Chapter

CAT SWARM OPTIMIZATION APPLIED TO MAKESPAN REDUCTION IN APPLICATION INTEGRATIONS

*Francisco Lima, Rafael Z. Frantz, Sandro Sawicki *and Fabricia Roos-Frantz*

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

PACS 05.45-a, 52.35.Mw, 96.50.Fm. **Keywords:** Enterprise Application Integration, Makespan, Cat Swarm Optimization.

Abstract

Enterprise Application Integration provides methodologies and tools to design and implement integration solutions. A great number of companies rely on this research field to develop their integration solutions, since they provide the necessary technology to integrate different applications through external workflows. An integration solution is a new application whose function is to orchestrate a heterogeneous set of applications. It provides exogenous integration among different types of applications that compose the software ecosystem. The execution model of an integration solution schedules a certain number of threads to perform tasks on integration process. The implementation of the execution model can be of two types, namely the global pool

*E-mail address: sawicki@unijui.edu.br

model and the local pool model. Currently, the problem is distributing threads in a balanced way in order to reduce the makespan and, consequently, increase the performance of the integration solution. In this context, this chapter seeks to show that the high level of thread management can impact the performance of integration solutions. Thus, we propose the use of the Cat Swarm Optimization (CSO) technique to indicate the ideal number of threads in local pools. Our method was implemented and tested with a real-world integration process. Results demonstrate the reduction of execution time when compared to random distributions of threads and were validated using the statistical technique ANOVA and the Scott-Knott method.

1. Introduction

In business, the purpose of software applications is to support business processes. Typically, companies have a software ecosystem [1] with applications that have been developed without considering a possible integration. A software ecosystem involves different applications, acquired from different suppliers or developed in the company itself, which makes it heterogeneous [2]. The research area of software engineering that studies integration methodologies is known as Enterprise Application Integration (EAI) [3]. Application integration is a strategic approach for connecting applications [4].

The EAI provides methodologies, techniques and tools so that the platforms can perform the integration of the applications [5] observing patterns of integration, proposed by Hohpe e Woolf [6]. The EAI considers the Pipes-and-Filters architecture style [7], in which pipes represent message channels and filters represent tasks of the concrete integration solution that process data encapsulated in messages. The EAI can be performed through topologies and integration styles [6]. Topologies are models that represent how systems can be organized. Figure 1 presents integration processes as an intermediate layer between business processes.

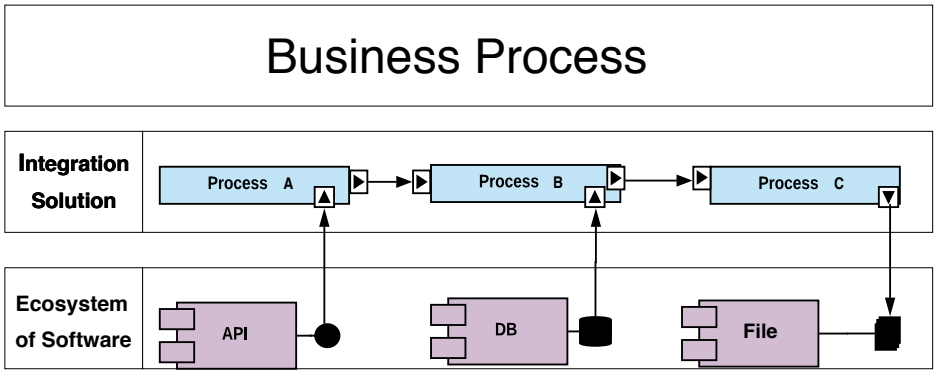

Figure 1. Application integration environment

An integration solution connects multiple applications to an integration flow composed of tasks that process data encapsulated in messages. Each task executes an atomic operation that implements an integration pattern on the message [6], and these tasks are out of synch through communication channels following the Pipes-and-Filters architecture [7]. Mes-

sages enter the flow of the integration solution through the solution entry port and arrive at the tasks through message channels. For each message to be processed, the execution engine uses threads stored in pools. Integration solutions can be developed across several open-source integration platforms available, such as Guaraná SDK [8], Apache Camel [9], Spring Integration [10], WSO2 ESB [11], and Mule [12].

Integration platforms provide tools for the development of integration solutions, such: Domain Specific Language (DSL), used to create the conceptual model of an integration solution; a Software Development Kit (SDK), which enables the transformation of the conceptual model into executable code; an execution engine, responsible for executing the integration processes; monitoring tools that record and analyze statistical performance values and identify errors that may occur during message processing in integration solutions [13, 14].

The execution engine has the fundamental role of establishing processes and instances in the execution of tasks processed by units of time [5], considering the heterogeneity of the software applications and the different characteristics of the tasks. The performance of the execution engines is directly related to the average time of execution of tasks, called makespan. The execution engines can be based on two models: the global thread pool model and the local thread pool model. The global thread pool model is characterized by presenting a single thread pool to perform all the tasks of the integration solution, while the local thread pools model is characterized by presenting different thread pools associated with the tasks present in the integration solution.

In this context, this chapter presents a strategy to find the best thread distribution in local pools, using the least number of threads to perform the tasks of the integration process based on the meta-heuristic Cat Swarm Optimization. This work is motivated by the possibility of optimizing the running of the execution engine, considering that the amount of threads available for execution is limited. We also discuss the use of metaheuristic techniques to obtain a better distribution of threads in the local pools and also reduce the total execution time.

This chapter is organized as follows. Section 2 introduces the Background for Enterprise Application Integration, Styles, Platforms and Integration Topologies, as well as introduces the concepts of the Cat Swarm Optimization metaheuristic. Section 3 describes the work related to the theme of this research. Section 4 presents the proposed solution to find the optimal number of threads in local pools. Section 5 shows the execution environment and discusses the experimental results. Section 6 presents the conclusions of this work.

2. Background

This section presents a brief description of Enterprise Application Integration, its integration platforms, integration styles. It also introduces the conceptual model of execution of the integration platform, as well as the metaheuristic Cat Swarm Optimization.

2.1. Enterprise Application Integration

Enterprise Application Integration (EAI) aims to provide methodologies and tools to integrate the many applications of the enterprise software ecosystem. Thus, the EAI needs to

minimize the dependencies between the systems, maintaining a low coupling observing patterns and integration styles [2]. EAI seeks to prioritize a high level of planning in designing a solution so that the integration of applications is operationally feasible and efficient.

2.1.1. Integration Platforms

Integration platforms are used to develop integration projects. Platforms allow designing, develop and implement integration solutions. For the development of integration solutions, the platforms offer a set of tools that usually include a Domain Specific Language (DSL), an Application Programming Interface (API), an execution engine, and monitoring tools.

In integration platforms, each component used for developing, executing and monitoring a solution has specific functions that assist in the management and execution of solutions. There are a large number of platforms to implement integration solutions. Some prominent Open Source platforms: Apache Camel, Mule ESB and Spring Integration, which are described below:

Apache Camel: is an integration platform that aims to make integration projects focus on productivity. Apache Camel is based on the integration patterns defined by Hohpe and Woolf [6]. The Apache Camel framework offers many elements for most technologies within the context of integration. It has a perfect relationship with other frameworks, such: CDI, Spring Integration, Blueprint and Guice. Apache Camel's DSL XML presents framework performance, such as Spring Integration and Mule-Esb.

Mule Esb: for Dossot et al. [12], Mule is a communication system, event-driven corporate service bus and an integration broker platform. In addition, to complete, it includes several additional features rather than just an integration framework. Like other platforms, Mule also has its foundations in the Business Integration Standards (EIP) registered by Hohpe and Woolf [6]. Mule offers built-in tools on its platform like Mule Studio that contains an intuitive visual interface setting performance differences when logic is used in high complexity integration.

Spring Integration: provides an extensive set of tools for creating integration solutions that meet the demands of the business. The Spring platform has declarative adapter systems. Such adapters provide a higher level of abstraction in integration solution models. The Spring Framework provides essential supports such as files, FTP, JNS, TCP, HTTP or Web services. This framework uses the implementation of a messaging system based on the integration patterns presented by Hohpe and Woolf [6].

