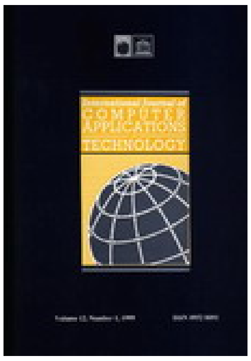


International Journal of Computer Applications in Technology

ISSN online: 1741-5047 - ISSN print: 0952-8091
https://www.inderscience.com/ijcat

Experimental study for makespan reduction in enterprise application integration processes using bio-inspired algorithms

Maira S. Brigo, Fernando Parahyba, Rafael Z. Frantz, Sandro Sawicki, Fabricia Roos-Frantz

DOI: 10.1504/IJCAT.2023.10056445

Article History:

Received:	21 December 2021
Last revised:	29 March 2022
Accepted:	02 May 2022
Published online:	23 May 2023

Experimental study for makespan reduction in enterprise application integration processes using bio-inspired algorithms

Maira S. Brigo*, Fernando Parahyba, Rafael Z. Frantz, Sandro Sawicki and Fabricia Roos-Frantz

Unijuí University,
Rua do Comércio, Ijuí, Rio Grande do Sul, Brazil
Email: maira.brigo@sou.unijui.edu.br
Email: fernando.parahyba@sou.unijui.edu.br
Email: rzfrantz@unijui.edu.br
Email: sawicki@unijui.edu.br
Email: frfrantz@unijui.edu.br

Abstract: Enterprise Application Integration area seeks to support the companies' business processes by enabling data and functionality of the applications to become reusable. Integration platforms are tools that develop and execute integration processes. This execution is done by a key component of the platforms called run-time system; that said, the performance from integration processes heavily depends on the efficiency of the run-time system. The task-based execution model implemented by the run-time system can use a strategy based on local pools to store computational threads associated with each task that make up the workflow of the integration process, in order to execute them. The challenge in this strategy is to evenly distribute the threads in each pool, minimising the makespan. We propose an experimental study, which uses two meta-heuristics to find the best distribution with the optimal number of threads. We compared both Particle Swarm Optimisation and Cat Swarm Optimisation, with the latter showing better results.

Keywords: makespan; task-based; run-time system; optimisation; integration platforms; integration process; meta-heuristics; particle swarm optimisation; cat swarm optimisation; threads.

Reference to this paper should be made as follows: Brigo, M.S., Parahyba, F., Frantz, R.Z., Sawicki, S. and Roos-Frantz, F. (2023) 'Experimental study for makespan reduction in enterprise application integration processes using bio-inspired algorithms', *Int. J. Computer Applications in Technology*, Vol. 71, No. 1, pp.18–32.

Biographical notes: Maira S. Brigo is a PhD student in the Post-Graduation Program in Mathematical and Computational Modelling at Unijuí University. Her current research interest includes systems integration and analysis, optimisation and mathematical and computational models.

Fernando Parahyba is a PhD student in the Post-Graduation Program in Mathematical and Computational Modelling at Unijuí University. His current research interest includes systems integration and analysis, model verification and smart contracts in the context of integration processes modelling and implementation to ensure agreements and detect contract violations.

Rafael Z. Frantz is an Associate Professor at Unijuí University and Leads the Applied Computing Research Group. He received the PhD degree in Software Engineering from the University of Seville, Spain. His current research interests focus on integration of enterprise applications, optimisation, decentralised technologies including blockchain, smart contracts and data integration.

Sandro Sawicki is an Associate Professor at Unijuí University. He received his PhD degree in Computer Science from Federal University of Rio Grande do Sul, Brazil. His current research interests include mathematical optimisation, graph theory, hypergraph partitioning and search-based software engineering.

Fabricia Roos-Frantz is an Associate Professor at Unijuí University. She received the PhD degree in Software Engineering from the University of Seville, Spain. Her current research interests include data integration and analysis, optimisation, petri nets and learning analytics.

1 Introduction

Over time, companies have either developed software applications or bought them from third parties to support their business processes, which are in constant transformation. A company's set of applications is called software ecosystem (Manikas, 2016). A large part of these applications were developed in different programming languages, to be executed on different operating systems. Since they have different data models, the software ecosystem is heterogeneous. Often, such applications are not projected for exchanging data and sharing functionality. Therefore, making it possible for applications to work together and supporting the growing volume of data currently involved in business processes is a recurrent challenge for companies' Information Technology (IT) sectors.

The Enterprise Application Integration (EAI) area provides methodologies, techniques and tools to carry out the integration of software ecosystem applications, through the design, construction and execution of integration processes (Hohpe and Woolf, 2003; Frantz et al., 2021). An integration process is implemented as a piece of software so that it can be executed to integrate different applications involved in a business process. On the other hand, integration platforms are tools that offer resources to model, implement and execute integration processes (Freire et al., 2019a). Generally, an integration platform has: (i) a Domain-Specific Language (DSL), proposed to conceptually model an integration process; (ii) a development toolkit, which transforms the model into executable code; (iii) a run-time system, responsible for executing processes and (iv) monitoring tools, which analyse the integration process performance.

Several open-source message-based platforms use the pipes-and-filters architectural style (Alexander, 1977). On these platforms, pipes are communication channels connecting two tasks and allowing them to asynchronously execute messages that flow within the integration process. Messages are structures that encapsulate data flowing through the process. Filters are atomic tasks that generally perform one of the integration patterns documented by Hohpe and Woolf (2003), and perform concrete actions on the messages.

Figure 1 illustrates a typical integration process. Integration processes are usually composed of an entry and an exit port, and a set of organised tasks that make up the workflow. Through ports, the process communicates with applications of the software ecosystem. In an integration process, tasks are linked in such a way that a message can only be executed by a task if it has already been executed by the predecessor task. A message is considered processed when it has been executed by all tasks that make up the workflow. A recurrent challenge in integration platforms is how to improve the performance of run-time systems and to optimise the computational resources used. Owing to the increasing generation of data in the companies software ecosystem – including data from sensors, mobile devices and digital media, run-time systems from integration platforms need to be adapted to efficiently handle the growing volume of data (Freire et al., 2019b).

There are two theoretical models in the literature that can be followed for the implementation of run-time systems: the Process-Based (Alkhanak et al., 2016; Boehm et al., 2011) model and the Task-Based (Blythe et al., 2005; Frantz et al., 2012) model. In both cases, the computational resources used are threads. In the Process-Based model, threads are allocated at process level and when a message arrives at an integration process input port, one thread is designated to process the message along all process tasks. Concrete occurrence tasks are named workunits. In the process-based, a workunit of the whole process is created and a thread executes in sequence all algorithms of tasks that make up the flow in a single workunit of the process. Note that in this model, there is no need for slots, as the set of tasks is executed by the same thread on the message. In the Task-Based model, threads are allocated at task level. In this model, threads execute the task algorithm on the message and are free to execute any other task in the process. In this very case, workunits are created for each task. This paper addresses the Task-Based execution model, once it is better suited to scenarios with large volumes of data, which eventually demand higher performance (Boehm et al., 2011). Also according to Blythe et al. (2005), the desynchronisation of tasks avoids a possible waiting time and allows a better use of computational resources, thus providing a more efficient execution.

Figure 1 Example of a typical integration process

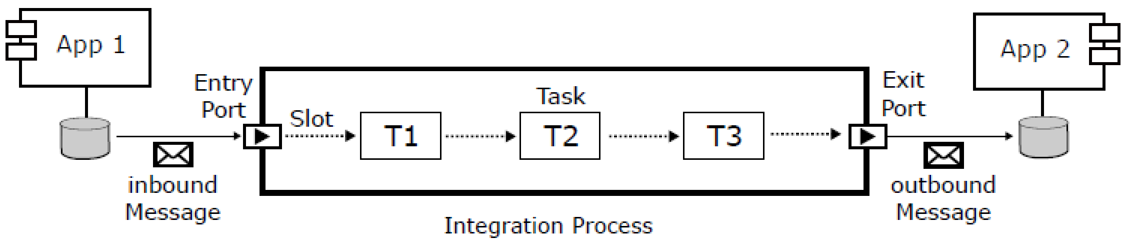
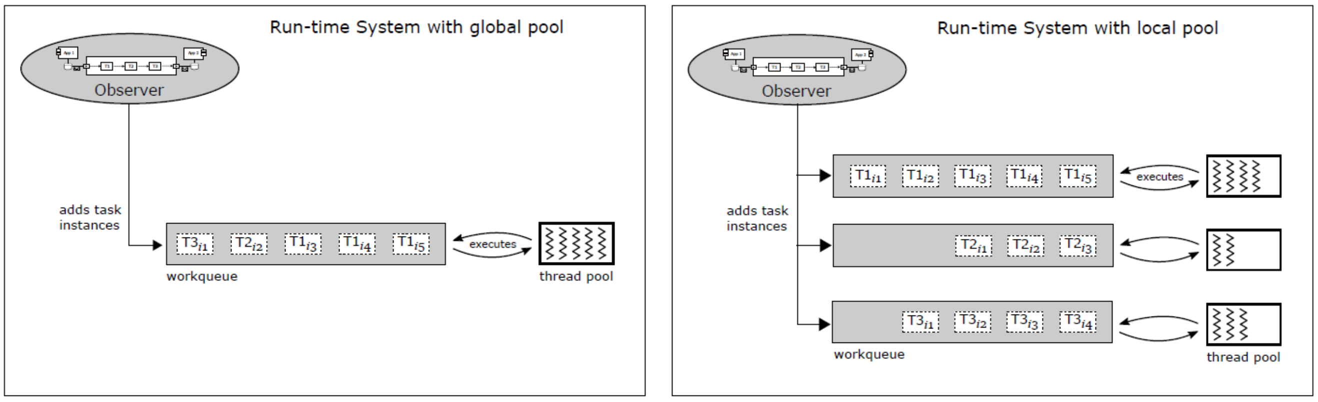


Figure 2 Elements involved in the task-based run-time system with the global and local pool strategies. '*T*' represents a type of task and 'i_n' represents the instant in time the instance was added to the workqueue


Run-time System with global pool
Observer
adds task instances
$T3_{i1}$ $T2_{i2}$ $T1_{i3}$ $T1_{i4}$ $T1_{i5}$
executes
workqueue
thread pool
Run-time System with local pool
Observer
adds task instances
$T1_{i1}$ $T1_{i2}$ $T1_{i3}$ $T1_{i4}$ $T1_{i5}$
executes
$T2_{i1}$ $T2_{i2}$ $T2_{i3}$
$T3_{i1}$ $T3_{i2}$ $T3_{i3}$ $T3_{i4}$
workqueue
thread pool

Commonly, there are two strategies to allocate threads in the Task-Based model: global and local pools strategy, as shown in Figure 2. The global pool strategy consists of a single pool to store all threads available to execute messages in the integration process; a single thread from the global pool is shared by all internal tasks of the process. When there are messages in all task entries, this task is able to be executed. Then, a workunit of it is created and stored in a workunit queue, called workqueue. The workqueue receives all workunits for all tasks in the integration process. In this strategy, when the process needs to handle a large volume of data, that is, the message arrival rate is very high, it is likely that threads shall concentrate to prioritise the execution of tasks at the beginning of the workflow, once messages entering the process will generate workunits from the first workflow tasks in the workqueue, harming tasks arranged at the end of the workflow and consequently, the execution of the integration process.

