 represents a task of the type *or**. The nodes t_5, t_7, and t_{11} represent tasks of the type *join*; nodes t_1, t_4, t_6, t_8, t_{10}, and t_{12} represent tasks of the type *message processor*; the nodes t_{x1} and t_{x2} represent tasks that send and receive information to/from applications and are tasks of the type *external call*; and nodes t_{1end} and t_{2end} are ending nodes that represent tasks of the type *end*. The possible paths of the conceptual model `SC2` and their segments of tasks are shown in Table 4.

5.3 Study of case 3 (`SC3`) : Huelva's County council integration process

The Huelva's County Council problem is a real-world integration process [30] that consists on the automatisation of the user registration into a central repository. Its conceptual model is depicted in Fig. 7. This process is an example of an integration process with four different paths for messages, where the following types of tasks

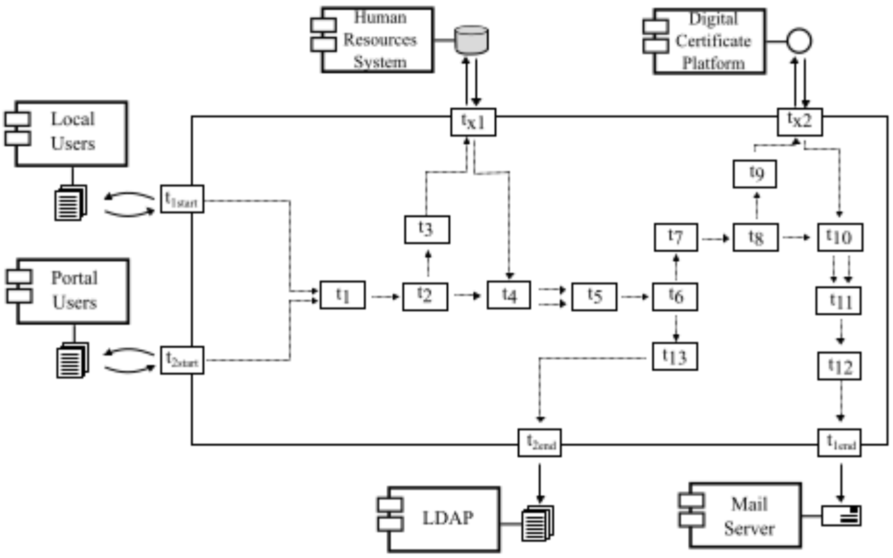


Fig. 7 Huelva's County council conceptual model

can be found: *start*, *and*, *or**, *join*, *message processor*, *external call* and *end*. The `SC3` has a different integration logic from the `SC1` and the `SC2` and, besides, in the `SC3` there are two starting tasks and four possible paths.

In the `SC3` integration process, the integrated applications are: «Local Users » , «Portal Users » , «LDAP » , «Human Resources System » , «Digital Certificate Platform » and «Mail Server » . The «Local Users » represents one of the source applications that manage the data of the users of the county council information systems. The «Portal Users » represents the web portal used to manage the users and is another source application. The «Human Resources System » represents the application that provides personal information about the employees, information like name and e-mail are required to compose notification e-mails. The «Digital Certificate Platform » represents the application that manages digital certificates. Finally, the «Mail Server » represents the application that runs the e-mail service and is used exclusively for notification purposes.

There are two input tasks represented by t_{1start} and t_{2start}, and two output tasks represented by t_{1end} and and t_{2end}. The tasks that exchange messages with applications during runtime are represented by t_{x1} and t_{x2}. The intermediary tasks are represented by t_i, where i ranges from 1 to 13. In the integration logic of the `SC3`, the data of users that arrive from t_{1start} and t_{2start} are replicated and one copy flows towards «Human Resources System » to search for information about the employee who owns a user record. Further, on t_6, the message is replicated, and one copy flows towards «LDAP » , and another flows towards «Digital Certificate Platform » . «Digital Certificate Platform » is queried using the email address included in the message. The sending of the certificate and its notification to the employee about the inclusion in the «LDAP » is made by «Mail Server » . There is a path for a local

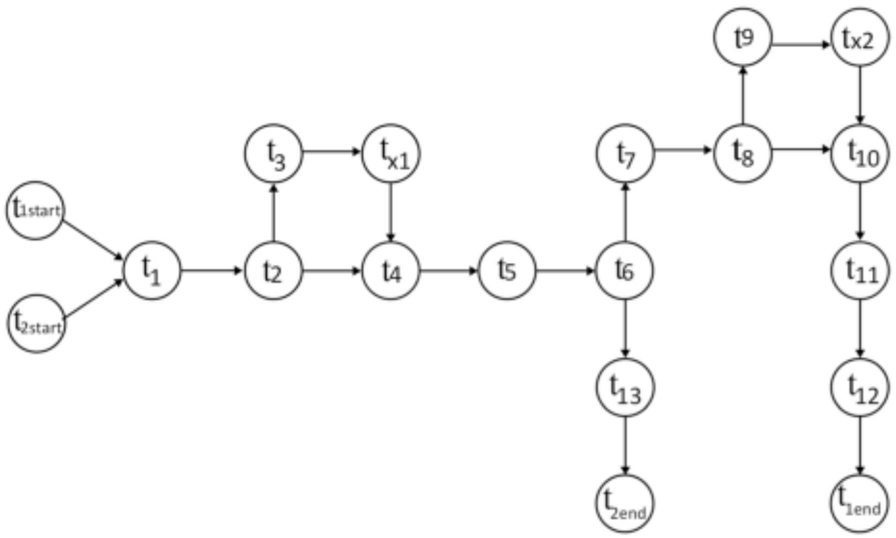


Fig. 8 Huelva's County council represented in a DAG task model

user that has an e-mail address; another path for a local user that does not have an e-mail address; another path for a web user that has an e-mail address; and another path for a web user that does not have an e-mail address. Examples of tasks that can be executed in parallel in the `SC3` integration process are $[t_3, t_7]$, $[t_9, t_{10}]$, $[t_7, t_{18}]$. The Huelva's County Council integration process is represented by a DAG in Fig. 8.