2.1.2. Conceptual Model of Implementation of Integration Platforms

The conceptual model of execution engines is based on a task queue and the number of threads to be allocated for the execution of the integration solution. This model executes task scheduling based on queueing theory, following a First-In-First-Out (FIFO) policy, and has structural components such: scheduler, task queue, threads, and monitors.

The scheduler can run multiple instances at the same time. To perform on instantiation, the scheduler uses the elements it has in the toolkit, such: a task queue, a list of work units, and three monitors [5]. The task queue is a priority queue that contains work units to process. Tasks are performed by a standard worker process in which they can be run

through the available threads. The task queue follows a pattern in which the first task that enters the queue is the first one to be executed.

The task queue stores the work units to be processed. The task queue commonly creates buffer pools and flow controls for the integration solution. Each work unit refers to the runtime process in which, in most cases, the deadline is set to the current time; this is, that the corresponding task can be executed as quickly as possible. Threads are computational resources that allow running a program. Implemented through a library and divided into small lines, threads enable tasks to be performed concurrently.

Figure 2 describes the execution engine model and its main elements responsible for implementing the integration solutions, formulated in the concept of the task-based execution model.

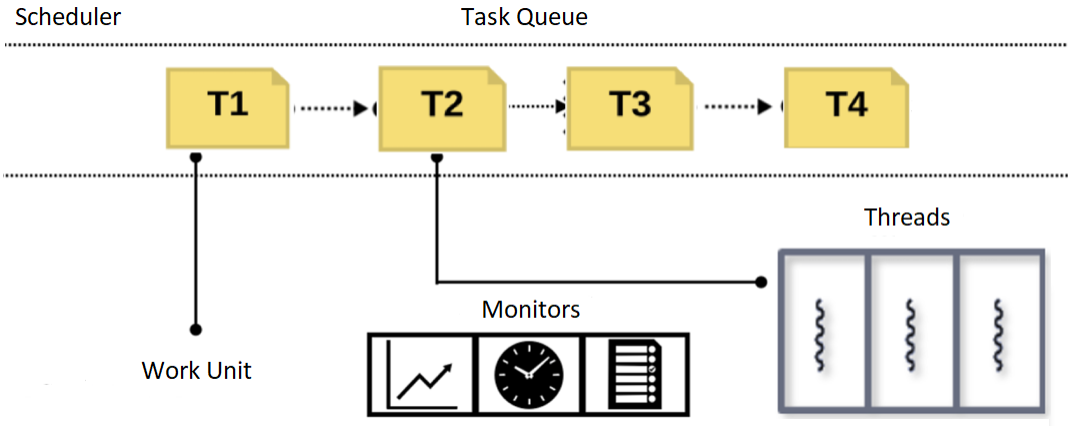

Figure 2. Conceptual model of the execution engine

2.2. Integration Styles

The following four integration styles are listed below: file transfer, database sharing, remote procedure call, and message sending.

File Transfer: occurs in export and import data exchange between applications. In this style, it is only necessary to have a standard data transfer mechanism that has the property of being able to be used by several programming languages and platforms. This style is considered simple with others, as long as the file format is maintained according to mutually agreed rules. The integrity of the file contents will be maintained without loss of information during transmission between the sending and receiving applications.

Shared Database: shares data with multiple applications. One factor that facilitates the use of database sharing is the generalized use of relational databases based on Structured Query Language (SQL). Virtually all application development platforms can work with SQL, often with reasonably sophisticated tools. By sharing the same database, all applications begin to deal with data that is reconcilable and consistent. On the other hand, there is the possibility that the database becomes a bottleneck for performance.

Remote Procedure Call: file transfer styles and the shared database are primarily focused on data integration. The so-called remote procedure style looks for process inte-

gration, which occurs when an application invokes procedures from another application. Basically, an A application exposes features that can be accessed remotely by another application B. Each application can change the format of its internal data without affecting any other application. In other words, application A communicates through its stub interface with the skeleton interface provided by application B, making a procedure call.

Messages: applications use communication channels to transfer data and events to share functionality immediately, reliably and asynchronously. An application publishes a message on a message channel to other applications that have access to that channel. The exchange of messages between applications occurs asynchronously [6]. In this context, the ability to transform messages allows applications to be decoupled from one another [6].

2.3. Cat Swarm Optmization

Heuristic optimization methods have, in many cases, been motivated by natural phenomena such as Simulated Annealing, Genetic Algorithm, Swarm Algorithms, Ant Colony Optimization. According to De Castro [15], swarm intelligence is also considered a branch of the computational approach known as Natural Computing. Such an approach is the computational version of the process of extracting ideas from nature to develop computational systems. The best-known techniques of swarm intelligence are Ant Colony Optimization (ANT), Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC) and Cat Swarm Optimization (CSO).

Cat Swarm Optimization (CSO) is one of the newly established high-performance computational techniques introduced in 2007 by Chu et al. [16]. The CSO is inspired by the natural behaviour of the cats, in which two main behaviours are modelled: the search mode and the tracking mode [17, 18]. This technique uses an analogy with the behaviour of cats. Thus apparently resting time is more significant than their hunting time; however, senses are all on alert. With slow and calculated movements, the cat places itself in a search mode state. However, when pursuing targets, it moves at high speed, converging on the solution. At this new time, the cat is in a tracking mode state [19]. CCSO makes use of this behaviour of cats to search for more solution spaces. We used an initial population of cats that are randomly divided into search and tracking modes according to the ratio of the defined mixing factor [20].

2.3.1. Search Mode

In the CSO technique, the search mode aims to model the behaviour of the cats during a rest period, but they are always in a state of alert in the dimensions of the environment to execute the next movement. Four key factors in search mode are applied: search memory pool (SMP), SRD search, count dimension for change (CDC), and own search position consideration (SPC). The SMP is used to set the memory to fetch size of each cat, which indicates any cat position.

Figure 3 illustrates the case where a cat has three dimensions. If the CDC is set to 100%, the copy distribution will be ball-shaped. Thus, the possible candidates to be surveyed around the cat are eight vertices of a cube. If the CDC is set to 66%, the copy distribution will be four vertices of 3 flat surfaces that have the original cat in the centre. In general, a higher value of CDC gives many possible candidates, generating many mutations.

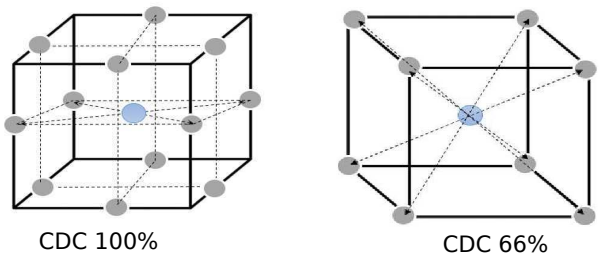

Figure 3. Configuration mode of CDC

All of these factors play essential roles in search mode. The SPC is a boolean expression variable and indicates whether the point where the cat is already standing will be one of the candidate points for the next move. The SPC cannot influence the value of the SMP [20]. This mode has the following steps:

1. Add j copies of the cat's current position, where $j = SMP$. If the value of the SPC is true, let j = (SMP - 1), then hold the current position as one of the candidates;
2. For each copy, according to the CDC, more or less SRD percentages are randomly added, the generated values have replaced the old ones;
3. Calculate the fitness values (FS) of all candidate points;
4. If all FS are not precisely equal, calculate the probability of selection of each candidate point by Equation (1), otherwise set the whole probability of selecting each candidate point to 1;
5. Randomly select the point to change to the candidate points and replace the cat's position.

$$Pi = \frac{FSi - FSb}{FSmax - FSmin}, onde\ 0 < i < j. \tag{1}$$

The purpose of the fitness function is to find the minimum solution, $FS_{\text{b}} = FS_{\text{max}}$, otherwise $FS_{\text{b}} = FS_{\text{min}}$.

We describe below the pseudo-code of the search mode of the algorithm.