The local pool strategy minimises the problem of accumulation of threads in the initial tasks of the process, as it uses several pools of threads – with one pool per task – making each task have its own set of threads and consequently, a workqueue for each task. The Observer has the role of observing the integration process so that whenever there are messages in all entries of a task, it creates a workunit of the task by adding that workunit in the corresponding workqueue. Although this strategy minimises the problem of concentration of threads at the beginning of the workflow (caused by using a single global pool), the challenge is to correctly distribute threads in each pool, so that this distribution can improve performance. A widely used metric to calculate the performance of a run-time system is the average processing time of a message in the integration process, known as makespan (Chirkin et al., 2017). A low makespan is indicative of more messages processed per unit of time, which means better performance. While a high makespan indicates fewer messages processed per unit of time and, consequently, poorer performance (Parahyba et al., 2021).

Meta-heuristics can be used to find the best thread distribution and the optimal number for each pool. A meta-heuristic is a method used to solve optimisation problems in a generic way (AbdelAziz et al., 2020). Through it, an initial population is generated and undergoes transformations of different principles to reach a result that maximises or minimises the objective function. In this paper, we propose a comparative experimental study to analyse the efficiency of Particle Swarm Optimisation (PSO) (Slowik, 2020; Freire et al., 2020) and Cat Swarm Optimisation (CSO) (Slowik, 2020; Lima et al., 2019) meta-heuristics to find the best distributions with the optimal numbers of threads for local pools to minimise the makespan in an integration process. We chose to use PSO at first, as it is an easy technique to implement, widely used in scheduling problems in Software Engineering, and one of the most popular among algorithms based on a technique (Kaleche et al., 2020). Despite being a more recent technique, CSO is also easy to implement and widely used in this area. Singh et al. (2017) implemented CSO and compared it through different numbers of independent tasks, with the Genetic Algorithm (GA) and with the standard PSO; the results indicated that CSO is 50% better than PSO, and that it is 27% better than GA when it comes to run-time. Thus, in this paper we design and execute an experiment by using a single protocol, turning heuristics into input variables with the adopted parameters. Algorithms from two of these meta-heuristics are executed in the same controlled environment, aiming at finding the best thread distribution to generate the lowest makespan in the integration process. The results were analysed through comparison to identify which meta-heuristic is the most adequate to solve the thread distribution problem. At the end of the experiments, we found out that CSO delivered a superior performance and a better average result than PSO.

The remainingder of this paper is organised this way: Section 2 introduces the PSO and CSO meta-heuristics used in the experiments, and adds important information and characteristics of them; Section 3 presents related works; Section 4 describes the formulation of the problem; Section 5 presents the experiments performed, and discusses all stages of the experimentation protocol; Section 6 presents the analysis from results and lists possible future works.

2 Background

In this section we discuss, in a broad sense, what the meaning of heuristic is, and more specifically what the meaning of meta-heuristics is. We explain the algorithms based on nature and the organisms that live in it; we also mention the classes into which they are divided. The focus is on swarm-based algorithms, and then there is a debate on the two algorithms which belong to this class, and were used in the experiments.

According to Game et al. (2020), a heuristic is an exploratory technique that aims to improve performance and solve problems. A heuristic algorithm uses information available to advance towards the solution. This is an approximate method, that is, it provides good solutions to a given problem in a reasonable amount of time. A meta-heuristic is a heuristic method that finds the best solutions to the problem. According to Kaleche et al. (2020), heuristics are specific to a problem, while meta-heuristics are independent from a problem. The bio-inspired optimisation algorithm was created based on the characteristics and behaviour of living organisms in nature. This natural computation has been divided into three main classes: computation inspired by nature; simulation and emulation of nature; and computation with natural materials. Therefore, nature has served as a source of inspiration, and these nature-based algorithms are meta-heuristic algorithms divided into three parts: evolutionary algorithms, physics-based algorithms and swarm intelligence algorithms. The choice of working with meta-heuristic swarm intelligence algorithms is explained by the fact that, according to Game et al. (2020), swarm intelligence algorithms are advantageous because they use few parameters, they can memorise the best previous results and are ease of implementation.

The PSO meta-heuristic is motivated by the simulation of social behaviour (Shi and Eberhart, 1999), and we can infer that it is a collective iterative method that emphasises cooperation (Clerc, 2010). It was introduced by Eberhart and Kennedy (1995), who were inspired by biological organisms, more precisely in the ability of some beings when it comes to group social work behaviour, and how they find sources of food (Bratton and Kennedy, 2007). It can be inferred that PSO has a cooperative behaviour, and when it is applied, there is an exchange of information among those involved in solving a problem. It is, therefore, based on a swarm of particles that move in the space they are inserted in, and the communication among them points the direction in the search they wish to follow. This technique can be applied to scheduling tasks in network environments and can also reduce the overall execution time of scheduling tasks in computational grids. It is the most used technique based on swarm intelligence for scheduling tasks in the cloud (Singh et al., 2017).

According to the creator of PSO, a swarm looks like a disorganised population of moving individuals which tend to cluster together, while each individual appears to be moving in a random direction. The swarm of particles has artificial roots and evolutionary calculus, in addition to being a simple, easy-to-implement concept, effective in several problems. An interesting feature is that each particle controls its best position in hyperspace. PSO moves around its own research space, and its current particles decide their next positions in this very space. From a population of solutions that are candidates for the best solution there is a movement, and the movement of each of these solutions – which can be called particles – is influenced by its position at the best-known location. Since the purpose is precisely to bring the entire swarm of particles close to the best solution, the position is always updated to the best position found.

The CSO meta-heuristic, implemented by Chu et al. (2006), is based on the behaviour of cats. Considering that such animals pay close attention to everything and are always alert, it can inferred that they are intelligent and deliberate beings. This technique models two main behaviours from cats: the seeking mode and the tracing mode (Chu et al., 2006). The seeking mode is used to model the behaviour of the cat in its rest period. When a cat chases its preys, it enters the tracing mode (Shojaee et al., 2012).

According to Shojaee et al. (2012), in order to use CSO in an optimisation problem you must first decide the number of cats to be used, noting that each one of them has a dimensional position, velocities for each dimension and fitness value, which represents the cat accommodation and a search or trace flag, to reveal in which mode the cat finds itself. The behaviour of cats technique was used because, from this point on, spaces for complex solutions are searched for optimal solutions. The initial population of cats is randomly divided into seeking and tracing mode. The best approximation results per cat are recorded in the memory. When all cats have their best position, the final solution is the best position of one of these cats (Lima et al., 2019).

3 Related work

Through experiments, a set of works seek to reduce makespan by applying optimisation techniques in different types of processes. Algorithms were analysed and compared in the related works studied, with some authors improving existing algorithms, and others proposing new algorithms. Table 1 presents a comparative summary of these works based on a set of properties. Column 1 identifies the works analysed. Column 2 indicates whether the work uses PSO, CSO or some variation of these meta-heuristics. Column 3 refers to the use of makespan and workflow in the analysis performed. Column 4 summarises the techniques used, in addition to those already mentioned in column 2. Finally, column 5 reproduces what technique is the best according to the authors. Our proposal - used of original algorithms - stands out for using two unmodified meta-heuristics, which minimises the makespan of a workflow. Subsequently, we present the related works one by one, with their main characteristics.

Table 1 Summary of related works

Proposal	*Meta-heuristic*	*Makespan/Workflow*	*Techniques used*	*Best technique*
Junaid et al. (2020)	ACOPS, CPSO, QMPSO, CSO	makespan	SVM, ACO, D-ACOELB	SVM
Natesha et al. (2018)	PSO, CSO	makespan	GWO, FCFS, MT, GA	GWO
Jing et al. (2018)	PSO, CSO	makespan / workflow	D-ITGO, GA, ABC, Cuckoo, BA	D-ITGO
Abdullahi et al. (2017)	PSO	makespan / workflow	CSOS, SOS	CSOS
Maurya and Tripathi (2018)	PSO, CSO	makespan / workflow	IC-PCP, SCS	CSO
Gupta et al. (2018)	PSO, CSO, MCSO	makespan / workflow	–	MCSO
Xu et al. (2017)	PSO	makespan	ICSO, CSO (Chicken)	ICSO
Semlali et al. (2018)	–	makespan	CSOSA, CSO (Chicken), AS	CSOSA
Bousselmi et al. (2020)	BiO-CSO, BiO-PSO	makespan / workflow	–	BiO-CSO
Gupta et al. (2019)	PSO, CSO	makespan / workflow	Jaya, GA, ACO, Honey bee	Jaya
Gabi et al. (2017)	OTB-CSO, PSO-LDIW, HPSO-AS	makespan	Min-Max	OTB-CSO
Du et al. (2019)	CSO-FA	makespan	–	CSO-FA
Gabi et al. (2019)	CSO-LDIW	makespan	–	CSO-LDIW
Irman et al. (2019)	PSO	makespan	Proposed algorithm, CDS	Proposed algorithm
Zarrouk and Jemai (2018)	PSO-JMS, PSO-OMS	makespan	–	PSO-OMS
Freire et al. (2020)	PSO-based	makespan / workflow	–	–
Lima et al. (2019)	CSO-based	makespan / workflow	–	–
Our proposal	PSO e CSO	makespan / workflow	Original algorithms	CSO

Junaid et al. (2020) seek to maintain accuracy in load balancing in cloud computing, with large volumes of data through the use of meta-heuristics. They propose a new hybrid model that classifies an amount of files present in the cloud through file type formatting. The classification is carried out with Support Vector Machine (SVM). This new model is further fed into a meta-heuristics algorithm called Ant Colony Optimisation (ACO). In addition, a comparative analysis was performed with recent meta-heuristics such as Ant Colony Optimisation-Particle Swarm Optimisation (ACOPS), Chaotic Particle Swarm Optimisation (CPSO), Q-learning Modified Particle Swarm Optimisation (QMPSO), Cat Swarm Optimisation (CSO) and D-ACOELB. The proposed algorithm surpasses them and offers good performance with scalability and robustness.