In the `SC3` Integration Operation Typed Graph, there are 19 nodes, which represent the tasks, and there are 20 edges, which represent the channels. Nodes t_{1start} and t_{2start} are starting nodes that represent tasks of the type *start*. The nodes t_2 and t_8 represent tasks of the type *and*. The t_6 represents a task of the type *or**. The nodes t_1, t_4, and t_{10} represent tasks of the type *join*; nodes t_3, t_5, t_7, t_9, t_{11}, t_{12}, and t_{13} represent tasks of the type *message processor*; the nodes t_{x1} and t_{x2} represent tasks that send and receive information to/from applications and are tasks of the type *external call*; and nodes t_{1end} and t_{2end} are ending nodes that represent tasks of the type *end*. The possible paths of the conceptual model `SC3` and their segments of tasks are shown in Table 5.

5.4 Study of case 4 `(SC4)` : Unijuí University integration process

The Unijuí University problem is a real-world integration process [30] that consists on the automatisation of the charging of personal phone calls from phone lines belonging to the university. Its conceptual model is depicted in Fig. 9. This process is an example of an integration process with single path for messages, where the following types of tasks can be found: *start*, *and*, *join*, *message processor*, *external call* and *end*. The `SC4` has a different integration logic from the previous cases, and besides, in `SC4`, there are three ending tasks and one single possible path.

Table 5 Huelva's county council path characterisation

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

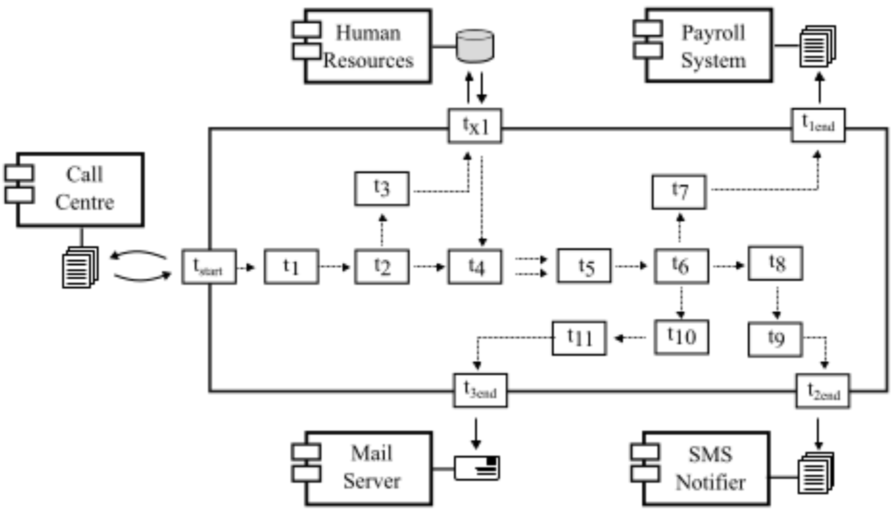


Fig. 9 Unijuí University conceptual model

In the SC4 integration process, the integrated applications are: «Call Centre » , «Human Resources » , «Payroll System » , «Mail Server » and «SMS Notifier » . The «Call Centre » records every call that every employee makes from a phone line belonged to the university. The code is also used to correlate phone calls with the information in «Human Resources » and «Payroll System » . The «Human Resources » supplies personal data concerning employees, and the «Payroll System » computes their wages. The «Mail Server » and the «SMS Notifier » notify employees about their charges. The former provides e-mail service and the later offers short message system services.

There is an input task represented by t_{start} and three output tasks represented by t_{1end}, t_{2end}, and t_{3end}. The task that exchanges messages with another application during runtime is represented by t_{x1}. The intermediary tasks are represented by t_i, where i ranges from 1 to 11. In the integration logic of the SC4, the data of users that arrive from t_{start} are replicated and one copy flows towards «Human Resources System » to search for information about the employee. Further on t_6, the message is replicated and one copy flows towards «Payroll System » , another flows towards «Mail Server » , and another, towards «SMS Notifier » . The sending of the notifications to the employees about the charge is made by «Mail Server » and «SMS Notifier » . Examples of tasks that can be executed in parallel in the SC4 integration process are $\left[t_7, t_8, t_{10}\right]$, $\left[t_9, t_{11}\right]$. The Unijuí University integration process is represented by a DAG in Fig. 10.

In the SC4 Integration Operation Typed Graph, there are 16 nodes, which represent the tasks, and there are 20 edges, which represent the channels. Node t_{start} is a starting node that represents a task of the type *start*. The nodes t_2 and t_6 represent tasks of the type *and*; The node t_4 represents a task of the type *join*; nodes t_1, t_3, t_5, t_7, t_8, t_9, t_{10} and, t_{11} represent tasks of the type *message processor*; The node t_{x_1} represents a task that sends and receives information to/from applications, and is a task of

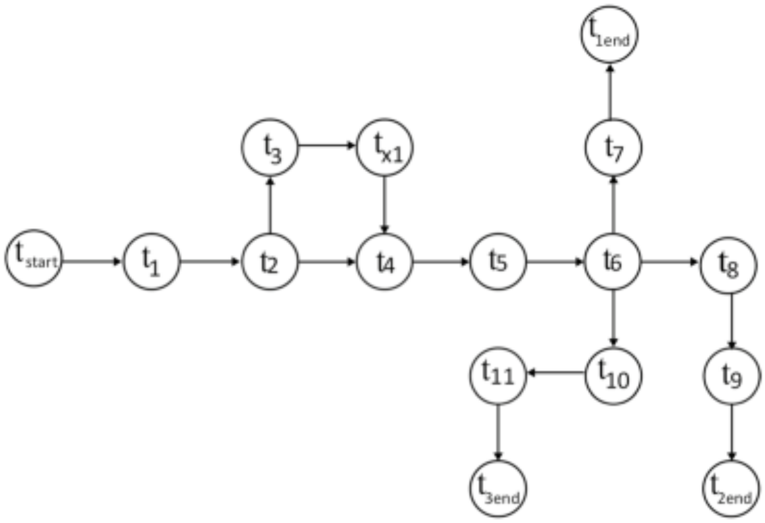


Fig. 10 Unijuí University represented in a DAG task model

Table 6 Unijuí University path characterisation

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

the type *external call*; and nodes t_{1end}, t_{2end}, and t_{3end} are ending nodes that represent tasks of the type *end*. The possible paths of the conceptual model `SC4` and their segments of tasks are shown in Table 6.