Algorithm 1: Search Mode Algorithm

```
begin
    for each agent-cat do
        copies j = SMP;
        if SPC = true then
            Save copies;
        end
    end
    for each copy do
        add randomly (ou sub) SRD;
    end
    for each copy do
        calculate the value considering the fitness FS_i;
    end
    if all values of the fitness function are not equal to each other then
        calculate P_i;
    end
    if FS_i are equals then
        P_i = 1;
        FS_b = FS_min;
        And replace cat-agent with its copy;
    end
end
```

2.3.2. Tracking Mode

The tracking mode shapes the cat's case in the crawling state. Once a cat enters the tracking mode, it depends on its speeds for each dimension. The tracking mode comes from the rapid movements of a cat. This mode corresponds to a global survey.

In tracking mode, the cat has a velocity and position toward the cat that is in the best overall position $(gbest)$. The velocity and position are updated according to Equation 2 e 3.

$$V_i(t+1) = \omega * V_i + acc * round * (gbest) - P_i(t)) \tag{2}$$

$$P_i(t+1) = P_i(t) + V_i(t) \tag{3}$$

Where ω is the weight of inertia; acc is the acceleration coefficient, and a rand is a random number evenly distributed in the range between 0 and 1. The tracking mode algorithm is represented in the pseudo-code below:

Algorithm 2: Tracking Mode Algorithm

```
N cats in SRD randomly (velocity vector = zero) ;
Generate a flag for each cat ;
begin
    for k number of iterations do
        Calculate pbest (best position) ;
        Move each cat considering its flag ;
        else
            flag = 0
        end
        Run Seach Modea;
        Run Tracking Mode;
        Redistribute the flags ;
    end
end
```

2.4. Combination of Search and Trace Modes

In the basic description of cat flock optimization, it is observed that the CSO has two secondary modes, defined as search mode and tracking mode. To combine these two modes in the algorithm, we define a mixture index MR, which determines the union of the search mode with mode tracking [20]. Most of the time is spent in the search mode, and MR receives a minimal value. The process of combining CSO search and trace states is described below.

1. Create the number of N cats in the process;
2. The cats are randomly positioned in the space of the M-dimensional solution and values are randomly given, which are within the range of the maximum velocity, for the velocities of each cat. Then, randomly select the number of cats and set them in the tracking mode according to the MR, while the others are in the search mode;
3. The value of the aptitude of each cat is evaluated by applying the positions of cats in the aptitude function that represents the criteria of the objectives, and the position of the cats in the memory is maintained. Note that it is necessary to remember the best position of cats (gbest), because it represents the best solution so far;
4. Move the cats according to their flags; if the cat k is in the search mode, the cat k applies the process of the search mode; otherwise, the process of the tracking mode applies;
5. The number of cats is again selected and set to the tracking mode according to the MR, and then the other cats are set in the search mode;
6. The end condition is checked, if satisfied, the program is finalized, otherwise step 3 is repeated for step 5.

3. Related Work

With the systematic review of the literature, we identify studies that seek to provide strategies for the distribution of resources, aiming to achieve the smallest makespan possible through the optimization of the flow of tasks. We also identify work that proposes strategies that seek to minimize the makespan that is not in the scope of the application integration area (EAI) but use some optimization techniques such as Cat Swarm Optimization, Particle Swarm Optimization, and Artificial Bee Colony. Table 1 represents the approach adopted concerning the study presented in this chapter. Garg and Singh [21] addressed adaptive workflow scaling to minimize makespan. Udomkasemsub et al. [22] proposed a workflow scheduling framework using an Artificial Bee Colony considering multiple goals, such as makespan and cost. Wu et al. [23] propose a revised optimization of Discrete Particle Swarms to schedule workflow applications in the cloud. The authors consider both the cost of transmission and the cost of communication, in order to minimize the impact and cost (makespan) of the workflow.

Liu et al. [24] presented a Cost-Time-Committed scheduling algorithm to minimize the makespan and cost of workflows. The Cost-Time-Commitment scheduling algorithm considers the characteristics of cloud computing to accommodate workflows that are constrained by the intensive cost of instances committing time and cost of execution with user input activated in real time. Chirkin et al. [25] proposed an algorithm that highlights common problems in estimating the workflow execution time, as well as its execution time. The authors have developed an algorithm that considers the complexity and stochastic aspects of workflow components as well as their execution time.

Abdi et al. [26] presented three heuristic approaches for scheduling tasks in the cloud environment, which were compared to each other. These approaches are the Algorithm Optimization of Particle Swarm, the genetic algorithm and also the PSO algorithm modified for efficient scheduling of tasks. In all of these three algorithms, the goal was to generate an optimal timeline to minimize task completion time. Sellaro et al. [27] proposes a task scheduling algorithm that assigns the tasks to the threads, considering the waiting time in the ready task queue and the computational complexity of each task that optimizes the total execution time of a message in the integration process.

Singh et al. [28] review the use of metaheuristic techniques to schedule tasks in cloud computing. To schedule tasks in cloud computing, presenting the taxonomy and comparative review on these algorithms. The methodical analysis of task scheduling in cloud and grid computing is presented on the basis of swarm intelligence and bio-inspired techniques. Although it does not explicitly consider makespan, scheduling tasks leaves it implied.

Bilgaiyan et al. [29] experiment with the CSO algorithm using a hypothetical workflow and compare the results of workflow planning with the existing PSO algorithm, registering a better cost of time. Singh and Singh [30] proposed a workflow scheduling algorithm that seeks to minimize time-constraints defined by the user. Kumar and Ravichandran [31] present a priority-based algorithm whose goal is to maximize the use and cost of resources and reduce makespan. The framework for modelling and simulation of cloud computing infrastructures and services known as CloudSim.

The protocol defined for the bibliographic review has, among other things, a sequence of divergent points of the proposal presented in this chapter, they are:

1. Restriction to the EAI field of study;
2. Pipes-and-Filters structure;
3. Configuration model in the execution of integration processes;
4. Restrict the mount of computational resources (threads).

Authors	Schedule	Workflow	Cloud Computing	Techniques
Garg e Singh [21]	x			
Udomkasemsub et al. [22]		x		
Wu et al. [23]		x	x	
Chirkin et al. [25]		x		
Sellaro et al. [27]	x			PSO
Singh et al. [30]	x		x	
Bilgaiyan et al. [29]		x		PSO/CSO
Singh and Singh [28]	x	x	x	ABC
Kumar et al. [31]	x		x	

Table 1. Comparative of related work

4. Proposal based on Cat Swarm Optimization

Platforms engines perform the tasks in message processing. These engines have available threads, being able to have different numbers of threads, where a thread is the basic unit of processing. The execution engines can be based on two models for the execution of the integration solution: a global pool of threads and local pools of threads. The global thread pool model consists of a single thread pool to perform all the tasks of the integration solution. In the local thread pools model, each task will have exclusively a thread pool dedicated to its execution. In the integration solution tasks are independent, connected by channels called slots. Slots are responsible for transferring messages between tasks. Tasks consume a certain amount of time to process messages.

When a high message rate enters the integration solution, as the tasks have their own message execution time and a subsequent task has a longer execution time than its predecessor, this event will result in the accumulation of messages waiting to be processed in the slot. Moreover, this will imply in the makespan of any integration solution, making it slower. In the scenario where the execution engine is based on the local thread pools model, it is expected that the amount of threads in each pool will be enough to process the messages arriving at the task slots.

However, relating the ideal number of threads in the local pool to the time-out of the messages in the slots is not easy, it requires an optimal distribution of threads in the local pools that minimize the makespan of the integration solution. The distribution of the

optimal number of threads in each pool can be achieved using algorithms, mathematical modeling, heuristics and metaheuristics [32, 33]. Figure 4 represents the configuration of an integration solution based on local thread pools.