Natesha et al. (2018) proposed to manage the cloud datacentre heavy load by using a multi-objective technique for task scheduling they call Gray Wolf Optimisation (GWO). The main objective is to achieve optimal utilisation of cloud resources, as to reduce the power consumption of the datacentre and the scheduler's total makespan to the provided to-do list, while providing the services requested by the users. The comparison is performed with First-Come-First-Serve (FCFS) and Modified Throttle (MT) non-meta-heuristic algorithms, and meta-heuristic algorithms (Genetic Algorithm (GA), PSO and CSO). Experimental results demonstrate that the proposed GWO-based scheduler outperforms all algorithms considered for performance evaluation in terms of makespan for the task list, resource utilisation and energy consumption.

Also to solve the task scheduling problem in a cloud environment, Jing et al. (2018) proposed an optimisation of the Invasive Tumor Growth Optimisation (ITGO) meta-heuristic, which was called Discrete Invasive Tumor Growth Optimisation (D-ITGO). The main strategy is to make a match between cloud tasks and virtual machines, then map that type of match to the coordinates of a tumour cell, to finally solve the problem by improving some search strategies from the original ITGO. In the experiment, we used the CloudSim toolkit for testing, and the non-parametric test (Wilcoxon Test) to assess effectiveness. Experimental results and analysis showed that the new algorithm has better results than GA, PSO, Artificial Colony Algorithm (ABC), Cuckoo Search Algorithm (Cuckoo), Bat Search Algorithm (BA) and CSO.

Abdullahi et al. (2017) focused on the task scheduling problem by using a new algorithm to minimise makespan and cost they call Chaotic Symbiotic Organisms Search (CSOS). The main idea is to avoid premature convergence of the Symbiotic Organisms Search (SOS) algorithm in the early stages of the optimisation process by implementing a chaotic map, which expands the search space and provides diversity. The performance of the proposed CSOS algorithm is evaluated by extensive simulation through the CloudSim toolkit simulation framework, and by comparing with SOS and PSO. The simulation results reveal a significant performance improvement of the proposed CSOS in reducing costs and makespan in scheduling tasks.

Still on the task scheduling issue in cloud computing environments, Maurya and Tripathi (2018) reported this issue for Bag-of-Task (BoT) applications. Applications can have numerous tasks, which can increase execution costs if the scheduling is improperly done. This problem was solved using scheduling algorithms, and the authors tested two heuristic algorithms and two meta-heuristic algorithms. The heuristic algorithms were IaaS Cloud Partial Critical Path (IC-PCP) and Scaling Consolidation Scheduling (SCS) and the meta-heuristic algorithms were PSO and CSO. All the mentioned algorithms tried to minimise the manufacturing cost (already mentioned by the authors as a problem) and yet, to meet the observed time restrictions. Three BoT categories were used, so that it was possible to compare algorithms by their performance. Finally, the result of the performed experiments was that the CSO algorithm has a better performance than the others.

Gupta et al. (2018) reported that task scheduling is a very important aspect when it comes to cloud computing. Thus, their objective was to minimise cost and execution time to achieve an optimised cost mapping based on an algorithm. For doing so, they learned that several algorithms could be executed to schedule tasks in cloud computing, based on intelligent swarms. The authors chose PSO and CSO. A merged CSO (MCSO) was proposed, where combinations of merits from both CSO and PSO were presented with the aim of achieving better results. The MCSO algorithm was implemented in the CloudSim simulator and the experiment obtained good results, concluding that the MCSO algorithm has a better performance compared to the PSO and CSO algorithms in terms of execution time and convergence.

To solve the Flexible Job Scheduling Problem (FJSP) more effectively, Xu et al. (2017) proposed an algorithm called Improved Chicken Swarm Optimisation (ICSO) to minimise the makespan of the machine. The algorithm combined the advantages of the simulated annealing algorithm and the dynamic inertia cosine weighting strategy, which achieved an effective balance of global search and local exploration. It was performed a comparison between PSO and the Chicken Swarm Optimisation (CSO), and the effectiveness and superiority of the ICSO was validated. In this same context, Semlali et al. (2018) presented an adaptation of a meta-heuristic algorithm, which they called Chicken Swarm Optimisation with Simulated Annealing (CSOSA) to solve the workshop scheduling problem. The aim of the authors was to optimise production involving this new algorithm. The performance of the proposed algorithm was improved from a hybridisation of the CSO algorithm with the SA algorithm. The empirical results were obtained through the application of the new algorithm with instances of the OR Library. The computational results concluded the effectiveness of CSOSA in terms of solution quality and execution time compared to other meta-heuristics in the literature.

Scientific workflows handling large volumes of data need to be run in scalable distributed environments, such as cloud infrastructure services. Bousselmi et al. (2020) formulated a big data scientific workflow scheduling problem as a bi-objective optimisation problem that aimed to minimise makespan and cost of the workflow. The problem was solved by using a Bi-Objective Cat Swarm Optimisation (BiO-CSO) algorithm, which is an extension of the bio-inspired CSO algorithm. The performance of the proposed method was compared to that of the multi-objective PSO. Experimental results showed that BiO-CSO had a better performance than the multi-objective PSO, as it provided more and better final scaling solutions. Also in the context of workflow scheduling, Gupta et al. (2019) implement a Jaya optimisation algorithm to solve this problem in the context of cloud computing. The efficiency of the algorithm was compared with four nature-inspired algorithms, which are PSO, GA, Ant Colony Optimisation (ACO), honey bee and Cat Swarm Optimisation (CSO), keeping the same function for all of them by using CloudSim. The results were compared based on the execution cost and makespan of the algorithm on an independent set of tasks and on a set of tasks that follow a workflow schedule. It was observed that Jaya outperforms the other algorithms as it produces similar results in the shortest possible time, once it converges very quickly.

Gabi et al. (2017) proposed an algorithm called Orthogonal Taguchi Based Cat Swarm Optimisation (OTB-CSO) to minimise the total task execution time in a cloud computing datacentre. In the algorithm, the Taguchi orthogonal approach was incorporated into the CSO algorithm tracking mode for better task mapping in VMs, and minimal execution time. The algorithm was implemented in the CloudSim tool, and evaluated based on the makespan metric. Comparisons were made with Job First Minimum and Maximum (Min-Max), Particle Swarm Optimisation Linear Descender Inertia Weight (PSO-LDIW) and Hybrid Particle Swarm Optimisation Simulated Annealing (HPSO-SA). The results obtained showed that OTB-CSO is effective in optimising task scheduling and improving overall cloud computing performance. with better system utilisation.

Du et al. (2019) make an attempt to minimise both makespan and cost, and to balance the load rate of scheduling processes for manufacturing resources in clouds. The authors created a multi-objective programming model in order to achieve the aforementioned goals. Then, CSO and Firefly Algorithm (FA) were combined into a single hybrid multi-objective programming algorithm to be verified through a CloudSim simulation. As a result of this simulation, they obtained an algorithm that generates an ideal programming plan in a short time. Thus, they also learned that the application of swarm intelligence algorithms in scheduling problems is promising. Gabi et al. (2019) worked in a similar vein, as they sought to minimise production time for the efficient scheduling of tasks in a cloud datacentre. To that end, they introduced a conventional CSO based task scheduling technique, by incorporating a Linear Descending Inertia Weight (LDIW) equation in the CSO local search. This incorporation led to better convergence speed and decreased execution time. This experiment was implemented by the CloudSim simulator with five heterogeneous virtual machines, thus finding an ideal solution to the problem.

Irman et al. (2019) propose an algorithm to reduce makespan in the production time of an industry in Indonesia. The authors compared the efficiency of their algorithm in solving the problem also applying the Camppbell Dudek Smith (CDS) and PSO heuristic methods. In this work they demonstrate that when using CDS, makespan was the highest; when using PSO, it had medium performance; and with the algorithm proposed in the article, they obtained the smallest makespan. The work of Zarrouk and Jemai (2018) is intended to minimise production and total workload. To do so, the performance of two PSO variants with different particle representations were evaluated and compared: the PSO with the Job-Manchine Scheme coding (PSO-JMS), and the PSO with the Only-Manchine Scheme coding (PSO-OMS). As a result, PSO-OMS offered the best performance.

Freire et al. (2020) implemented in an ad-hoc way a PSO-based meta-heuristic for the EAI area. Likewise, Lima et al. (2019) also implemented in an ad-hoc way a meta-heuristic based on CSO. Both sought to find the best distribution with the optimal number of threads for local pools, but different from our proposal, each of them has targeted a single heuristic. Although they have addressed the problem of distributing threads, their implementation was developed by different person, which can introduce differences in the implementation depending on the skills of that person. Furthermore, the experiments carried out by both authors follow a different protocol, did not execute a large number of repetitions (which is important in this kind of study) and their algorithm implementations changed the velocity and discarded some variables and constants found in the original implementation of PSO and CSO, which can compromise the performance. We take the original algorithm implementation proposed by Eberhart and Kennedy (1995) and Chu et al. (2006), for PSO and CSO, respectively, and used the same protocol for conducting the experiments with both PSO and CSO, so that resulting data can be fairly compared.