5.5 Study of case 5 `(SC5)` : real estate integration process

The Real State problem is a real-world integration process [33] that consists on the automatisation of the real estate tax management system used in Ijuí City, Brazil. Its conceptual model is depicted in Fig. 11. This process is an example of an integration process with three paths for messages, where the following types of tasks can be found: *start*, *and*, *or*, *join*, *message processor*, *external call* and *end*. The `SC5` has a different integration logic from the previous cases and, besides, in `SC5`, there are three starting tasks and two ending tasks and six possible paths.

In the `SC5` integration process, there are three input tasks represented by t_{1start}, t_{2start}, and t_{3start}; and two output tasks represented by t_{1end}, and t_{2end}. The tasks that exchanges messages with another application during runtime are represented by t_{x1} and t_{x2}. The intermediary tasks are represented by t_i, where i ranges from 1 to 10. In

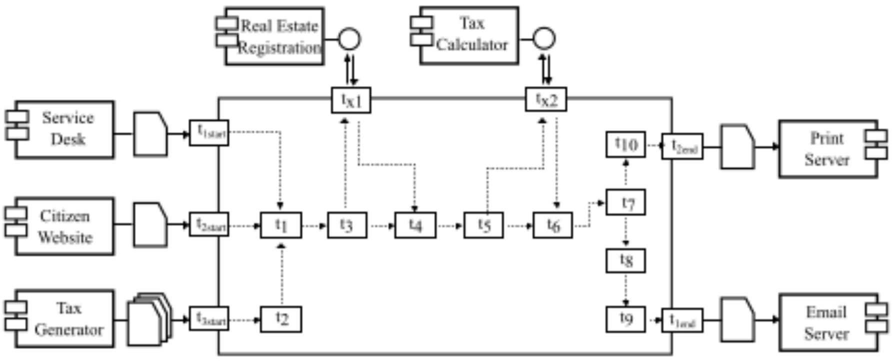


Fig. 11 Real Estate conceptual model

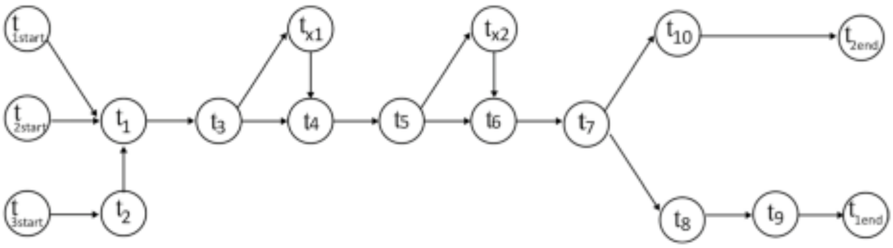


Fig. 12 Real estate represented in a DAG task model

the integration logic of the SC5, the requests of users that arrive from t_{start1}, t_{start2}, and t_{start3} flow towards «Real Estate Registration » to search for information about the citizens and their real estate. Further, on t_4, the message is enriched with information and flows towards «Tax Calculator » to calculate tax payment. If the citizen has and e-mail address, the message flows towards «Mail Server » and in all cases towards «Print Server » to further sending to the address of the citizen. There is a path for a «Service Desk » user that has an e-mail address; another path for a «Service Desk » user that does not have an e-mail address; another path for a «Citizen Website » user that has an e-mail address; another path for a «Citizen Website » user that does not have an e-mail address; another path for a message from the «Tax Generator » where the user has an e-mail address; and another path for a message from the «Tax Generator » where the user does not have an e-mail address. Examples of tasks that can be executed in parallel in the SC5 integration process are $\left[t_8, t_{10}\right]$. The Real Estate integration process is represented by a DAG in Fig. 12.

In the SC5 Integration Operation Typed Graph, there are 17 nodes, which represent the tasks, and there are 18 edges, which represent the channels. Nodes t_{1start}, t_{2start} and t_{3start} are starting nodes that represent tasks of the type *start*. Nodes t_3 and t_5 represent tasks of the type *and*. Node t_7 represents a task of the type *or*. Nodes t_1, t_4, and t_6 represent tasks of the type *join*. Nodes t_2, t_8, t_9, and, t_{10} represent tasks of the type *message processor*. Nodes t_{x_1}, t_{x_2} represent tasks that send and receive information to/from applications and are tasks of the type *external call*; and nodes

Table 7 Real estate path characterisation

Path	Segment	
	Sequential	Parallel
$\{t_{1start}, t_1, t_3, t_{x1}, t_4, t_5, t_{x2}, t_6, t_7, t_8, t_9, t_{1end}, t_{10}, t_{2end}\}$	$\{t_{1start}, t_1\}; \{t_7, t_8, t_9, t_{1end}\}; \{t_{10}, t_{2end}\}$	$\{t_3, t_{x1}, t_4\}; \{t_5, t_{x2}, t_6\}$
$\{t_{1start}, t_1, t_3, t_{x1}, t_4, t_5, t_{x2}, t_6, t_7, t_{10}, t_{2end}\}$	$\{t_{1start}, t_1\}; \{t_7, t_{10}, t_{2end}\}$	$\{t_3, t_{x1}, t_4\}; \{t_5, t_{x2}, t_6\}$
$\{t_{2start}, t_1, t_3, t_{x1}, t_4, t_5, t_{x2}, t_6, t_7, t_8, t_9, t_{1end}, t_{10}, t_{2end}\}$	$\{t_{2start}, t_1\}; \{t_7, t_8, t_9, t_{1end}\}; \{t_{10}, t_{2end}\}$	$\{t_3, t_{x1}, t_4\}; \{t_5, t_{x2}, t_6\}$
$\{t_{2start}, t_1, t_3, t_{x1}, t_4, t_5, t_{x2}, t_6, t_7, t_{10}, t_{2end}\}$	$\{t_{2start}, t_1\}; \{t_7, t_{10}, t_{2end}\}$	$\{t_3, t_{x1}, t_4\}; \{t_5, t_{x2}, t_6\}$
$\{t_{3start}, t_2, t_1, t_3, t_{x1}, t_4, t_5, t_{x2}, t_6, t_7, t_8, t_9, t_{1end}, t_{10}, t_{2end}\}$	$\{t_{3start}, t_2, t_1\}; \{t_7, t_8, t_9, t_{1end}\}; \{t_{10}, t_{2end}\}$	$\{t_3, t_{x1}, t_4\}; \{t_5, t_{x2}, t_6\}$
$\{t_{3start}, t_2, t_1, t_3, t_{x1}, t_4, t_5, t_{x2}, t_6, t_7, t_{10}, t_{2end}\}$	$\{t_{3start}, t_2, t_1\}; \{t_7, t_{10}, t_{2end}\}$	$\{t_3, t_{x1}, t_4\}; \{t_5, t_{x2}, t_6\}$