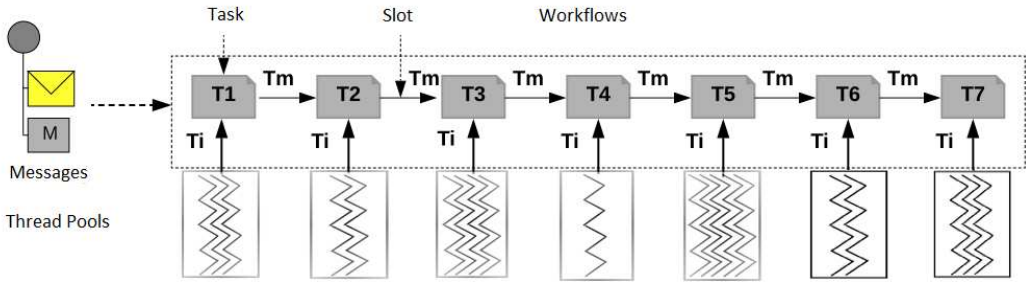

Figure 4. Integration solution configured with local pools

4.1. Problem Formulation

Performing tasks in an integration solution are complex and critical work [29]. The amount of data produced by software applications included in an integration solution is relatively large, so the processing time tends to be relatively high. It is considered that these data are processed in a task t_1, then transferred to the successor task t_2, through the slots, thus following the flow of the integration solution.

The integration solution can be represented in the form of a directed acyclic graph, whose nomenclature is given $DAG = (Z, H)$, where Z is defined as the set of n tasks t = $\{t_1, t_2, ... t_n\}$ and H is defined as directional edges representing the dependency between two tasks. Assume that a set of processing resources is represented by r = $\{r_1, r_2,..., r_n\}$. We are assuming that each processing resource r_i (where $i \in 1...n$) has its storage, denoted by Dr = $\{Dr_1, Dr_2, Dr_3, Dr_4..., Dr_n\}$, and that all processing resources are located in P different locations.

The makespan can be obtained by summing the execution time of each task in each resource and by summing the data transfer time between a task and its dependent successor task [29]. Figure 5 represents a workflow, modelled in the acyclic format, containing seven tasks, where d represents the time spent in the transfer of data T_m between tasks t_i and its successor t_j with a respective thread pool. It is assumed that T_m can be monitored and measured.

Therefore, the makespan of the entire process flow T_t (r_i) is the sum of the time incurred from the execution of the task T_i for the resources r_i and the sum of the transmission times T_m between a set of dependent tasks. T_t is calculated by Equation 1:

$$TT_p = \sum T_i + \sum T_m \tag{4}$$

Thus, the objective function described below can be expressed by minimizing the makespan T_t of the integration solution, with complexity of its implementation, in order not to exceed the total available resources (T_R = |R| = N_t), that is, to respect the parameter of constraint of an N_t number of threads. The formulation is represented by Equation 2:

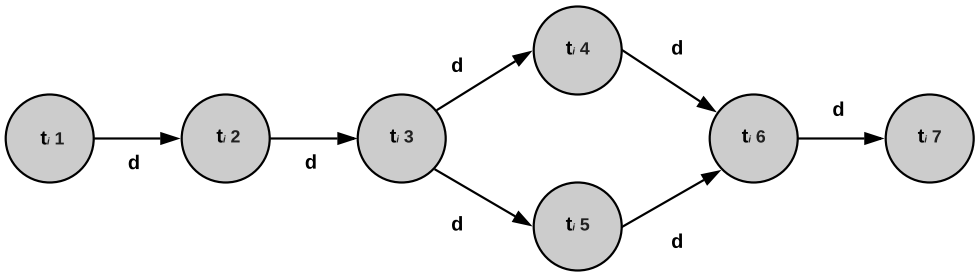

Figure 5. DAG representation of integration solution flow

$$Minimize\,(TT_p) \tag{5}$$

Subject to: $T_R \leq N_t$.

We assume that each task has a unit of time U_t of execution related to the complexity of the work performed by it. We also assume that in this flow, there are tasks with different units of time. Tasks with relatively small time unit have processed the messages in less time, while tasks with relatively large time unit (U_t) process the same message rate. The accumulation or predominance of messages in the task slots makes the execution of the integration solution slow. Then a message (M_1) would take a considerably considerable time to be processed from start to end of the flow. Therefore, minimizing the time a message waits in the slot to be processed will also minimize the makespan of the integration solution.

4.2. Algorithm Proposed

In order to transform the CSO problem into an algorithm, it is necessary to define the particle and its dimensions. In the approach adopted in this work, a particle represents workflow and its tasks, whereas the particle dimension represents the number of tasks in the workflow. The dimension of a particle serves to locate its position in the environment, defining the system of positions represented by flags. In the solution example presented, the particle size is the location, and a system specifies its position with two flags, to which values of 0 and 1 are assigned.

We consider that each task has the same number of thread pools with a single assignment to perform the task. Thus the location value is represented by the flag, which is in the range of 0 to 1. In the positioning system, the particle's moving space has a range of up to 1, with 1 being the maximum number of available thread pools. The value of a flag at the position of a particle corresponds to the number of thread pools and represents the computational resource assigned to a task defined by that particular flag. Thus, the position of the particle corresponds to the respective task incurred by the computational resource.

In the formulation of the problem, we consider the representation of $DAG = (Z, H)$ for an integration solution flow, with Z sets of nodes and H a set of edges. For the proposed

problem, the nodes represent the tasks, and each edge H represents the time spent for data transmission between every two tasks of Z. Then T_i and T_j are two tasks of Z.

Besides three operators were defined: position, speed and flag. The position is defined by Equation 4, the solution is obtained through the thread distribution; the velocity is the set of pairs (T_i, T_j), , which represents the permutation to be applied for the distribution of a pair number (T_i, T_j) and the velocity is defined by Equation 2, the flag allows us to determine if each particle is in search mode or in crawl mode. The goal is to find each particle to which it has the best aptitude, namely, the time of execution of the particle in the best position.

The objective function characterizes the object of the distribution problem since it is used to indicate the degree to which the solution is optimal or close to optimal. In the proposed approach, the objective function is minimized and its value will be the T_t makespan contained in the distribution, associated with the position of the particle. Algorithm 3 presents the pseudo-code of the proposal through the set of R resources to be allocated and the set of threads directed to the tasks.

4.3. Structure of the CSO Algorithm

To generate the solutions it is necessary to use the nine tasks that make up the integration solution, T= {t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9}, so that the quantity of tasks is equal to the number of N_{pools}. Another critical factor is the number of threads N_t to not exceed the total available resources {T_R = |R| = N_t}, according to the constraint $T_R \leq N_t$. The $Minimize(\ TT_p)$ objective function calculates the makespan according to the TX_M message rate that arrives at the input port of the integration solution flow, in particular, the case study addressed. The makespan relation and the thread distribution in the pools configure a set of solutions stored in vectors. After the calculation is performed through interactions, it searches for and classifies the best distributions based on makespan. In this part of the algorithm, we use the standard CSO parameters: MR (number of cats that hunt), SMP (searching for memory pool), SPC (self-positioning, default value is $False$), CDC (default value is 0.1), W (inertia constant, default value is 0.1), and C (default value is 1), SRD (searching for range of selected dimension, constant acceleration (the default value is 1.05).These parameters are elements of fundamental importance for local and global search modes, according to the probabilistic factor of MR mixing. The complete interface of the proposed can be observed in Algorithm 3.

4.4. Encoding CSO Algorithm

The proposed algorithm was implemented through the GNU Octave version 4.4.1. This software primarily works with a rectangular numeric object-array type, which can accept even complex values. All variables in Octave represent matrices. The three coding parts of the CSO algorithm with Matlab programming language compiled in Octave.

Algorithm 3: Algorithm based on Cat Swarm Optimization

begin
  Number of **K** population of solutions;
  Message rate TX_M;
  Number of Tasks T = {T_1, T_2, T_n} = Number os Pools N_{pools};
end
Objective Function $Minimize(\ TT_p\ )$;
Better threads Distribution MD_T;
begin
  Generate **K** population of solutions **end**
  Start threads distribution R = {R_1, R_2,..., R_n}) in pools located at K solutions
  Start mesage rate TX_M
  Calculate TT_p of each K solutions, such as N_{pools} e TX_M
  Store K solutions with MD_T and smaller than TT_p
  Based on MR indicate randomly K solutions with the best MD_T and smaller than TT_p in: search and tracking modes
  Classify (to rank) K solutions with the best MD_T and smaller than TT_p $\rightarrow$ (**Pbest1**)
  Store K solutions with the best rating (**Pbest1**)
  K Solution withe the best MD_T and smaller than TT_p in search mode Set search mode considering MR;
  Set tracking mode considering MR;
  Classify (to rank) K solutions with the best MD_T and smallher TT_p in both modes $\rightarrow$ (**Pbest2**)
  Store K solution with the best rank $\rightarrow$ (**Pbest2**)
  Compare (**Pbest1**) and (**Pbest2**)
  Choose K solution with the best MD_T and smaller TT_p $\rightarrow$ (**Pbest3**)
  Pbest3 is the best K solutions concluded
  Show $\rightarrow$ (**Pbest3**)
  Based on MR distributed randomly K in search or trancking mode.