4 Problem formulation

The workflow of an integration process can be compared to a Directed Acyclic Graph (DAG). This graph is represented by $DAG = (Z, H)$, where Z is the set of n tasks and H are the edges that represent the dependency relationship between tasks, that is, tasks depend on each other and relate to each other. We assume that a set of resources used for processing is represented by r and its storage is indicated by Dr, and all these resources are located in different P places. In a DAG there are nodes connected by edges. In our context, nodes are tasks, edges are communication channels that desynchronise tasks and workunits are concrete occurrences of tasks executed by threads. Threads are available in local pools. A path is a sequential set of tasks in a workflow, and the longest path in the integration process is called a critical path. In each node there is an associated processing time, and in the nodes there is a set of workunits that allow a parallel execution of the node.

The objective is to find the perfect number of resources r for each node, so that the processing time may be as short as possible. We consider that the resources used are Tr_n threads to be distributed in each of the local pools referring to the n tasks minimising makespan. We consider that there is a restriction that limits the total number of threads (which in the DAG is Dr and we call it $Restriction_{threads}$), and that it is necessary to have at least one thread for the task to be executed, as shown in equation (1). The processing time of a message (TTp_i) is the sum of the execution time $(T_{(n,i)})$ and the waiting time in the workqueue $(Tm_{(n,i)})$, as shown in equation (2). Thus, the $TTp_{(i)}$ of a message throughout the workflow of T_t tasks combined with the threads used (Tr_n) can be simplified by the sum of the $T_{(n,i)}$, and the sum $Tm_{(n,i)}$ between a set of tasks that are dependent on each other.

$$1 \leq Tr_n < Dr \tag{1}$$

$$TTp_{(i)} = \sum T_{(n,i)} + \sum T_m \tag{2}$$

The makespan is the average processing time of a message, which is obtained by adding the $TTp_{(i)}$ of all messages, divided by the total number of messages (k) (Bilgaiyan et al., 2014). As we want the processing time to be as little as possible, the objective function is:

Minimise makespan of message processing through the critical path of an integration process, by finding the best distribution of local pools of thread, and by using a predefined number of threads to be distributed for all local pools.

Makespan calculation can be described by equation (3):

$$Makespan = \frac{\sum_{1}^{k} \left(TTp_{(i)}\right)}{k} \tag{3}$$

The $ObjectiveFunction$ can be represented by equation (4), where $Sum_{threads}$ stands for the sum of *threads* present in the *pools* in a distribution, while $Restriction_{threads}$ represents the restriction on the number of *threads* that can be distributed. Therefore, $Sum_{threads}$ must be lower or equal to $Restriction_{threads}$ (Freire et al., 2020).

$$Objective\,Function = Minimise\,(Makespan) \tag{4}$$

Subject to restriction:

$$Sum_{threads} \leq Restriction_{threads}$$

5 Experiment

We used the objective function proposed in a meta-heuristic, which is a technique to seek optimisation. It is based on a population, in which the objective function is tested for individuals in this initial population and operations are carried out to improve the convergence of the result, in order to minimise it. Initially, it is generated a population with possible thread distributions for local pools, which is the initial population. Then, the average time for processing messages on the workflow is calculated, the makespan for each distribution. Whenever the current makespan is lower than the previous one, the minimum makespan is updated along with its distribution. At the end, the best setting is found, the one that has the lowest makespan of all repetitions.

We fitted the PSO and CSO original algorithms (Slowik, 2020) to the context of our object of study. For doing so, two functions proposed by Freire et al. (2020) were used: the function called Population Generation and the one called Makespan Calculation. The former performs the generation of populations of distributions from thread pools, while the latter performs the calculation of the corresponding makespan to each population generated, and then optimised by the algorithm. Source codes from Matlab algorithms are available for download

The experimentation protocol applied to conduct this experimental study is based on the work of Jedlitschka and Pfahl (2005), Perry et al. (2000) and Wohlin et al. (2012). It must follow some procedures, like the presentation of the research question, the presentation of the hypothesis, the experimentation environment, variables, the object of study, execution and data collection, results and discussion and threats to validity.

5.1 Research question

The Research Question (RQ) we want to answer is:

RQ: Which of the techniques (PSO/CSO) meets the best distribution of threads with the optimal number of threads in the local pools of the run-time system, also minimising makespan?

As a hypothesis, we expect that Cat Swarm Optimisation meets the best makespan, as there are authors who have implemented CSO and compared it with some techniques, including PSO and concluded that CSO was more efficient. Since our distribution problem is similar to that of the aforementioned authors, we believe that such hypothesis can be confirmed.

5.2 Experimentation environment

The experiments were carried out on a machine equipped with 32 processors Intel Xeon CPUs E5-4610 1.80 GHz CPU, 32 GB RAM and Microsoft Windows 10 64-bit operating system. Matlab (Leonard and Levine, 1995) software version R2018 was used to implement the algorithms.

5.3 Variables

The independent variables we considered in our experiments were:

- *Number of solutions*: the population of initial thread pool distribution. The value of this variable was 25 solutions.
- *Number of threads*: the total number of threads distributed in local pools. The value of this variable was divided into four: 25, 50, 75 and 100 threads.
- *Number of messages*: the workload of the integration process, where the value tested was 1,000,000 messages.
- *Algorithms*: PSO algorithm or CSO algorithm.

The dependent variable was:

- *Makespan*: the average processing time a message takes to be processed by all tasks part of the critical path of the integration process.

5.4 Object of study

The object of study adopted is the integration process called order processing, proposed by Hohpe and Woolf (2003) and showed in Figure 3. The integration process has five applications: Ordering System, Widget Inventory, Gadget Inventory, Invalid Items Log and Inventory System. The Ordering System represents the order application which delivers data from new orders to the integration process. An order data composes a message that flows through the on-boarding process. Each message with a new order is divided into separate messages by the workflow of tasks, containing only one item. The message destination is either for the Widget Inventory application or for the Gadget Inventory application, depending on what is in its content. Messages with items not belonging to any of these inventories are routed to the Invalid Items Log. The Inventory System application represents the final destination application, which responds by recording the availability of the items.

As an integration process can be represented by a DAG, the integration process used as the object of study is represented by the DAG in Figure 4. We emphasise that ports are internally implemented by the integration process as tasks that represent a specific type of adaptor for communication with the applications and that, therefore, they are also modelled as nodes in the graph.

This integration process has distinct paths for the messages. They are represented in the DAG from Figure 4. The critical path is called this way because it has more time-consuming task execution times. Table 2 shows the processing times of each task in the critical path in millisecond, taken from the work of Haugg et al. (2019).

5.5 Execution and data collection

The execution experimentation of this process was carried out in the described experimentation environment with Matlab software, by using the critical path with a processing of 1,000,000 messages. With such message arrival rate, we aimed to evaluate how the system behaves when exposed to a large volume of data, something increasingly common now. Algorithms perform 100 iterations, and at the end the best result found is selected as answer. A function was created to carry out each experiment 25 times, thus creating 25 different distributions in the local thread pools, with one pool per task. 11 pools were found in the definition of the critical path.

Figure 3 Conceptual model of the integration process

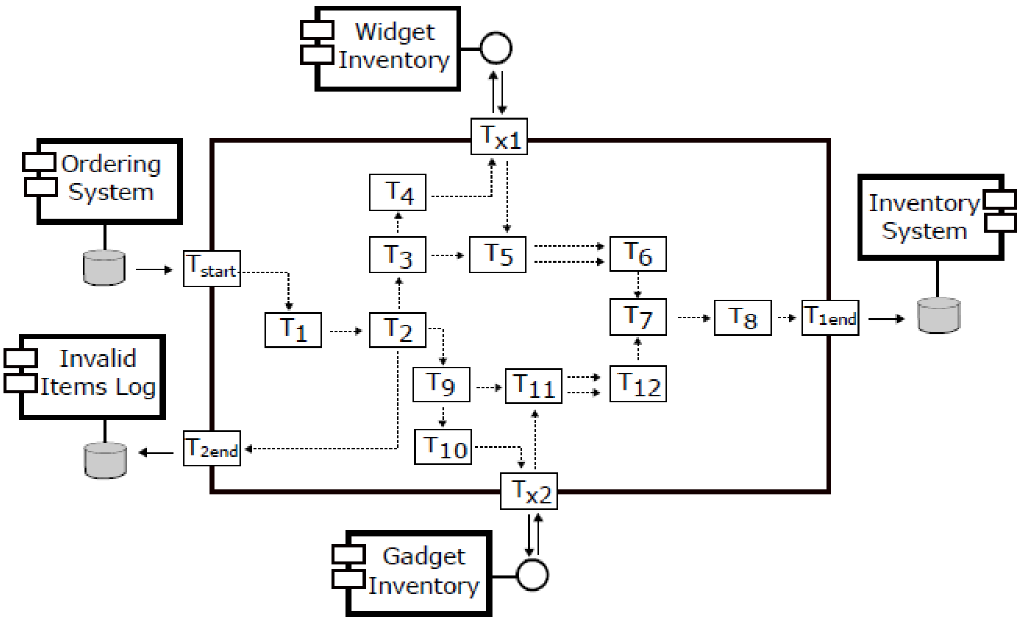


Figure 4 Integration process represented as a Directed Acyclic Graph

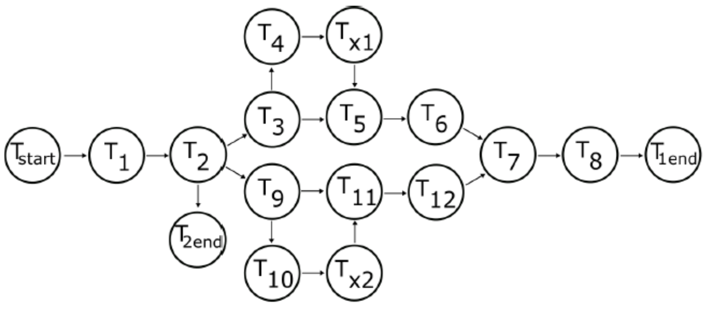


Table 2 Time of critical path tasks in milliseconds

Task	T_{start}	T_1	T_2	T_3	T_4	T_{x1}	T_5	T_6	T_7	T_8	T_{1end}
Time (ms)	2.005	2.005	3.005	3.005	2.005	2.005	4.553	2.005	4.553	2.005	2.005

Four different scenarios were used, with the combination of the following independent variables: the number of solutions, the number of threads, the number of messages, and the number of algorithms $(25 \times 4 \times 1 \times 2)$, therefore resulting in 200 executions. Four different options for the number of maximum threads to be employed were used in the execution of each of the algorithms, totalling 25, 50, 75 and 100 threads. The choice for these values came from the empirical knowledge of software engineers, and it is usually four arbitrary amounts of threads. As integration processes that use more than 100 threads are not commonly found, for comparison purposes, we divided into four values that had the same thread difference among them. The simulation for each option per number of threads of each algorithm resulted in 25 distributions, and each one of them generated a corresponding makespan. Each distribution, makespan and averages were generated through Matlab software and tabulated for analysis.