Table 8 Conceptual model characterisation

ID	No. of tasks-per type of function								No. of channels	No. of segments†	
	Start	And	Or	or*	Join	Message processor	External call	End		Sequential	Parallel
`SC1`	1	2	1	0	3	6	2	1	18	2	3
`SC2`	1	2	0	1	3	6	2	2	19	2	1
`SC3`	2	2	0	1	3	7	2	2	20	3	2
`SC4`	1	2	0	0	1	8	1	3	16	4	1
`SC5`	3	2	1	0	3	4	2	2	18	3	7

Considering the longest path

t_{1end}, t_{2end}, and t_{3end} are ending nodes that represent tasks of the type *end*. The possible paths of the conceptual model `SC5` and their segments of tasks are shown in Table 7.

5.6 Summary of the characterisation

It is possible to extract information not just from the conceptual models of integration processes [9, 51, 77], but also from the logic semantics of the task. Processes that have more than one task of the type *start*, benefit from the adoption of the task-based execution model because it is not necessary to have messages in all the inputs of the process in order to begin processing. Similarly, processes that have several tasks of the type *external call* also benefit from the task-based model because while a thread can be blocked waiting for a response of an external application, the other threads can execute the tasks of the integration process. Integration processes that have more than one task of the type *and*, would benefit from the use of parallel processing. Processes that have several tasks of the type *join*, may have a delay, mainly when these given tasks are involved in the correlation of messages. Processes with several communication channels demand more memory to store messages. The characterisation of the integration processes, concerning tasks, channels and paths, is shown in Table 8.

Usually, the message arrival is random and can be batch, periodic or in real time. The size and format of a message can also vary, from bytes to petabytes, structured or unstructured. This variety of messages causes a variation in the execution time of tasks. Regarding the total number of threads, although it is not limited, it is more realistic to assume that there is some constraint to it.

Parallel task processing is only possible when there are multiple physical cores in a machine where the integration process is executed. The number of cores directly impacts the execution of the threads created by the runtime system [58], and thus, it influences the performance of the execution of the integration process. The number of threads is configurable and, theoretically, unlimited. However, a high number of threads can degrade the performance of the execution because more time needs to be spent on managing these threads and shared resources, such as cache capacity or memory bandwidth, and tend to quickly saturate [53]. There are several possible configurations of thread pool that a runtime system can implement, such as (i) a thread pool with predefined and fixed number of threads, (ii) a thread pool with an unlimited number of threads that increases according to demand; (iii) a thread pool that distributes the workload amongst the available cores.

6 Experiment

In this section, we present an experiment to evaluate the makespan resulted in the current task scheduling of runtime systems of integration platforms, which use FIFO policy. We followed a protocol based on edlitschka and Pfahl [47], Wohlin et al. [88], and Basili et al. [8], with procedures for controlled experiments in the field of software engineering.

6.1 Research questions and hypotheses

This experiment aims to answer the following research questions:

RQ1: Is it possible to identify the variables more statistically significant to the makespan of integration processes?

RQ2: Is it possible to build a mathematical model to predict the makespan as a function of types of segments and tasks and the number of channels of integration processes?

Our hypotheses to these research questions are:

H1: Statistic techniques can identify the variables more statistically significant to the makespan of integration processes.

H2: The statistic technique of linear regression can build a mathematical model for the makespan of integration processes as a function of the workload, elements of the conceptual model and integration logic.

6.2 Variables

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

`Workload` (w). The number of input tasks, or workload, when the algorithm starts to perform the integration process. The values tested for this variable were: 100, 500,000, 1,000,000 and 1,500,000.
`Integration process`. The conceptual model of the integration process. The values tested for this variable were: `SC1`, `SC2`, `SC3`, `SC4` and `SC5`. Every conceptual model varies in:

- x_1: number of tasks of type *start*.
- x_2: number of tasks of type *and*.
- x_3: number of tasks of type *or*.
- x_4: number of tasks of type *or**.
- x_5: number of tasks of type *join*.
- x_6: number of tasks of type *message processor*.
- x_7: number of tasks of type *external call*.
- x_8: number of tasks of type *end*.
- x_9: number of channels.
- x_{10}: number of sequential segments.
- x_{11}: number of parallel segments.

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

`Makespan`. This variable corresponds to the average processing time of the job instances accomplished during the time interval of the experiment.

6.3 Environment and supporting tools

The experiments were carried out on a machine equipped with 16 processors Intel Xeon CPU E5-4610 V4, 1.8 GHz, 32GB of RAM, and Windows Server 2016 Datacenter 64-bits operating system. The programming language Java, version 8.0 update 152, was used to implement and execute the algorithms. The MATLAB [54] software, version R2018, was used to process the statistic analysis. The source code, data set and script of functions statistics used in this experiment are publicly available for download [1].

6.4 Execution and data collection

The experiments were conducted using a simulator built on Java, which simulates the execution of the integration processes using the FIFO policy. The simulation started with a probability distribution of these random messages is a continuous uniform distribution in [0,100] of random input messages and received an average of 100 new random input messages in a random time interval. The term "random" means that the time that a task spent to process a message varied within an interval, in microseconds. We assumed that tasks of types *start*, *end*, *message processor* and *external call* ranged 1-2 ms; *and*, *or*, *or** ranged 2-3 ms; and *join* ranged 3-4 ms. We configured the simulation time to 60 seconds, so, after this time, the simulator interrupted the current task executions. Then, the simulator collected the makespan and stored it in a text file. Afterwards, we handled and analysed the data and then applied statistic tests.