5. Experimental Results

In this section, we present the experiments performed to evaluate the performance of the execution engine. We use the proposed algorithm based on Cat Swarm Optimization to simulate the actual behaviour of the execution engine.

According to Franca and Travassos [34], the uncertainties, time and costs of the experiments can be reduced through simulation-based studies because they are performed in virtual environments. Our simulation compares the results of the average total processing time of a message in the integration processing flow represented in Table 5.. We use the proposed model with a random set of threads to minimize processing time.

The program receives the following parameters:

Task	Caption	T-Exec	T-Slot	T-Send	T-Rec	T-Proc
T1	Splitter	0.531	2	-	-	2.531
T2	Normalizer	0.303	1	-	-	1.030
T3	Content Enricher	0.005	-	2	2	4.005
T3*	Content Enricher	0.005	1	-	-	1.005
T4	Content Filter	0.003	1	-	-	1.003
T5	Content Enricher	0.005	-	2	2	4.005
T5*	Content Enricher	0.005	1	-	-	1.005
T6	Recipient List	0.531	1	-	-	1.531
T7	Message Filter	0.003	1	-	-	1.003
T8	Message Translator	0.001	1	-	-	1.001
T9	Message Translator	0.001	1	-	-	1.001

Table 2. Task processing time

- Parameter 1: Number of solutions to be tested;
- Parameter 2: Total number of threads to be distributed in pools;
- Parameter 3: Number of messages to be processed;
- Parameter 4: Task processing times;
- Parameter 5: Number of Cat (candidate cats) for the search.

The protocol applied to perform the experiments was developed based on the works of Jedlitschka e Pfahl [35], Wohlin et al. [36] e Basili et al. [37], which established procedures for controlled experiments in the field of software engineering studies. The authors propose the development of an experimental component based on a unified standard. These guidelines serve as a starting point for further discussion of the quality improvement of empirical studies and controlled experiments. This protocol is detailed in the following sections to present the experiment that was performed.

5.1. Case study

We use a real software ecosystem as a case study, which is responsible for managing the collection of the Urban Land Tax in the municipality of Ijuí, located in the state of Rio Grande do Sul, Brazil. The ecosystem calculates the amount of tax due, relative to each of the properties registered. Files generated from this integration are converted to PDF format. The final document, in the form of a ticket for payment, must be sent by mail and/or e-mail to the owner of the property.

The ecosystem consists of seven applications: Application of accounting, application of IPTU, ARCetil, ARCit, Database, E-mail Server and Print Server. Each of these applications is described below based on Kühne [38].

1. Application of accounting: this application is made in a direct way in which the taxpayer seeks information in the municipal unit;

2. Application of IPTU: this application is responsible for requesting the creation of tax tickets for each of the properties registered in the city's database. This action is performed manually by the operator around the last month of each year so that payment is made the following year;

3. ARCetil: calculates the amount due from the Urban Land Tax and creates the ticket in PDF file format;

4. ARCit: allows the taxpayer to request, through the internet network, the second leg of the Urban Land Tax ticket and/or updated route of the tax due;

5. Database: Microsoft SQL Server 2014 database that includes data on taxpayers, which are used by the ARCetil application in the calculation of the Urban Land Tax relative to the property;

6. E-mail server: accepts the sending requests of taxpayers who have registered an e-mail address and sends the file in PDF format to the address;

7. Print server: PDF files are printed and mailed to each taxpayer.

5.2. Abstract Model of the Integration Solution

The following is the role that each of the patterns used in the model has in the integration flow. Figure 6 represents the conceptual model designed with the integration patterns proposed by Hohpe and Woolf [6], to solve the problem of integration of the Urban Land and Land Tax (IPTU) collection solution.

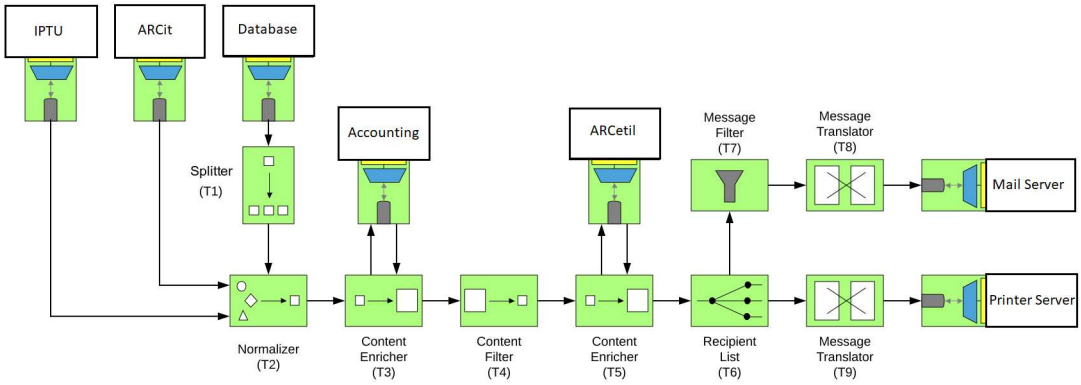

Figure 6. Solution modeling in EAI standards

1. Splitter: breaks the message requesting the generation of the tickets, relative to all existing registrations, so that each outgoing message corresponds to single property registration, and forwards all these outgoing messages to the standard (2);

2. Normalizer: transforms inbound messages from the three applications, which have different formats, to the same canonical format that is used in the rest of the solution and forwards this transformed message to the standard (3);

3. Content Enricher $(T3)$: requests information from each registration to the city's database, enriches the message with the data obtained and forwards this enriched message to the standard (4);

4. Content Filter : eliminates unwanted data from the real-estate return message and forwards this message to the default (5);

5. Content Enricher $(T5)$: asks the ARCetil application to perform the IPTU calculations of the property referenced in the message, enriches the message with a PDF file corresponding to the generated ticket and forwards this enriched message to the default (6);

6. RecipientList: receives the message and makes a copy, forwarding it to the defaults (7) and (9), so that the PDF file can be sent by e-mail to the contributor (segment of the solution that contains the default (7)) , and printed to be sent by mail (segment of the solution containing the standard (9));

7. Message Filter: filters the messages so that only those related to the contributors who have a registered e-mail address are forwarded to the default (8), discarding the others;

8. Message Translator $T8$: transforms the content of the received message into an e-mail sending requirement compatible with the e-mail server, and forwards this transformed message to that server so that the PDF file is sent to the e-mail -mail;

9. Message Translator $T9$: transforms the content of the received message into a print request compatible with the print server and forwards that transformed message to that server so that the PDF file is printed and sent to the taxpayer by mail.

The Urban Land Tax integration solution formulated in acyclic DAG graph represents the directed flow, according to Figure 7. It represents the arrangement of the times incurred from the execution of the task T_i and the transmission times T_m between subsequent tasks.

Where: TA=$\{TA1, TA2, TA3, TA4, TA5, TA6, TA7, TA8, TA9\}$ represents the set of tasks and times incurred in the execution of task T_i; and the set a=$\{a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15, a16, a17\}$ present directed edges and transmission times T_m.

5.3. Experimental Environment

The implementation and execution of the proposed algorithm for the simulation were performed in a computer with Mac OS Mojave Operating System version 10.14.1; Intel (R) Core (TM) i5-5200U, 2.9 GHz CPU; 2 colours; 256KB Cache L2 (per Core); 4 MB L3 cache; 8 GB of RAM. GNU Octave version 4.4.1 was installed to encode and execute the simulation. In order to process the descriptive statistics, the software Genes [16], version 2015.5.0, was used to calculate the Anova to use the Scott-Knott method.