5.6 Results and discussion

The results of the experiments were obtained with the execution of the PSO algorithm and the execution of the CSO algorithm. For a better composition of the analysis and comparison purposes, four different scenarios were chosen: 25, 50, 75 and 100 threads. An algorithm was created, which performed 25 repetitions of the same experiment for each scenario chosen without the need for user intervention. Each iteration yielded a result that presented a vector of thread pool distributions and a corresponding makespan. Originally, makespan was taken in milliseconds and later converted to seconds.

We will report the results found for the PSO algorithm and for the four predicted scenarios, which is explained in Table 3. The table beholds results for the four different numbers of threads used. On the first part of the table, 25 threads; on the second one, 50; on the third part, 75 threads; and on the fourth one, 100 threads. The information reported below follows this organisation of the tables. For 25 threads, the algorithm found 758.86 s as the lowest makespan, which was shown in repetition 10. The largest makespan, 2276.53 s, was demonstrated in repetitions 3, 20 and 21. For 50 threads, the algorithm found 334.20 s as the lowest makespan with repetition settings 4, 12 and 22 and the largest makespan was 1002.53 s with 13 settings – i.e., repetitions 1, 2 and 5. For 75 threads, the algorithm found the lowest makespan 227.68 s in repetitions 2 and 8 and the largest makespan was 1002.53 s with repetition settings 5, 11, 21 and 24. Finally, with 100 threads, the algorithm found the lowest makespan 187.84 s with the distributions of repetitions 1 and 5 and the largest makespan was 1002.53 s in repetitions 14, 19 and 22.

Table 3 Thread distributions found and the makespan for PSO experiments

	25 THREADS		*50 THREADS*	
Repetition	*Threads in each pool*	*Makespan**	*Threads in each pool*	*Makespan**
1	[03; 03; 03; 02; 03; 02; 03; 01; 03; 01; 01]	1002.53	[02; 01; 03; 05; 05; 09; 02; 06; 03; 03; 01]	1002.53
2	[02; 03; 03; 02; 03; 02; 03; 02; 03; 01; 01]	1002.53	[04; 02; 04; 08; 05; 06; 13; 03; 03; 01; 01]	1002.53
3	[07; 05; 04; 01; 01; 01; 02; 01; 01; 01; 01]	2276.53	[03; 04; 06; 07; 03; 03; 08; 04; 06; 03; 03]	379.44
4	[02; 02; 03; 02; 02; 02; 03; 02; 04; 01; 01]	1002.53	[04; 03; 05; 05; 04; 03; 07; 03; 07; 05; 04]	334.20
5	[04; 03; 03; 04; 01; 01; 03; 01; 03; 01; 01]	1002.53	[07; 09; 04; 03; 05; 09; 03; 03; 05; 01; 01]	1002.53
6	[03; 02; 02; 03; 02; 02; 03; 03; 03; 01; 01]	1002.53	[14; 04; 02; 02; 10; 05; 04; 04; 03; 01; 01]	1002.53
7	[03; 02; 02; 02; 02; 03; 04; 02; 03; 01; 01]	1002.53	[09; 12; 02; 02; 07; 04; 05; 03; 04; 01; 01]	1002.53
8	[02; 02; 03; 02; 02; 03; 03; 02; 03; 01; 01]	1002.53	[05; 05; 04; 04; 04; 03; 06; 05; 06; 05; 03]	379.45
9	[02; 02; 03; 02; 02; 03; 03; 03; 03; 01; 01]	1002.53	[02; 05; 14; 08; 04; 05; 04; 02; 04; 01; 01]	1002.53
10	[02; 02; 03; 02; 02; 02; 03; 02; 03; 02; 02]	758.86	[05; 03; 06; 04; 04; 05; 07; 03; 07; 03; 03]	375.65
11	[03; 02; 04; 02; 02; 02; 03; 02; 03; 01; 01]	1002.53	[06; 04; 06; 06; 04; 03; 06; 03; 06; 03; 03]	379.44
12	[03; 02; 02; 02; 02; 03; 04; 02; 03; 01; 01]	1002.53	[03; 03; 05; 05; 03; 03; 08; 05; 08; 03; 03]	334.20
13	[03; 02; 02; 02; 02; 02; 03; 04; 03; 01; 01]	1002.53	[04; 04; 04; 02; 03; 02; 06; 08; 03; 01; 01]	1002.53
14	[02; 02; 03; 03; 02; 02; 04; 02; 03; 01; 01]	1002.53	[08; 03; 10; 04; 09; 03; 03; 02; 06; 01; 01]	1002.53
15	[02; 02; 02; 02; 02; 03; 03; 02; 03; 01; 01]	1002.53	[03; 05; 07; 05; 04; 04; 06; 03; 06; 03; 04]	379.44
16	[02; 02; 02; 04; 02; 02; 03; 02; 04; 01; 01]	1002.53	[03; 03; 09; 03; 06; 05; 03; 05; 03; 01; 01]	1002.53
17	[03; 02; 02; 03; 02; 02; 04; 01; 04; 01; 01]	1002.53	[03; 03; 06; 04; 04; 04; 06; 06; 06; 03; 05]	379.45
18	[03; 03; 02; 02; 02; 02; 03; 03; 03; 01; 01]	1002.53	[04; 03; 05; 06; 03; 04; 06; 05; 06; 05; 03]	379.44
19	[03; 02; 03; 02; 02; 03; 03; 02; 03; 01; 01]	1002.53	[05; 08; 12; 03; 02; 03; 10; 03; 01; 01; 01]	1002.53
20	[04; 04; 01; 04; 01; 03; 04; 01; 01; 01; 01]	2276.53	[03; 03; 05; 05; 05; 04; 05; 03; 07; 03; 07]	455.33
21	[11; 03; 01; 02; 01; 01; 02; 01; 01; 01; 01]	2276.53	[07; 02; 07; 03; 04; 11; 06; 05; 03; 01; 01]	1002.53
22	[02; 02; 03; 03; 02; 03; 03; 02; 03; 01; 01]	1002.53	[04; 06; 05; 05; 03; 04; 07; 03; 07; 03; 03]	334.20
23	[03; 02; 02; 02; 02; 04; 03; 02; 03; 01; 01]	1002.53	[09; 04; 07; 07; 02; 02; 07; 05; 05; 01; 01]	1002.53
24	[02; 03; 02; 03; 03; 02; 03; 01; 04; 01; 01]	1002.53	[09; 04; 02; 11; 03; 03; 03; 10; 03; 01; 01]	1002.53
25	[02; 03; 02; 03; 02; 03; 03; 02; 03; 01; 01]	1002.53	[03; 03; 05; 05; 03; 03; 06; 05; 06; 06; 05]	379.44
	75 THREADS		*100 THREADS*	
1	[04; 10; 06; 08; 06; 09; 10; 04; 10; 04; 04]	250.65	[07; 14; 08; 08; 07; 08; 14; 08; 13; 06; 07]	187.84
2	[06; 05; 07; 08; 07; 05; 10; 07; 10; 05; 05]	227.68	[11; 05; 14; 10; 10; 05; 14; 05; 13; 06; 07]	200.53
3	[06; 10; 09; 06; 05; 07; 09; 05; 10; 04; 04]	252.97	[06; 09; 08; 09; 10; 06; 12; 13; 14; 07; 06]	189.74
4	[07; 04; 08; 08; 06; 06; 10; 06; 11; 05; 04]	250.65	[06; 07; 12; 09; 07; 10; 13; 08; 13; 08; 07]	175.14
5	[20; 05; 06; 26; 03; 05; 03; 02; 03; 01; 01]	1002.53	[07; 10; 10; 08; 06; 08; 13; 06; 13; 10; 09]	187.84
6	[11; 04; 06; 07; 05; 04; 10; 06; 11; 04; 07]	250.65	[14; 06; 11; 09; 05; 05; 13; 08; 13; 09; 07]	200.53
7	[04; 08; 08; 08; 05; 04; 11; 04; 09; 06; 08]	252.97	[07; 05; 11; 11; 05; 08; 14; 09; 15; 06; 09]	200.53
8	[07; 05; 07; 08; 06; 07; 10; 05; 10; 05; 05]	227.68	[07; 06; 08; 08; 06; 09; 12; 13; 12; 07; 12]	189.74
9	[04; 04; 10; 09; 04; 07; 11; 04; 10; 04; 08]	250.65	[05; 09; 13; 08; 07; 07; 13; 05; 13; 15; 05]	200.53
10	[05; 06; 06; 06; 06; 05; 10; 06; 13; 06; 06]	250.45	[08; 05; 11; 10; 10; 06; 15; 07; 13; 09; 06]	200.53
11	[03; 05; 02; 07; 13; 14; 05; 18; 06; 01; 01]	1002.53	[05; 05; 11; 09; 11; 06; 19; 09; 15; 05; 05]	200.53
12	[04; 05; 08; 09; 05; 06; 10; 04; 10; 08; 06]	250.65	[12; 05; 09; 11; 09; 10; 13; 06; 14; 06; 05]	200.53
13	[05; 05; 09; 07; 05; 06; 11; 04; 14; 04; 05]	250.65	[05; 07; 15; 09; 05; 10; 14; 05; 15; 05; 10]	200.53

Table 3 Thread distributions found and the makespan for PSO experiments (continued)