Usually, the results are statistically analysed by the method of the executions, in which 20-30 executions are sufficient to obtain an average population using the distribution with more extreme values than a normal distribution [80]; our experiments were repeated 25 times. For each integration process, we repeated the execution 25 times every four workloads, resulting in 500 different observations, which were summarised below:

Heuristics	FIFO	1
Workloads	100, 500,000, 1,000,000, and 1,500,000	4
Integration Processes	`SC1`, `SC2`, `SC3`, `SC4`, and `SC5`	5
Elapsed time	60 seconds	1
Rate of task input	100	1
Repetitions	1...25	25
Number of observations	$1 \times 4 \times 5 \times 1 \times 1 \times 25$	500

6.5 Results and discussion

We present the results of the metrics collected in the simulation in tables and charts for each integration process. The statistical theory was indicated to analyse data from experiments on performance [36], because it deals with non-determinism in computational systems, such as runtime systems of integration platforms [27]. We used the stepwise statistical method to build the mathematical model.

Regression analysis is a method to estimate the relation amongst the dependent variable makespan and the independent variable workload. In regression analysis, the correlation coefficient (R^2) is a parameter that determines the degree of linear correlation of variables and is defined by $R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$, where SSE is the sum of squared error and SST is the sum of squared total [48]. Thus, R^2 tends to 1 when the sum of squared error is too small when compared to the sum of the squared total. The *t*-statistic is used for making inferences about the regression coefficients.

Table 9 Estimated coefficients of the linear regression constant model

	Estimate	Standard error	t-Statistic	p Value
(Constant term)	0.0015759	0.0023158	0.68052	0.4965
w	8.2271×10^{-9}	2.4645×10^{-9}	3.3382	0.00090714
x_9	-0.00017883	0.0001079	-1.6574	0.098067
x_{10}	0.00047377	0.00019455	2.4352	0.015239
x_{11}	0.0062507	0.00019563	31.951	4.2878×10^{-122}
$w \cdot x_9$	-5.786×10^{-10}	1.1503×10^{-10}	-5.0299	6.8907×10^{-7}
$w \times x_{10}$	1.4763×10^{-9}	2.0393×10^{-10}	7.2393	1.7461×10^{-12}
x_{11}^2	-0.00070712	2.3798×10^{-5}	-29.714	7.2238×10^{-122}

Number of observations: 500, Error degrees of freedom: 492

Root Mean Squared Error: 0.00158

R^2: 0.769, Adjusted R^2 0.766 F-statistic versus constant model: 234, p Value = 4.88 $cdot10^{-152}$

The hypothesis test on coefficient x_i tests the null hypothesis that it is equal to zero against the alternate hypothesis that the coefficient is different from zero. The null hypothesis that equals to zero means that the corresponding term is not significant. The p value, probability value, is another common metric used to determine the significance of the model results when applying hypothesis testing. The result is considered statistically significant when p value is small, i.e. p value ≤ 0.05 [62].

Stepwise regression is an iterative method for adding and removing terms from a multilinear model based on their statistical significance in a regression. The method starts with an initial model and then compares the explanatory power of incrementally larger and smaller models. At each iteration, the p value of an F-statistic is calculated to test models with and without a potential term. The method finishes when no single step improves the model. We select an upper bounding model that has linear terms, interaction terms and squared terms. Then, we generate a quadratic model described by Eq. 1.

$$\overline{\text{makespan}} \sim 1 + w \cdot x_9 + w \cdot x_{10} + x_{11} + {x_{11}}^2 \tag{1}$$

The coefficients of Eq. 1 are indicated by the Estimate column in Table 9. The t-statistic column is calculated by the division of Estimate column and Standard Error column, i.e. $t\text{-statistic} = \frac{\text{Estimate}}{\text{Standard error}}$. The square root of the mean squared error estimates the standard deviation of the error distribution. The degrees of freedom for error are defined as the number of observations minus the number of coefficients in the model, including the constant term. "F-statistic versus constant model" tests whether the model fits significantly better than a degenerate model consisting of only a constant term.

There are eight coefficients and only the constant term , w, x_9, x_{10} and x_{11} terms are significant at a 5% significance level. These terms correspond, respectively, to these variables: workload, number of channels, number of sequential segments and number of parallel segments. The correlation coefficient, R^2, was 0.769, and the interval of deviations of the residuals varied from a minimum value of – 0.003 to a maximum value of 0.0208. The F-statistic of the linear fit versus the constant model is 234, with a p value of 4.88×10^{-152}. The model is significant at a 5% significance level. The R^2 0.769 means that the model explains about 77% of the variability in the response.

Regarding the research questions and hypotheses:

- **RQ1:** The workload, the number of channels, the number of sequential segments and the number of parallel segments are the variables that most influence the makespan of integration processes using the FIFO policy for their task scheduling.
- **RQ2:** A stepwise multiple linear regression with a high correlation coefficient, found a quadratic function to the makespan. Thus, this function can predict the makespan as a function of the workload, the number of channels and the number of sequential and parallel segments.

From the experiment, we confirmed our hypotheses for each of the research questions, respectively:

- **H1:** It is possible to identify the variables more statistically significant to the makespan of integration processes.
- **H2:** It is possible to build a mathematical model to predict the makespan as a function of their more statistically significant variables.

6.6 Threats to validity

In this section, we evaluate the threats that could influence the results of the experiment and how we tried to mitigate them, taking into consideration that threats to validity are present in any empirical research [21].

6.6.1 Construct validity

Construct validity discusses whether the planning and execution of the study are adequate to answer the research questions. We planned the experiment according to procedures from empirical software engineering [8, 47, 88]. Firstly, we defined our research question, formulated our hypotheses and defined the independent and dependent variables. After that, we provided information about the execution environment, supporting tools, execution and data collection. Then, we performed our simulation in two hundred different scenarios and used statistical techniques to evaluate the results.

6.6.2 Conclusion validity

As reported by Wohlin et al. [88], conclusion validity “are 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 heuristics used and that there was a significant difference amongst them.

6.6.3 Internal validity

Internal validity aims to ensure that the treatment caused the outcome, mitigating effects of other uncertain or not measured factors [26]. Instrumentation and source of noise are possible threats. We experimented the same machine, which was on security mode, with minimal features and disconnected from the Internet during the executions, in order to minimise interference in the execution time of the algorithm. We built our algorithm in Java, usually, the first executions of codes were slower, and it is advisable to let the virtual machine eventually perform code optimisation [69]. Then, we, firstly, executed the algorithm once only to warm the Java virtual machine up. Additionally, the researchers accurately inspected the procedures and used statistical tests to validate the measures.