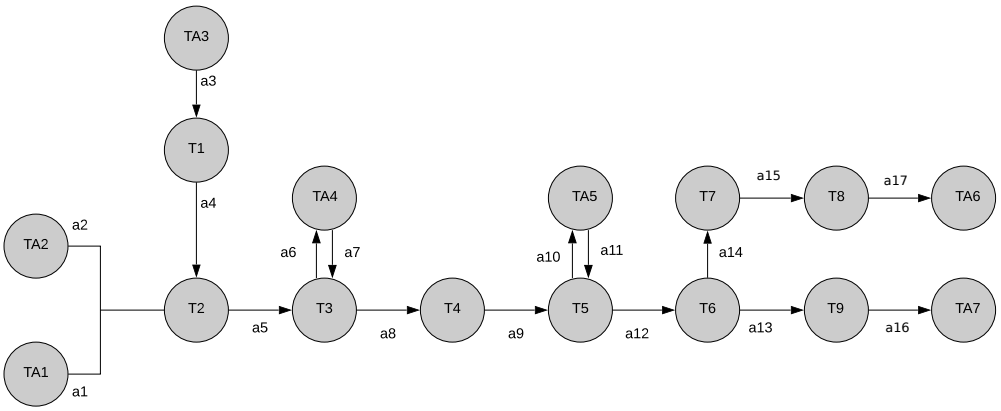

Figure 7. Case study graph

5.3.1. Simulation Scenarios

The ideal distribution in the thread pools depends on the amount of resource available by looking at the number of thread constraints as well as the message rate. For a simulation of message processing in the case study solution flow, 200 different scenarios were used, using ten configuration sets and ten constraints on the number of threads described in 8. We consider that there is one thread per pool, and the total number of constraint threads must be greater than or equal to the number of tasks.

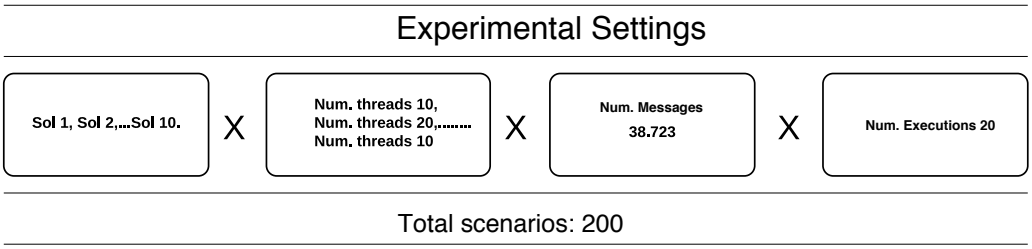

Figure 8. Experimental scenarios

5.3.2. Execution and Data Collection

The execution of the algorithm was performed using the flow of the integration solution represented in the graph model shown in Figure 7. This experiment is classified in the literature as a termination simulation. In this type of simulation, the outputs are a function of the initial conditions and are generally analyzed statistically by the method of the executions, in which several runs between 20 and 30 is acceptable to obtain a population [39]. In this experiment, the execution of the algorithm was repeated 20 times. We decided to limit the allowable number of population solutions to 10 thread pool configurations, generated by the first part of Algorithm 3. Moreover, we used ten constraints on the number of 10 sets of

distributions interspersed in 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

The algorithm was executed with the number of possible solutions, the number of messages of the integration solution and a vector to represent the processing times of tasks. At each interval, the number of threads was varied. The algorithm was executed 20 times for each set with the number of threads. In each execution, we collect the following values: smaller makespan best distribution threads in the pools and the execution time related to the best distribution threads in the pool performed by the proposed algorithm. The standard deviation was calculated from the average total processing times measured in the experiment.

We used the statistical technique of ANOVA variance analysis for the different execution times, among the variations found in a set of results. The Results are derived from random factors called error and are influenced by the total number of segments. According to Georges et al. [40], the statistical theory is indicated for the analysis of data from performance experiments, since statistical reasoning is an adequate resource to deal with non-determinism in computational systems such as time systems of execution [41]. In the experiment, we verified that the same execution time value was repeated for different distributions in each set of threads. This fact could lead to redundancy, but this was amplified through ANOVA analysis of variance using the Scott-Knott technique. The Scott-Knott technique is considered the most demanding test because it considers significant differences between the alternatives; it is adopted in the literature in experiments with performance due to its ease of application.

5.4. Results and discussion

The results for each thread distribution in the local pools and the total message processing times in the integration solution tasks (Figure 6) are presented in Figure 9. In the graphs, the $x - axis$ represents the order of execution. The $y - axis$ represents the average total processing time makespan represented by TT_p in seconds (s). Line charts present the best distributions of thread sets an average total time processing for each of the quantities of the total number of threads allocated.

A vector stores the distribution of threads in the pool, integral elements are the number of threads in each thread pool, the index of the vector corresponds to the order of threads in the local pool for tasks in an execution flow of the integration solution. The highlighted point in the curves presents the lowest makespan TT_P, corresponding to the distribution of in the thread pool. The standard deviation of the mean of total processing time, are presented in Table 3, summarizes the standard deviations and averages of the algorithm.

A scatter plot was generated to analyze the total time values obtained from the processing with the best thread pool distribution relative to the treatment of $38,723$ messages. This graph, shown in Figure 10, compares the results of the lowest total average processing time for each of the numbers of threads sets. The $x - axis$ represents the constraint of the total number of threads, and the $y - axis$ represents the values of the total average minimum processing time.

The graph also demonstrates a statistical disposition line of TT_p, represented by a line, which corresponds to a polynomial equation, according to Equation 6:

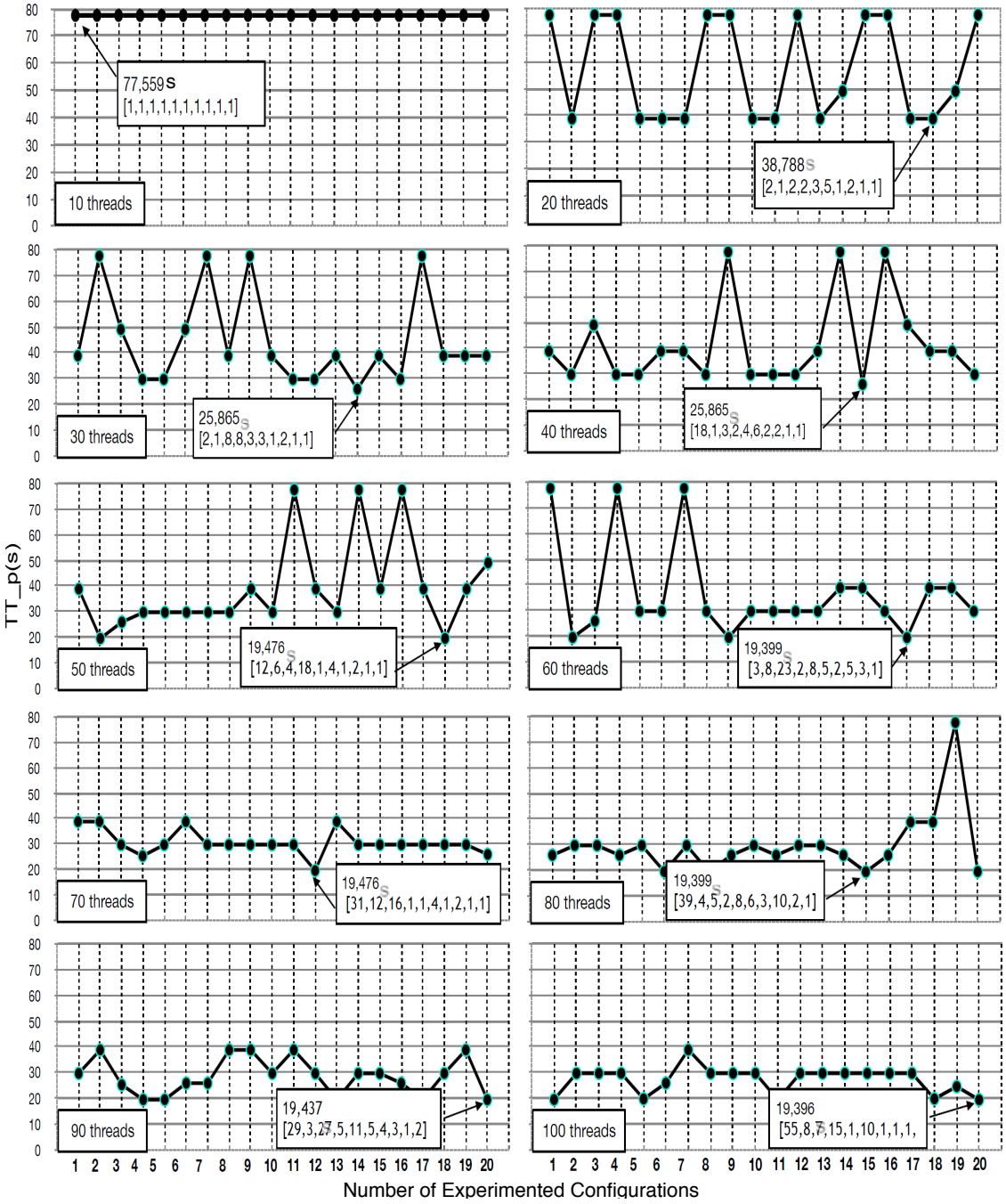

Figure 9. Experimental Distributions

Threads	Standard Deviation (TT_p)
10	77.559
20	38.788
30	25.865
40	25.865
50	19.476
60	19.399
70	19.476
80	19.399
90	19.437
100	19.399

Table 3. Standard deviation and averages of TT_p

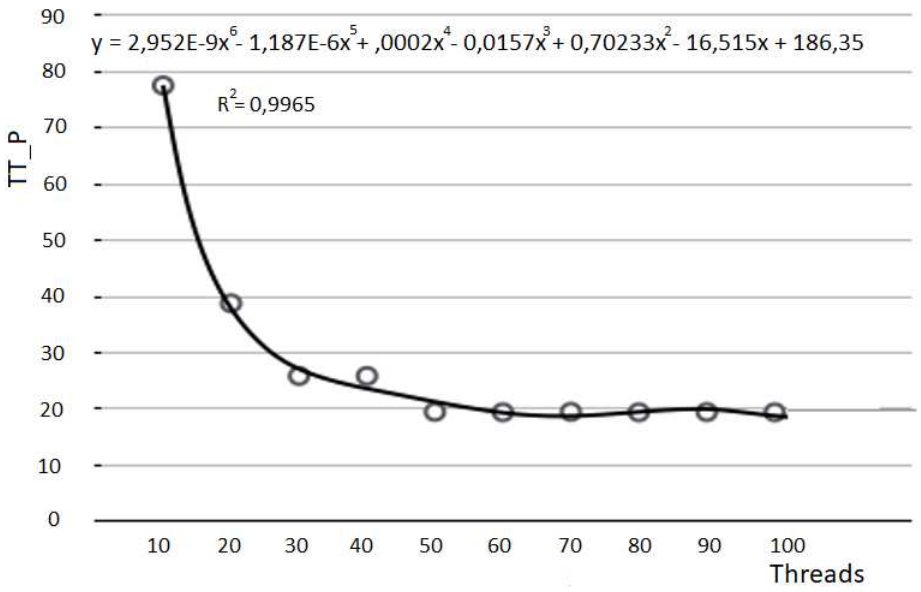

Figure 10. Results of the lowest total average processing time for each of the numbers of threads

$$TT_p = 2,952(N_t)^6 - 1,187(N_t)^5 + 0,0002(N_t)^4 \\ -0,0157(N_t)^3 + 0.7033(N_t)^2 - 16,515 + 186,32 \quad (6)$$

In this equation, the trend line is represented by a polynomial equation that treats the behaviour of the makespan as a function of the total number of threads and their respective value of R^2 = 0,9965. The variable N_t epresents the number of threads, and TT_p is the value of the lowest makespan.

Statistical techniques were used to verify the influence of the total number of threads used in makespan. The independent variable is the total number of threads, and the dependent variable is TT_p. Table 4 presents the analysis of variance of the dependent variable. The total of results is the number of executions multiplied by the number of possible values for the total number of segments. The degree of freedom of the total is its value by subtracting 1.

Source of variation (FV)	Degree of relaxation (GL)	Mean Square (QM)
Threads available	9	4761,43*
Error	190	204,09
Total	199	
Média Geral		41,58
CV(%)		34,35

*significant at a level of 5 % confidence

Table 4. Analysis of Variance

The comparison of means by the Scott Knott technique of the dependent variable is present in Table 5. The constraints on the number of threads in the first column correspond to the TT_p in the second column for 20 runs, and the Scott-Knott technique group is in the third column. This technique of constraint groups on the number of threads about the execution time indicates that the sets of threads inserted in a group are the same for the processing of messages.

In the analysis performed, there are three groups: a, b and c. O grupo a representa as restrições sobre o número de 10 threads. Group a represents constraints on the number of 10 threads. Group b represents the constraints on the number of 20 threads. Group c exposes constraints on the number of 30, 40, 50, 60 threads. The group d presents the constraints on the number of 70, 80, 90, 100 threads.

The minimum number of threads to distribute among the 10 thread sets is equal to 10, one thread for each pool. In this case, the algorithm can only provide distribution, and the minimum processing has pan was 77.55 seconds. When the number of threads to be distributed is greater than 10, the algorithm can provide the different amounts of threads distributed in the pools, resulting in different values for the makespan. With the number of threads equal to 20 and 30 threads, the lowest TT_p value was $25,865$ seconds for both.

A reduction of 51.69 seconds concerning the processing time with the distribution of 10 threads is observed. With the number of threads equal to 40, the shortest TT_p execution

Number of threads available	Average of TT_p	Groups
10	77.5590	a
20	57.2587	b
30	44.6378	c
40	41.3299	c
50	39.3434	c
60	36.9474	c
70	30.5661	d
80	29.7867	d
90	28.6042	d
100	27.1410	d

Confidence Level 5% by the Scott test & Knoot

Table 5. Scott-Knoot Statistical Analysis

time was 25.86 seconds, with a reduction of 11.92 seconds concerning the number of 20 threads, but equal to the number of 30 threads. With the number of threads equal to 50, the shortest execution time was 19.47 seconds, a reduction of 6.40 seconds concerning the restriction number of 40 threads is noticed.

With sets of thread numbers equal to 60, 80, and 100, it is observed that the lowest TT_p is the same for all sets, which is 19.39 seconds. Another identified observation is a reduction of only 0.03 seconds concerning the thread sets 70 and 90, respectively. In the relation of the set of 90 threads and the set of 100 threads, we have the smaller TT_p de $19,47$ of 19.47 and 19.39 seconds respectively, reduction of only 0.03 seconds.

It is important to emphasize that the value of the smaller makespan has decreased with the addition of threads up to a limit, so this lower mean total time stabilizes and tends to no more prolonged decrease. The Scott Knott averages comparison test demonstrates that the TT_p value for the 100-thread distribution is 19.39 seconds. The most significant difference between averages occurs for 10 and 100 threads, being 58.16 seconds.

The present study is pertinent to be applied in other case studies. With different integration solutions, which receive different message rates. The study can still serve as a reference for software engineers, assisting in the distribution of pooled threads in conceptual execution models.

5.5. Threats to Validity

We analyzed the factors that may influence the results and the central interventions that occurred in the experiment. The execution of processes parallel to the process of execution of the optimization algorithm and that possibly will imply in the results of the realized experiments. Another threat is that running with another integration solution with another number of messages may present different results and with several possible solutions.

During the experiments, we tried to reduce the likelihood of influencing external processes to extract the maximum performance from the actual hardware. They include procedures such as disconnecting the machine from the Internet and avoid starting any other process during the execution of the experiment.

The results obtained are valid to compare with the execution of other integration solutions, with different numbers of messages and with different numbers of possible solutions, generated by the population algorithm, as long as it is maintained in an experimentation environment. However, the results obtained in this experiment may vary, so it is not possible to consider them a general standard.