	25 THREADS		*50 THREADS*	
Repetition	*Threads in each pool*	*Makespan**	*Threads in each pool*	*Makespan**
14	[05; 04; 14; 10; 07; 04; 09; 04; 10; 04; 04]	252.97	[19; 12; 13; 08; 08; 14; 03; 17; 04; 01; 01]	1002.53
15	[08; 05; 08; 07; 05; 05; 10; 06; 09; 06; 06]	252.97	[10; 05; 09; 09; 09; 07; 14; 05; 13; 12; 07]	200.53
16	[04; 09; 06; 07; 05; 04; 10; 05; 11; 09; 05]	250.65	[07; 13; 08; 14; 06; 06; 14; 06; 12; 07; 07]	189.74
17	[09; 05; 07; 06; 04; 04; 12; 06; 12; 06; 04]	250.65	[05; 06; 11; 10; 07; 05; 16; 06; 14; 07; 13]	200.53
18	[05; 05; 06; 11; 04; 04; 11; 05; 11; 08; 05]	250.65	[05; 13; 09; 09; 06; 08; 15; 09; 14; 06; 06]	200.53
19	[04; 06; 08; 06; 08; 04; 10; 05; 11; 04; 09]	250.65	[10; 14; 06; 12; 17; 05; 09; 19; 06; 01; 01]	1002.53
20	[04; 04; 11; 09; 06; 05; 14; 04; 10; 05; 05]	250.65	[05; 05; 10; 08; 05; 09; 15; 05; 13; 06; 19]	200.53
21	[26; 07; 04; 05; 03; 11; 08; 05; 04; 01; 01]	1002.53	[07; 06; 11; 11; 09; 08; 12; 06; 11; 12; 07]	206.98
22	[07; 04; 10; 08; 04; 07; 11; 04; 10; 05; 05]	250.65	[14; 27; 14; 19; 08; 03; 05; 02; 06; 01; 01]	1002.53
23	[06; 04; 07; 10; 04; 05; 12; 06; 11; 06; 04]	250.65	[05; 13; 10; 11; 06; 06; 13; 08; 14; 06; 08]	200.53
24	[03; 22; 07; 08; 19; 02; 04; 04; 04; 01; 01]	1002.53	[06; 12; 10; 12; 08; 08; 12; 10; 12; 05; 05]	200.53
25	[05; 08; 08; 07; 05; 04; 12; 05; 11; 04; 06]	250.65	[10; 07; 15; 09; 07; 09; 14; 05; 13; 05; 06]	200.53

Note: * time in seconds.

The results found by CSO were also collected from the four scenarios, as explained in Table 4. The table contemplates results for the four different numbers of threads used. On the first part of the table, 25 threads; on the second one, 50; on the third part, 75 threads; and on the fourth one, 100 threads. The information reported below follows this organisation of the tables. For 25 threads, the algorithm found the lowest makespan to be 1002.53 s in twenty-two distributions. Since the probability for such a value is very large, the results led to this number. Some of the settings for this makespan were repetitions 1, 3 and 4. For this same number of threads, the largest makespan found was 2276.53 s with the repetition settings 2, 5 and 10. For 50 threads, the lowest makespan found was 334.20 s with the 12th repetition setting, while the largest makespan was 1002.53 s for ten different distributions, some of them being repetitions 1, 2 and 10. For 75 threads, the algorithm found the lowest makespan 250.65 s in eight distributions, the same as in repetitions 1, 4 and 7 and the largest makespan 1002.53 s was from repetitions 2, 12 and 15. Finally, with 100 threads we found the lowest makespan 189.74 s with the distributions of repetitions 21 and 25, and the largest makespan 1002.53 s was found in repetition 24.

Table 4 Thread distributions found and makespan for CSO experiments

	25 THREADS		*50 THREADS*	
Repetition	*Threads in each pool*	*Makespan**	*Threads in each pool*	*Makespan**
1	[02; 02; 02; 02; 02; 03; 05; 02; 03; 01; 01]	1002.53	[08; 07; 06; 05; 08; 02; 03; 02; 05; 01; 01]	1002.53
2	[02; 11; 02; 02; 01; 01; 02; 01; 01; 01; 01]	227.65	[07; 09; 06; 03; 04; 04; 06; 05; 04; 01; 01]	1002.53
3	[03; 03; 02; 03; 02; 02; 03; 02; 03; 01; 01]	1002.53	[05; 04; 05; 04; 04; 05; 05; 04; 05; 04; 04]	455.33
4	[02; 02; 02; 03; 03; 02; 04; 01; 04; 01; 01]	1002.53	[04; 04; 05; 05; 06; 03; 05; 03; 05; 06; 04]	455.33
5	[05; 09; 02; 01; 01; 01; 02; 01; 01; 01; 01]	227.65	[03; 05; 04; 04; 03; 05; 06; 06; 06; 04; 04]	379.45
6	[02; 02; 02; 02; 02; 04; 03; 02; 04; 01; 01]	1002.53	[03; 04; 03; 07; 03; 05; 06; 03; 07; 05; 04]	500.86
7	[03; 02; 02; 02; 02; 02; 03; 03; 04; 01; 01]	1002.53	[06; 03; 04; 04; 03; 04; 06; 04; 05; 04; 03]	455.33
8	[03; 02; 04; 02; 02; 02; 03; 02; 03; 01; 01]	1002.53	[06; 03; 04; 04; 03; 04; 06; 04; 05; 04; 03]	455.33
9	[02; 02; 02; 02; 03; 02; 04; 02; 04; 01; 01]	1002.53	[03; 04; 04; 04; 05; 04; 06; 04; 07; 04; 05]	379.45
10	[03; 01; 03; 08; 03; 01; 02; 01; 01; 01; 01]	227.65	[06; 04; 04; 07; 04; 05; 09; 05; 03; 01; 02]	1002.53
11	[04; 01; 03; 04; 01; 01; 04; 01; 04; 01; 01]	1002.53	[03; 09; 06; 06; 07; 03; 06; 02; 03; 01; 04]	1002.53
12	[02; 03; 02; 04; 02; 02; 03; 02; 03; 01; 01]	1002.53	[03; 03; 05; 08; 03; 03; 07; 04; 07; 04; 03]	334.20
13	[02; 02; 02; 04; 02; 02; 03; 02; 03; 01; 01]	1002.53	[03; 05; 06; 03; 04; 05; 07; 05; 06; 03; 03]	500.86
14	[03; 02; 02; 03; 04; 02; 03; 01; 03; 01; 01]	1002.53	[05; 02; 05; 10; 11; 04; 03; 05; 03; 01; 01]	1002.53
15	[02; 03; 02; 02; 03; 02; 03; 03; 03; 01; 01]	1002.53	[05; 02; 04; 10; 04; 02; 04; 14; 03; 01; 01]	1002.53
16	[03; 02; 03; 02; 02; 03; 03; 02; 03; 01; 01]	1002.53	[03; 04; 08; 04; 04; 05; 05; 03; 06; 05; 03]	455.33
17	[02; 03; 04; 02; 02; 02; 04; 01; 03; 01; 01]	1002.53	[09; 05; 06; 08; 02; 04; 03; 06; 05; 01; 01]	1002.53
18	[03; 02; 03; 03; 02; 02; 03; 02; 03; 01; 01]	1002.53	[06; 03; 05; 04; 03; 06; 06; 04; 06; 03; 03]	379.45

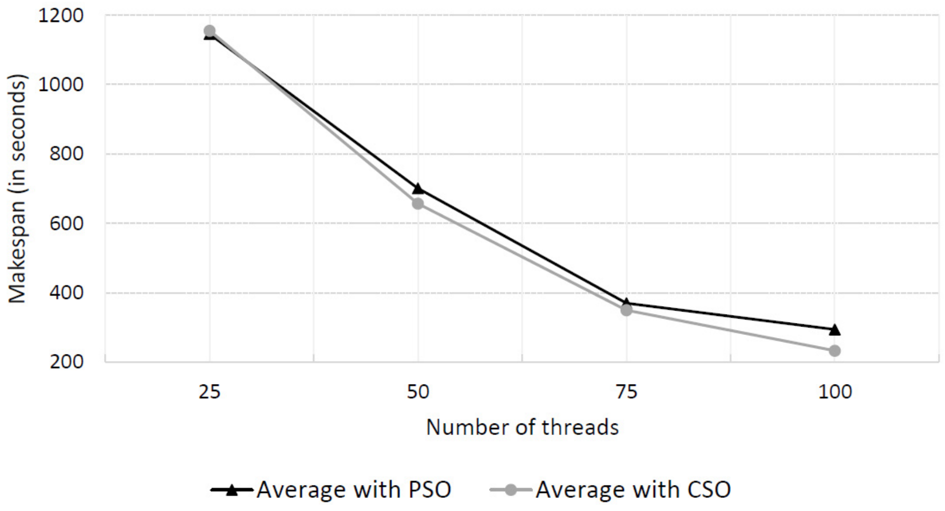
Table 4 Thread distributions found and makespan for CSO experiments (continued)