In the simulation, we used a `sleep()` function call to simulate workloads going through the paths in the IOTG, then measuring the wall clock time difference between the start and end of the process and considered the operation time of the tasks varying between from 1 to 4 milliseconds. However, this time is on the same magnitude as the Operational System scheduling time slice; then, the `sleep()` function may not be precise. In future experiments, we will add profiler results to get a time more precisely to sleep functions and to ensure the thread sleep calls are functioning as intended.

6.6.4 External validity

External validity focuses on the generalisation of the results outside the scope of our study [26]. This study is generalised for integration platforms that adopt the integration patterns by Hohpe and Woolf [44], the style Pipes-and-Filters and task-based model. We reported this study following an empirical guideline [88] that exact repetition would be possible, according to scientific methods. The experiment is valid to test other parameters, such as integration processes, message arrival rate, simulation duration. There is no guarantee, whatsoever, that a different initial model or a different sequence of steps would not lead to a better fit. In this sense, step-wise models are locally optimal, but may not be globally optimal.

7 Conclusion

Integration-Platform-as-a-Service (iPaaS) is a cloud service that allows the development of integration processes for applications to exchange data and functionalities. Many of the open-source integration platforms have been adopting the integration patterns documented by Hohpe and Woolf [44] and the Pipes-and-Filters architectural style [1]. In this architectural style, the pipes represent message channels, and the filters represent atomic tasks that implement a concrete integration pattern to 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 heuristics used and that there was a significant difference amongst them.

6.6.3 Internal validity

Internal validity aims to ensure that the treatment caused the outcome, mitigating effects of other uncertain or not measured factors [26]. Instrumentation and source of noise are possible threats. We experimented the same machine, which was on security mode, with minimal features and disconnected from the Internet during the executions, in order to minimise interference in the execution time of the algorithm. We built our algorithm in Java, usually, the first executions of codes were slower, and it is advisable to let the virtual machine eventually perform code optimisation [69]. Then, we, firstly, executed the algorithm once only to warm the Java virtual machine up. Additionally, the researchers accurately inspected the procedures and used statistical tests to validate the measures.

In the simulation, we used a `sleep()` function call to simulate workloads going through the paths in the IOTG, then measuring the wall clock time difference between the start and end of the process and considered the operation time of the tasks varying between from 1 to 4 milliseconds. However, this time is on the same magnitude as the Operational System scheduling time slice; then, the `sleep()` function may not be precise. In future experiments, we will add profiler results to get a time more precisely to sleep functions and to ensure the thread sleep calls are functioning as intended.

6.6.4 External validity

External validity focuses on the generalisation of the results outside the scope of our study [26]. This study is generalised for integration platforms that adopt the integration patterns by Hohpe and Woolf [44], the style Pipes-and-Filters and task-based model. We reported this study following an empirical guideline [88] that exact repetition would be possible, according to scientific methods. The experiment is valid to test other parameters, such as integration processes, message arrival rate, simulation duration. There is no guarantee, whatsoever, that a different initial model or a different sequence of steps would not lead to a better fit. In this sense, step-wise models are locally optimal, but may not be globally optimal.

7 Conclusion

Integration-Platform-as-a-Service (iPaaS) is a cloud service that allows the development of integration processes for applications to exchange data and functionalities. Many of the open-source integration platforms have been adopting the integration patterns documented by Hohpe and Woolf [44] and the Pipes-and-Filters architectural style [1]. In this architectural style, the pipes represent message channels, and the filters represent atomic tasks that implement a concrete integration pattern to

As future work, we intend to experiment with an extensive data set in order to evaluate the generalisation of the results; to improve the time measures of the simulations; to investigate an architectural strategy similar to that used at the hardware level to scale tasks to speed up the runtime of integration processes.

References

1. Alexander C, Ishikawa S, Silvertein M (1977) A pattern language: towns, buildings, construction. Oxford University Press, Oxford
2. Alipouri Y, Sebt MH, Ardeshir A, Chan WT (2018) Solving the fs-rcpsp with hyper-heuristics: a policy-driven approach. J Oper Res Soc 70:403–419
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. Angra S, Chanda A, Chawla V (2018) Comparison and evaluation of job selection dispatching rules for integrated scheduling of multi-load automatic guided vehicles serving in variable sized flexible manufacturing system layouts: a simulation study. Manag Sci Lett 8(4):187–200
5. Anwar N, Deng H (2018) Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. Future Internet 10(5):1–23
6. Baker KR, Trietsch D (2018) Principles of sequencing and scheduling. Wiley, London
7. Ballestín F, Pérez Á, Quintanilla S (2018) Scheduling and rescheduling elective patients in operating rooms to minimise the percentage of tardy patients. J Sched 22(1):1–12
8. Basili VR, Rombach D, Kitchenham KSB, Selby D, Pfahl RW (2007) Empirical software engineering issues. Springer, Berlin, Heidelberg
9. Belusso CLM, Sawicki S, Roos-Frantz F, Frantz RZ (2016) A study of petri nets, Markov Chains and queueing theory as mathematical modelling languages aiming at the simulation of enterprise application integration solutions: a first step. Procedia Comput Sci 100:229–236
10. Blazewicz J, Ecker KH, Pesch E, Schmidt G, Sterna M, Weglarz J (2019) Handbook on scheduling: from theory to practice. Springer, Berlin
11. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K (2005) Task scheduling strategies for workflow-based applications in grids. IEEE Int Sympos Clust Comput Grid (CCGrid) 2:759–767
12. Boehm M, Habich D, Preissler S, Lehner W, Wloka U (2011) Cost-based vectorization of instance-based integration processes. Inf Syst 36(1):3–29
13. Brahmi Z, Gharbi C (2014) Temporal reconfiguration-based orchestration engine in the cloud computing. In: International Conference on Business Information Systems (ICBIS), pp 73–85
14. Buddala R, Mahapatra SS (2019) Two-stage teaching-learning-based optimization method for flexible job-shop scheduling under machine breakdown. Int J Adv Manuf Technol 100(5):1419–1432
15. 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. Futur Gener Comput Syst 25(6):599–616
16. Canon LC, Jeannot E (2007) A comparison of robustness metrics for scheduling DAGs on heterogeneous systems. In: International Conference on Cluster Computing (IEEE Cluster), pp 558–567
17. Cats O, Gkioulou Z (2017) Modeling the impacts of public transport reliability and travel information on passengers waiting-time uncertainty. EURO J Transp Logist 6(3):247–270