	25 THREADS		*50 THREADS*	
Repetition	*Threads in each pool*	*Makespan**	*Threads in each pool*	*Makespan**
19	[02; 02; 03; 03; 02; 02; 03; 02; 03; 01; 01]	1002.53	[10; 03; 07; 14; 02; 02; 05; 02; 03; 01; 01]	1002.53
20	[03; 03; 02; 03; 02; 02; 03; 02; 03; 01; 01]	1002.53	[03; 05; 07; 04; 03; 03; 06; 05; 07; 04; 04]	379.45
21	[02; 02; 03; 03; 02; 02; 03; 01; 03; 01; 01]	1002.53	[02; 04; 06; 04; 05; 07; 08; 04; 08; 01; 01]	1002.53
22	[02; 02; 02; 02; 02; 04; 03; 02; 04; 01; 01]	1002.53	[12; 05; 08; 05; 05; 05; 03; 02; 03; 01; 01]	1002.53
23	[02; 04; 03; 03; 02; 02; 03; 01; 03; 01; 01]	1002.53	[03; 03; 04; 04; 03; 03; 07; 05; 10; 03; 04]	375.65
24	[02; 02; 04; 02; 03; 02; 03; 02; 03; 01; 01]	1002.53	[03; 03; 05; 04; 03; 07; 09; 04; 06; 03; 03]	379.45
25	[02; 02; 02; 03; 02; 02; 03; 03; 04; 01; 01]	1002.53	[06; 03; 04; 03; 06; 04; 05; 05; 05; 04; 05]	500.86
	75 THREADS		*100 THREADS*	
1	[08; 06; 07; 08; 07; 04; 11; 04; 11; 04; 05]	250.65	[05; 05; 13; 13; 05; 08; 13; 06; 15; 06; 11]	200.53
2	[25; 03; 09; 04; 06; 04; 04; 02; 03; 03; 01]	1002.53	[08; 10; 10; 17; 05; 06; 12; 05; 11; 10; 06]	206.98
3	[04; 06; 08; 08; 04; 08; 13; 04; 09; 04; 05]	252.97	[07; 05; 14; 09; 08; 05; 14; 05; 15; 12; 06]	200.53
4	[04; 04; 06; 11; 04; 06; 10; 07; 13; 04; 06]	250.65	[07; 05; 09; 08; 05; 06; 17; 09; 13; 09; 10]	200.53
5	[04; 09; 07; 07; 07; 05; 10; 07; 13; 04; 06]	252.97	[05; 06; 12; 10; 12; 05; 14; 06; 15; 10; 05]	200.53
6	[05; 05; 08; 09; 09; 09; 08; 04; 08; 05; 04]	284.59	[05; 07; 11; 09; 09; 05; 15; 06; 14; 05; 14]	200.53
7	[04; 04; 08; 06; 05; 09; 10; 04; 10; 04; 10]	250.65	[10; 09; 09; 08; 07; 12; 12; 05; 14; 07; 06]	200.53
8	[04; 08; 07; 07; 06; 05; 11; 07; 09; 05; 06]	252.97	[06; 08; 09; 10; 07; 05; 15; 08; 15; 09; 07]	200.53
9	[05; 05; 07; 08; 05; 06; 10; 04; 11; 05; 09]	250.65	[09; 09; 09; 10; 07; 06; 11; 09; 11; 07; 12]	206.98
10	[05; 05; 09; 10; 05; 07; 09; 06; 08; 04; 07]	284.59	[06; 06; 10; 08; 05; 11; 15; 08; 16; 10; 05]	200.53
11	[05; 09; 06; 07; 03; 05; 12; 11; 11; 03; 03]	334.19	[05; 08; 14; 09; 06; 05; 11; 09; 11; 14; 08]	206.98
12	[09; 14; 12; 06; 02; 08; 05; 13; 04; 01; 01]	1002.53	[12; 07; 09; 10; 05; 09; 13; 10; 15; 05; 05]	200.53
13	[04; 05; 07; 07; 05; 07; 09; 06; 09; 09; 07]	252.97	[07; 11; 14; 11; 05; 07; 13; 07; 13; 07; 05]	200.53
14	[04; 06; 07; 08; 05; 04; 10; 04; 18; 05; 04]	250.65	[11; 05; 09; 10; 07; 05; 13; 08; 16; 11; 05]	200.53
15	[20; 11; 06; 03; 14; 02; 04; 10; 03; 01; 01]	1002.53	[05; 08; 09; 10; 07; 11; 13; 05; 14; 12; 06]	200.53
16	[06; 04; 07; 06; 06; 07; 10; 10; 09; 04; 06]	252.97	[12; 05; 12; 08; 07; 05; 16; 07; 12; 07; 09]	200.53
17	[05; 05; 09; 06; 05; 09; 09; 05; 10; 06; 06]	252.97	[05; 07; 08; 09; 06; 14; 16; 09; 14; 06; 06]	200.53
18	[04; 04; 07; 08; 07; 04; 09; 09; 09; 08; 04]	252.97	[08; 07; 12; 09; 08; 09; 13; 05; 13; 07; 09]	200.53
19	[04; 04; 07; 06; 06; 05; 11; 05; 11; 10; 06]	250.65	[05; 06; 12; 15; 07; 06; 12; 08; 14; 09; 06]	200.53
20	[05; 05; 06; 10; 05; 06; 10; 05; 10; 07; 06]	250.45	[05; 05; 13; 11; 07; 05; 14; 10; 15; 06; 09]	2002.53
21	[06; 04; 06; 07; 04; 06; 10; 04; 10; 04; 08]	250.65	[06; 09; 11; 10; 08; 06; 12; 09; 12; 09; 08]	189.74
22	[04; 04; 09; 07; 10; 06; 11; 05; 09; 05; 05]	252.97	[07; 06; 10; 11; 06; 05; 13; 12; 13; 09; 08]	200.53
23	[05; 06; 07; 07; 05; 06; 09; 06; 08; 08; 08]	284.59	[05; 15; 13; 10; 06; 05; 16; 05; 15; 05; 05]	200.53
24	[04; 06; 06; 10; 05; 07; 09; 08; 04; 09; 07]	252.97	[19; 17; 16; 22; 03; 10; 04; 03; 03; 02; 01]	1002.53
25	[05; 08; 10; 09; 06; 04; 10; 04; 11; 04; 04]	250.65	[08; 11; 08; 11; 08; 11; 12; 07; 12; 06; 06]	189.74

Note: * time in seconds.

In both algorithms, the lowest makespan was noticed in distributions where threads were better split. That is, in those distributions where threads were very concentrated in some pools, but were few in others, there was a degradation in performance. It is noticed that in all scenario results, PSO obtained lower global makespan compared to CSO. So, for PSO to reach a global result better than the CSO, it will need 25 repetitions. In order to compare the performance of each of the algorithms, an average of 25 repetitions for each scenario was calculated. Aware of this, it is clear that CSO presented better average results than PSO, which seems to be a contradiction. What we emphasise, though, is that PSO can present a better overall result, while CSO presents a better average result. In face of that, every time it is necessary to change the case study, the number of threads, one of the algorithms is used to find the best distribution, and the execution is performed only once, it will obtain a better average result when using the CSO. Furthermore, if we compare the average results of each technique, with the increment of threads a greater difference between the techniques becomes evident. The graph in Figure 5 shows the results obtained.

Figure 5 Evolution of the makespan mean as a function of the increment in the number of threads



We need to consider the relationship that exists between the times of the tasks and the number of threads with their makespan obtained by the distribution. For instance, on tasks 7 and 9 the processing time was the largest; looking back the results of the lowest and largest makespan in each scenario, in distributions where makespan was lower, there was a greater concentration of threads in those elements, but also, there was no lack of threads in the final elements. It means threads were well distributed, so that all tasks that needed more threads could receive them, but not in excess. An example of this finding is repetition 9 of the Table 4, which shows the CSO experiment with 50 threads. In the settings where there was the largest makespan, pools that needed more threads due to time did not receive them. Furthermore, there are few threads left for the final tasks, since they are too concentrated on pools that do not need them, and in very large numbers. This last point can be analysed, for instance, in repetition 5 from the PSO experiment on Table 3, with 75 threads.

There was a drop from the average result with 25 threads to the average result with 50 threads, and this drop is lower with 75 threads, and even lower than 75 with 100 threads. This means that the result tends to be stable up to a number of threads and possibly deteriorate the performance due to the concurrency between the threads. Another point to be highlighted is that with 100 threads we were able to obtain a result equal to that when using 25 threads; by this we observed that a bad distribution of 100 threads can harm performance and not necessarily improve it, resulting in the same makespan when only 25 threads are used. Therefore, a bad distribution considerably affects makespan, so that the best optimisation of resources is necessary. This happens because threads are concentrated in pools that are not used for the best performance, for being idle.

5.7 Threat to validity

The first threat to validity was in the construction. In order to reduce problems of this threat with instrumentation and noise source, before carrying out the experiment, the experimentation protocol was defined: the formulation of the research question, the hypothesis, the definition of variables, information about the execution environment, support tools, execution and data collection. Regarding internal validity threats, and in order to minimise interference in the algorithm execution time, the entire experiment was performed by the same person on the same machine, using minimal resources and disconnected from the Internet during executions. Regarding external validity threats, it can be said that it was mitigated from the studies and analysis carried out on works related to our research. Finally, the threat of reliability was minimised through the use of a well-explained experimentation protocol and applied in the execution of all experiments performed, which even made it possible to compare data.

6 Conclusions

The Enterprise Applications Integration area has become essential for supporting companies' business processes. This area facilitates the implementation of integration processes, which aim to share data and functionality among applications. Ideally, an integration process should occur without degrading performance of the run-time system of integration platforms. The Task-Based execution model uses threads at task level, and execution decisions are made locally by each task. The local pool strategy consists of a pool of threads for each task that is part of the integration process. The PSO and CSO meta-heuristics were analysed and studied to solve the problem of computational resources distribution.

From the proposed experimental study: i) we confirmed the hypothesis that CSO would have a better performance, since according to comparative results, it has a better average result than PSO. In other words, if we run the algorithms once, the probability that CSO will have a better result is greater; ii) it is also necessary to list that PSO always obtained the lowest individual results, in all different modification options of the number of threads. This means that PSO is able to find the lowest global makespan aspect with the same 25 repetitions used by CSO, although it does not always last – despite finding the lowest result among 50 results (adding 25 executions of each algorithm), PSO also found many unsatisfactory results, which even led it to have a lower average; iii) we emphasise that for an urgent need to use an integration process, CSO would be more efficient, but if there is more preparation time, it would be valid to use an analysis between the two algorithms in order to mitigate the best use of threads.

The distribution of threads in the different pools is the main point for a lower or larger makespan in the experiment results. At this point, the task times – which were not all the same – should also be taken into account. In tasks where time was longer, whenever more threads were available for those threads, but without excess as to have enough threads left over for the final tasks, the lowest makespans were registered. Everything indicates that the reason for the worst distributions, that is, the ones that makespan was the highest on experiments, was due to the bad distribution and concentration of many threads on tasks that did not need them, leaving thus few threads for other tasks.

The work with heuristics is interesting for this type of problem, as it was possible to find the best distributions with the optimal number of threads for local pools in an integration process based on experiments with both PSO and CSO meta-heuristics. On the other hand, it is clear that the use of this practice on a daily basis is little, as it becomes unfeasible due to the effort of time necessary to apply it. At this point, heuristics need to move forward, as execution demands high costs. As future work, we suggest: i) using other case studies to capture possible changes linked specifically to the case study; ii) add new meta-heuristics to the comparison scope and iii) analyse and enable a reduction in the time that a meta-heuristic needs to find a satisfactory result for the EAI area.