43. Hohpe G (2005) Your coffee shop doesn't use two-phase commit [asynchronous messaging architecture]. IEEE Softw 22(2):64–66
44. Hohpe G, Woolf B (2004) Enterprise integration patterns: designing, building, and deploying messaging solutions. Addison-Wesley Professional, London
45. Hu Y, Zhu F, Zhang L, Lui Y, Wang Z (2019) Scheduling of manufacturers based on chaos optimization algorithm in cloud manufacturing. Robot Comput Integr Manuf 58:13–20
46. Huang J, Süer GA (2015) A dispatching rule-based genetic algorithm for multi-objective job shop scheduling using fuzzy satisfaction levels. Comput Ind Eng 86:29–42
47. Jedlitschka A, Pfahl D (2005) Reporting guidelines for controlled experiments in software engineering. In: International Symposium on Empirical Software Engineering (ESEM), pp 95–104
48. Kaytez F, Taplamacioglu MC, Cam E, Hardalac F (2015) Forecasting electricity consumption: a comparison of regression analysis, neural networks and least squares support vector machines. Int J Electr Power Energy Syst 67:431–438
49. Konsek H (2013) Instant Apache ServiceMix How-to. Packt Publishing, London
50. Kozik A, Rudek R (2018) An approximate/exact objective based search technique for solving general scheduling problems. Appl Soft Comput 62:347–358
51. Kraisig AR, Welter FC, Haugg IG, Cargnin R, Roos-Frantz F, Sawicki S, Frantz RZ (2016) Mathematical model for simulating an application integration solution in the academic context of unijuí university. Procedia Comput Sci 100:407–413
52. Kuhn R, Hanafee B, Allen J (2017) Reactive design patterns. Manning Publications Company, London
53. Lee J, Wu H, Ravichandran M, Clark N (2010) Thread tailor: dynamically weaving threads together for efficient, adaptive parallel applications. ACM SIGARCH Comput Archit News 38(3):270–279
54. Leonard NE, Levine WS (1995) Using MATLAB to analyze and design control systems. Benjamin-Cummings Publishing Company, New York
55. Lin X, Janak SL, Floudas CA (2004) A new robust optimization approach for scheduling under uncertainty: I. bounded uncertainty. Comput Chem Eng 28(6–7):1069–1085
56. Linthicum DS (2017) Cloud computing changes data integration forever: What's needed right now. IEEE Cloud Comput 4(3):50–53
57. Liu D, Xu Y, Weii Q, Liu X (2018) Residential energy scheduling for variable weather solar energy based on adaptive dynamic programming. J Autom Sin 5(1):36–46
58. Ma L, Agrawal K, Chamberlain RD (2014) A memory access model for highly-threaded many-core architectures. Futur Gener Comput Syst 30:202–215
59. Manasrah AM, Ali HB (2018) Workflow scheduling using hybrid GA-PSO algorithm in cloud computing. Wirel Commun Mob Comput 2018:1–16
60. Manikas K (2016) Revisiting software ecosystems research: a longitudinal literature study. J Syst Softw 117:84–103
61. Manzini M, Erik D, Urgo M, et al (2018) A proactive-reactive approach to schedule an automotive assembly line. In: International Conference on Project Management and Scheduling (PMS), pp 152–155
62. McCarthy RV, McCarthy MM, Ceccucci W, Halawi L (2019) Predictive models using regression. Springer, Berlin, pp 89–121
63. Nouiri M, Bekrar A, Jemai A, Trentesaux D, Ammari AC, Niar S (2017) Two stage particle swarm optimization to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns. Comput Ind Eng 112:595–606
64. Nowatzki T, Ardalani N, Sankaralingam K, Weng J (2018) Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign. In: International conference on parallel architectures and compilation techniques (PACT), pp 1–15
65. Parunak HVD (1991) Characterizing the manufacturing scheduling problem. J Manuf Syst 10(3):241–259
66. Pezzini M, Natis YV, Malinverno P, Iijima K, Thompson J, Thoo E, Guttridge K (2015) Magic quadrant for enterprise integration platform as a service. Gartner, Stamford, pp 1–35
67. Pietri I, Chronis Y, Ioannidis Y (2019) Fairness in dataflow scheduling in the cloud. Inf Syst 83:118–125
68. Pinedo ML (2016) Scheduling theory, algorithms, and systems. Springer, Berlin
69. Pinto G, Castor F, Liu YD (2014) Understanding energy behaviors of thread management constructs. ACM SIGPLAN Not 49:345–360
70. Prasad AVK (2017) Exploring the convergence of big data and the internet of things. IGI Global, New York