Acknowledgements

This work was supported by the Research Support Foundation of the State of Rio Grande do Sul in Brazil (FAPERGS), under Grant 17/2551-0001206-2; the Coordination for the Brazilian Improvement of Higher Education Personnel (CAPES) and the Brazilian National Council for Scientific and Technological (CNPq) under Grant 309315/2020-4.

References

AbdelAziz, A.M., Ghany, K.K.A., Soliman, T.H.A. and El-Magd Sewisy, A.A. (2020) 'A parallel multi-objective swarm intelligence framework for big data analysis', *International Journal of Computer Applications in Technology*, Vol. 63, No. 3, pp.200–212.

Abdullahi, M., Ngadi, M.A. and Dishing, S.I. (2017) 'Chaotic symbiotic organisms search for task scheduling optimization on cloud computing environment', *Proceedings of the 6th ICT International Student Project Conference (ICT-ISPC)*, IEEE, pp.1–4.

Alexander, C. (1977) *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press.

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

Bilgaiyan, S., Sagnika, S. and Das, M. (2014) 'Workflow scheduling in cloud computing environment using cat swarm optimization', *Proceedings of the IEEE International Advance Computing Conference (IACC)*, IEEE, pp.680–685.

Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A. and Kennedy, K. (2005) 'Task scheduling strategies for workflowbased applications in grids', *IEEE International Symposium on Cluster Computing and the Grid*, Vol. 2, pp.759–767.

Boehm, M., Habich, D., Preissler, S., Lehner, W. and Wloka, U. (2011) 'Costbased vectorization of instance-based integration processes', *Information Systems*, Vol. 36, No. 1, pp.3–29.

Bousselmi, K., Ben Hamida, S. and Rukoz, M. (2020) 'Bi-objective CSO for big data scientific workflows scheduling in the cloud: case of ligo workflow', *Proceedings of the 15th International Conference on Software Technologies*, Vol. 1.

Bratton, D. and Kennedy, J. (2007) 'Defining a standard for particle swarm optimization', *IEEE Swarm Intelligence Symposium*, IEEE, pp.120–127.

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

Chu, S-C., Tsai, P-W. and Pan, J-S. (2006) 'Cat swarm optimization', *Pacific Rim International Conference on Artificial Intelligence*, pp.854–858.

Clerc, M. (2010) *Particle Swarm Optimization*, Vol. 93, John Wiley & Sons.

Du, Y., Wang, J.L. and Lei, L. (2019) 'Multi-objective scheduling of cloud manufacturing resources through the integration of cat swarm optimization and firefly algorithm', *Advances in Production Engineering and Management*, Vol. 14, No. 3, pp.333–342.

Eberhart, R. and Kennedy, J. (1995) 'Particle swarm optimization', *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 4, pp.1942–1948.

Frantz, R.Z., Corchuelo, R. and Molina-Jiménez, C. (2012) 'A proposal to detect errors in enterprise application integration solutions', *Journal of Systems and Software*, Vol. 85, No. 3, pp.480–497.

Frantz, R.Z., Corchuelo, R., Basto-Fernandes, V., Rosa-Sequeira, F., Roos-Frantz, F. and Arjona, J.L. (2021) 'A cloud-based integration platform for enterprise application integration: a model-driven engineering approach', *Software: Practice and Experience*, Vol. 51, No. 4, pp.824–847.

Freire, D.L., Frantz, R.Z. and Roos-Frantz, F. (2019a) 'Ranking enterprise application integration platforms from a performance perspective: an experience report', *Software: Practice and Experience*, Vol. 49, No. 5, pp.921–941.

Freire, D.L., Frantz, R.Z., Roos-Frantz, F. and Sawicki, S. (2019b) 'A methodology to rank enterprise application integration platforms from a performance perspective: an analytic hierarchy process-based approach', *Enterprise Information Systems*, Vol. 13, No. 9, pp.1292–1322.

Freire, D.P., Frantz, R.Z. and Roos-Frantz, F. (2020) 'Towards optimal thread pool configuration for run-time systems of integration platforms', *International Journal of Computer Applications in Technology*, Vol. 62, No. 2, pp.129–147.

Gabi, D., Ismail, A.S. and Dankolo, N.M. (2019) 'Minimized makespan based improved cat swarm optimization for efficient task scheduling in cloud datacenter', *Proceedings of the 3rd High Performance Computing and Cluster Technologies Conference*, pp.16–20.

Gabi, D., Ismail, A.S., Zainal, A. and Zakaria, Z. (2017) 'Solving task scheduling problem in cloud computing environment using orthogonal taguchi-cat algorithm', *International Journal of Electrical and Computer Engineering (2088-8708)*, Vol. 7, No. 3, pp.1489–1497.

Game, P.S., Vaze, V. and Emmanuel, M. (2020) 'Bio-inspired optimization: metaheuristic algorithms for optimization', *arXiv preprint arXiv:2003.11637*.

Gupta, S., Agarwal, I. and Singh, R.S. (2019) 'Workflow scheduling using jaya algorithm in cloud', *Concurrency and Computation: Practice and Experience*, Vol. 31, No. 17.

Gupta, S., Singh, R.S. and Anand, A. (2018) 'Cloudlet scheduling using merged CSO algorithm', *Proceedings of the 5th International Conference on Parallel, Distributed and Grid Computing (PDGC)*, IEEE, pp.278–283.

Haugg, I.G., Frantz, R.Z., Roos-Frantz, F., Sawicki, S. and Zucolotto, B. (2019) 'Towards optimisation of the number of threads in the integration platform engines using simulation models based on queueing theory', *Revista Brasileira de Computação Aplicada*, Vol. 11, No. 1, pp.48–58.

Hohpe, G. and Woolf, B. (2003) *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley.

Irman, A., Febianti, E. and Khasanah, U. (2019) 'Minimizing makespan on flow shop scheduling using campbel dudek and smith, particle swarm optimization, and proposed heuristic algorithm', *Proceedings of the IOP Conference Series: Materials Science and Engineering*, IOP Publishing, Vol. 673, pp.12–99.

Jedlitschka, A. and Pfahl, D. (2005) 'Reporting guidelines for controlled experiments in software engineering', *Proceedings of the International Symposium on Empirical Software Engineering*, pp.95–104.

Jing, Z., Shou-Bin, D. and De-Yu, T. (2018) 'Task scheduling algorithm in cloud computing based on invasive tumor growth optimization', *Chinese Journal of Computer*, Vol. 41, No. 6, pp.1140–1155.

Junaid, M., Sohail, A., Ahmed, A., Baz, A., Khan, I.A. and Alhakami, H. (2020) 'A hybrid model for load balancing in cloud using file type formatting', *IEEE Access*, Vol. 8, pp.118135–118155.

Kaleche, R., Bendaoud, Z. and Bouamrane, K. (2020) 'Bio-inspired metaheuristics: a comprehensive survey', *International Journal of Organizational and Collective Intelligence (IJOCI)*, Vol. 10, No. 4, pp.1–18.

Leonard, N.E. and Levine, W.S. (1995) *Using MATLAB to analyze and design Control Systems*, Benjamin-Cummings Publishing Company, 1995.

Lima, F.R., Frantz, R.Z., Sawicki, S. and Roos-Frantz, F. (2019) 'Cat swarm optimization applied to makespan reduction in application integrations', *Advances in Engineering Research*, Nova Science Publishers, Inc., Vol. 31, pp.1–21.

Manikas, K. (2016) 'Revisiting software ecosystems research: a longitudinal literature study', *Journal of Systems and Software*, Vol. 117, pp.84–103.

Maurya, A.K. and Tripathi, A.K. (2018) 'Deadline-constrained algorithms for scheduling of bag-of-tasks and workflows in cloud computing environments', *Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications*, pp.6–10.

Natesha, B.V., Sharma, N.K., Domanal, S. and Guddeti, R.M.R. (2018) 'Gwots: Grey wolf optimization based task scheduling at the green cloud data center', *Proceedings of the 14th International Conference on Semantics, Knowledge and Grids (SKG)*, IEEE, pp.181–187.

Parahyba, F., Frantz, R.Z. and Roos-Frantz, F. (2021) 'On the estimation of makespan in runtime systems of enterprise application integration platforms: a mathematical modelling approach', *International Journal of Computer Applications in Technology*, Vol. 67, No. 1, pp.17–28.

Perry, D.E., Porter, A.A. and Votta, L.G. (2000) 'Empirical studies of software engineering: a roadmap', *Proceedings of the Conference on the Future of Software Engineering*, pp.345–355.

Semlali, S.C.B., Riffi, M.E. and Chebihi, F. (2018) 'Optimization of makespan in job shop scheduling problem by hybrid chicken swarm algorithm', *Proceedings of the International Conference on Advanced Intelligent Systems for Sustainable Development*, Springer, pp.358–369.

Shi, Y. and Eberhart, R.C. (1999) 'Empirical study of particle swarm optimization', *Proceedings of the Congress on Evolutionary Computation (CEC)*, IEEE, Vol. 3, pp.1945–1950.

Shojaee, R., Faragardi, H.R., Alaee, S. and Yazdani, N. (2012) 'A new cat swarm optimization based algorithm for reliability-oriented task allocation in distributed systems', *Proceedings of the 6th International Symposium on Telecommunications (IST)*, IEEE, pp.861–866.

Singh, P., Dutta, M. and Aggarwal, N. (2017) 'A review of task scheduling based on meta-heuristics approach in cloud computing', *Knowledge and Information Systems*, Vol. 52, No. 1, pp.1–51.

Slowik, A. (2020) *Swarm Intelligence Algorithms: Modifications and Applications*, CRC Press.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B. and Wessln, A. (2012) *Experimentation in Software Engineering*, Springer.

Xu, S.P., Wu, D., Kong, F. and Ji, Z. (2017) 'Solving flexible job-shop scheduling problem by improved chicken swarm optimization algorithm', *Journal of System Simulation*, Vol. 29, No. 7, pp.1497–1505.

Zarrouk, R. and Jemai, A. (2018) 'Performance evaluation of particles coding in particle swarm optimization with self-adaptive parameters for flexible job shop scheduling problem', *Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, Springer, pp.396–407.

Note

1 https://github.com/gca-research-group/algorithms-pso-cso