43. Hohpe G (2005) Your coffee shop doesn't use two-phase commit [asynchronous messaging architecture]. IEEE Softw 22(2):64–66
44. Hohpe G, Woolf B (2004) Enterprise integration patterns: designing, building, and deploying messaging solutions. Addison-Wesley Professional, London
45. Hu Y, Zhu F, Zhang L, Lui Y, Wang Z (2019) Scheduling of manufacturers based on chaos optimization algorithm in cloud manufacturing. Robot Comput Integr Manuf 58:13–20
46. Huang J, Süer GA (2015) A dispatching rule-based genetic algorithm for multi-objective job shop scheduling using fuzzy satisfaction levels. Comput Ind Eng 86:29–42
47. Jedlitschka A, Pfahl D (2005) Reporting guidelines for controlled experiments in software engineering. In: International Symposium on Empirical Software Engineering (ESEM), pp 95–104
48. Kaytez F, Taplamacioglu MC, Cam E, Hardalac F (2015) Forecasting electricity consumption: a comparison of regression analysis, neural networks and least squares support vector machines. Int J Electr Power Energy Syst 67:431–438
49. Konsek H (2013) Instant Apache ServiceMix How-to. Packt Publishing, London
50. Kozik A, Rudek R (2018) An approximate/exact objective based search technique for solving general scheduling problems. Appl Soft Comput 62:347–358
51. Kraisig AR, Welter FC, Haugg IG, Cargnin R, Roos-Frantz F, Sawicki S, Frantz RZ (2016) Mathematical model for simulating an application integration solution in the academic context of unijuí university. Procedia Comput Sci 100:407–413
52. Kuhn R, Hanafee B, Allen J (2017) Reactive design patterns. Manning Publications Company, London
53. Lee J, Wu H, Ravichandran M, Clark N (2010) Thread tailor: dynamically weaving threads together for efficient, adaptive parallel applications. ACM SIGARCH Comput Archit News 38(3):270–279
54. Leonard NE, Levine WS (1995) Using MATLAB to analyze and design control systems. Benjamin-Cummings Publishing Company, New York
55. Lin X, Janak SL, Floudas CA (2004) A new robust optimization approach for scheduling under uncertainty: I. bounded uncertainty. Comput Chem Eng 28(6–7):1069–1085
56. Linthicum DS (2017) Cloud computing changes data integration forever: What's needed right now. IEEE Cloud Comput 4(3):50–53
57. Liu D, Xu Y, Weii Q, Liu X (2018) Residential energy scheduling for variable weather solar energy based on adaptive dynamic programming. J Autom Sin 5(1):36–46
58. Ma L, Agrawal K, Chamberlain RD (2014) A memory access model for highly-threaded many-core architectures. Futur Gener Comput Syst 30:202–215
59. Manasrah AM, Ali HB (2018) Workflow scheduling using hybrid GA-PSO algorithm in cloud computing. Wirel Commun Mob Comput 2018:1–16
60. Manikas K (2016) Revisiting software ecosystems research: a longitudinal literature study. J Syst Softw 117:84–103
61. Manzini M, Erik D, Urgo M, et al (2018) A proactive-reactive approach to schedule an automotive assembly line. In: International Conference on Project Management and Scheduling (PMS), pp 152–155
62. McCarthy RV, McCarthy MM, Ceccucci W, Halawi L (2019) Predictive models using regression. Springer, Berlin, pp 89–121
63. Nouiri M, Bekrar A, Jemai A, Trentesaux D, Ammari AC, Niar S (2017) Two stage particle swarm optimization to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns. Comput Ind Eng 112:595–606
64. Nowatzki T, Ardalani N, Sankaralingam K, Weng J (2018) Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign. In: International conference on parallel architectures and compilation techniques (PACT), pp 1–15
65. Parunak HVD (1991) Characterizing the manufacturing scheduling problem. J Manuf Syst 10(3):241–259
66. Pezzini M, Natis YV, Malinverno P, Iijima K, Thompson J, Thoo E, Guttridge K (2015) Magic quadrant for enterprise integration platform as a service. Gartner, Stamford, pp 1–35
67. Pietri I, Chronis Y, Ioannidis Y (2019) Fairness in dataflow scheduling in the cloud. Inf Syst 83:118–125
68. Pinedo ML (2016) Scheduling theory, algorithms, and systems. Springer, Berlin
69. Pinto G, Castor F, Liu YD (2014) Understanding energy behaviors of thread management constructs. ACM SIGPLAN Not 49:345–360
70. Prasad AVK (2017) Exploring the convergence of big data and the internet of things. IGI Global, New York

71. Rameshkumar K, Suresh RK, Mohanasundaram KM (2005) Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. In: International Conference on Advances in Natural Computation (ICNC), pp 572–581
72. Rimal BP, Maier M (2017) Workflow scheduling in multi-tenant cloud computing environments. IEEE Trans Parallel Distrib Syst 28(1):290–304
73. Ritter D, May N, Sachs K, Rinderle-Ma S (2016) Benchmarking integration pattern implementations. In: International Conference on Distributed and Event-Based Systems (DEBS), pp 125–136
74. Ritter D, May N, Rinderle-Ma S (2017) Patterns for emerging application integration scenarios: a survey. Inf Syst 67:36–57
75. Ritter D, Forsberg FN, Rinderle-Ma S (2018) Optimization strategies for integration pattern compositions. In: International Conference on Distributed and Event-Based Systems (DEBS), pp 88–99
76. Rodriguez MA, Buyya R (2018) Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Futur Gener Comput Syst 79:739–750
77. Roos-Frantz F, Binelo M, Frantz RZ, Sawicki S, Basto-Fernandes V (2015) Using petri nets to enable the simulation of application integration solutions conceptual models. In: Conference on Enterprise Information Systems (ICEIS), pp 87–96
78. Russell J, Cohn R (2012) Jitterbit integration server. Book on Demand
79. 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
80. Sargent RG (2013) Verification and validation of simulation models. J Simul 7(1):12–24
81. Sharma S (2017) Ovum decision matrix highlights the growing importance of ipaas and api platforms in hybrid integration. Technical report, Ovum Consulting
82. Shoukry A, Khader J, Gani S (2019) Improving business process and functionality using IoT based E3-value business model. Electron Mark 1:1–10
83. 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
84. Surhone LM, Timpledon MT, Marseken SF (2010) Petals ESB. Betascript Publishing, New York
85. Varela MLR, Ribeiro RA (2014) Distributed manufacturing scheduling based on a dynamic multi-criteria decision model. In: Recent developments and new directions in soft computing. Springer, pp 81–93
86. 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
87. Witt C, Bux M, Gusew W, Leser U (2019) Predictive performance modeling for distributed batch processing using black box monitoring and machine learning. Inf Syst 82:33–52
88. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer, Berlin
89. 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. Futur Gener Comput Syst 97:36–378
90. Yahouni Z, Mebarki N, Sari Z (2018) Tardiness minimisation heuristic for job shop scheduling under uncertainties using group sequences. Int J Intell Eng Inf 6(1–2):4–22
91. Yavuz M, Ergin H (2018) Advanced constraint propagation for the combined car sequencing and level scheduling problem. Comput Oper Res 100:128–139
92. Younis MF, Marlowe TJ, Stoyen AD, Tsai G (1999) Statically safe speculative execution for real-time systems. IEEE Trans Softw Eng 25(5):701–721
93. Zaourar L, Aba MA, Briand D, Philippe JM (2018) Task management on fully heterogeneous micro-server system: modeling and resolution strategies. Concurr Comput Pract Exp 30(23):1–16
94. Zheng W, Tang L, Sakellariou R (2015) A priority-based scheduling heuristic to maximize parallelism of ready tasks for dag applications. In: International symposium on cluster. Cloud and grid computing (CCGrid). IEEE, pp 596–605
95. Zhou Q, Li G, Li J, Shu L, Zhang C, Yang F (2017) Dynamic priority scheduling of periodic queries in on-demand data dissemination systems. Inf Syst 67:58–70