For Wainer et al. [42], validation means comparing under the view of a condition, if the model behaves as a system in the real world on the same conditions, then the model is valid; otherwise it is not valid. To experiment, we followed procedures based on the integration solution of the case study.

We apply the proposed algorithm in the conceptual model of execution under the same conditions of the real world. The time obtained the execution of the algorithm with optimization technique, allowed to identify the shorter execution times. They demonstrate a direct relation as to the thread distribution of each local pool. After the collection of the metrics of execution of the algorithm, we tried to construct a representation of the analysis of variance using the statistical techniques of Scott Knott aiming to evaluate results. Statistical validation, in general, is used when one wants to measure mean, variance and degree of satisfaction of models of a universe through a sample that represents them in a statistically proven way.

6. Conclusions

This chapter introduced a method based on the Cat Swarm Optimization (CSO) technique to find the ideal distribution of threads in integration solutions. This approach can contribute to improve the performance of the execution of integration processes in runtime models and, consequently, increase the productivity of the system.

The algorithm was implemented and tested with a real-world integration process. The results of the executions showed improvements in the runtime model with the optimal distribution of threads in the pool. We verified that the statistical analysis utilizing the trend line described by Equation 6 allowed to find the behaviour of the makespan in the function of the total number of threads. However, this statistical technique showed some stability at the execution time, even with the increase in the number of threads. With this, we have verified that there is no significant reduction in the average total processing time over 70 threads. Therefore, it is possible to achieve an optimal distribution of threads in pools, and this distribution causes minimization of makespan. This experiment can be adopted for other scenarios, varying the message rate, different integration solution, number of threads and the number of solution populations. In future research, we intend to compare the CSO optimization technique with other metaheuristics in order to compare its execution results.

References

[1] Konstantinos Manikas. Revisiting software ecosystems research: A longitudinal literature study. *Journal of Systems and Software*, 117:84–103, 2016.

[2] David G Messerschmitt, Clemens Szyperski, et al. Software ecosystem: understanding an indispensable technology and industry. *MIT Press Books*, 1:424, 2005.

[3] Daniela L. Freire, Rafael Z. Frantz, Fabricia Roos-Frantz, and Sandro Sawicki. A methodology to rank enterprise application integration platforms from a performance perspective: an analytic hierarchy process-based approach. *Enterprise Information Systems*, (1):1–31, 2019.

[4] David S Linthicum. *Enterprise application integration*. Addison-Wesley Professional, 2000.

[5] Rafael Z Frantz, Rafael Corchuelo, and Fabricia Roos-Frantz. On the design of a maintainable software development kit to implement integration solutions. *Journal of Systems and Software*, 111:89–104, 2016.

[6] Gregor Hohpe and Bobby Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[7] John Dooley. *Software development and professional practice*. Apress, 2011.

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

[9] Claus Ibsen and Jonathan Anstey. *Camel in action*. Manning Publications Co., 2010.

[10] Craig Walls and Ryan Breidenbach. *Spring in action*. Dreamtech Press, 2005.

[11] Prabath Siriwardena. *Enterprise integration with WSO2 ESB*. Packt Publishing Ltd, 2013.

[12] David Dossot, John d'Emic, and Victor Romero. *Mule in action*. Manning Publications Co., 2014.

[13] Daniela L. Freire, Rafael Z. Frantz, Fabricia Roos-Frantz, and Sandro Sawicki. Survey on the run-time systems of enterprise application integration platforms focusing on performance. *Software: Practice and Experience*, 49(3):341–360, 2019.

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

[15] Leandro Nunes De Castro. *Fundamentals of natural computing: basic concepts, algorithms, and applications*. CRC Press, 2006.

[16] Shu-Chuan Chu, Pei-Wei Tsai, et al. Computational intelligence based on the behavior of cats. *International Journal of Innovative Computing, Information and Control*, 3(1):163–173, 2007.

[17] Lakshman Pappula and Debalina Ghosh. Linear antenna array synthesis using cat swarm optimization. *AEU-International Journal of Electronics and Communications*, 68(6):540–549, 2014.

[18] Neeraj Kanwar, Nikhil Gupta, KR Niazi, and Anil Swarnkar. Improved cat swarm optimization for simultaneous allocation of dstatcom and dgs in distribution systems. *Journal of Renewable Energy*, 2015, 2015.

[19] Reza Shojaee, Hamid Reza Faragardi, Sara Alaee, and Nasser Yazdani. A new cat swarm optimization based algorithm for reliability-oriented task allocation in distributed systems. In *Telecommunications (IST), 2012 Sixth International Symposium on*, pages 861–866, 2012.

[20] Pei-Wei Tsai, Jeng-Shyang Pan, Shyi-Ming Chen, Bin-Yih Liao, and Szu-Ping Hao. Parallel cat swarm optimization. In *Machine learning and cybernetics, 2008 international conference on*, volume 6, pages 3328–3333, 2008.

[21] Ritu Garg and Awadhesh Kumar Singh. Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology, an International Journal*, 18(2):256–269, 2015.

[22] Orachun Udomkasemsub, Li Xiaorong, and Tiranee Achalakul. A multiple-objective workflow scheduling framework for cloud data analytics. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pages 391–398, 2012.

[23] Zhangjun Wu, Zhiwei Ni, Lichuan Gu, and Xiao Liu. A revised discrete particle swarm optimization for cloud workflow scheduling. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 184–188, 2010.

[24] Ke Liu, Hai Jin, Jinjun Chen, Xiao Liu, Dong Yuan, and Yun Yang. A compromised-time-cost scheduling algorithm in swindew-c for instance-intensive cost-constrained workflows on a cloud computing platform. *The International Journal of High Performance Computing Applications*, 24(4):445–456, 2010.

[25] Artem M Chirkin, Adam SZ Belloum, Sergey V Kovalchuk, Marc X Makkes, Mikhail A Melnik, Alexander A Visheratin, and Denis A Nasonov. Execution time estimation for workflow scheduling. *Future Generation Computer Systems*, 75:376–387, 2017.

[26] Solmaz Abdi, Seyyed Ahmad Motamedi, and Saeed Sharifian. Task scheduling using modified pso algorithm in cloud computing environment. In *International conference on machine learning, electrical and mechanical engineering*, pages 8–9, 2014.

[27] Daniela L. Freire, Rafael Z. Frantz, and Fabricia Roos-Frantz. Towards optimal thread pool configuration for run-time systems of integration platforms. *International Journal of Computer Applications in Technology*, (in-press):1–18, 2019.

[28] Poonam Singh, Maitreyee Dutta, and Naveen Aggarwal. A review of task scheduling based on meta-heuristics approach in cloud computing. *Knowledge and Information Systems*, 52(1):1–51, 2017.

[29] Saurabh Bilgaiyan, Santwana Sagnika, and Madhabananda Das. Workflow scheduling in cloud computing environment using cat swarm optimization. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 680–685, 2014.

[30] Ranjit Singh and Sarbjeet Singh. Score based deadline constrained workflow scheduling algorithm for cloud systems. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, 3(6), 2013.

[31] B Kumar and T Ravichandran. Scheduling multiple workflow using de-de dodging algorithm and pbd algorithm in cloud: detailed study. *Int J Comput Electr Autom Control and Inf Eng*, 9(4):917–922, 2015.

[32] Leandro Ferro Luzia and Mauricio Chui Rodrigues. Study on metaheuristics. *Institute of Mathematics and Statistics, University of São Paulo (IME-USP). São Paulo*, 2009.

[33] Daniela L. Freire, Rafael Z. Frantz, Fabricia Roos-Frantz, and Sandro Sawicki. Optimization of the size of thread pool in runtime systems to enterprise application integration: a mathematical modelling approach. *Trends in Applied and Computational Mathematics*, (20):169–188, 2019.

[34] BB Nicolau de Franca and G Horta Travassos. Reporting guidelines for simulation-based studies in software engineering. *Systems Engineering and Computer Science Department*, 2012.

[35] Andreas Jedlitschka and Dietmar Pfahl. Reporting guidelines for controlled experiments in software engineering. In *International Symposium on Empirical Software Engineering, 2005.*, pages 10–pp, 2005.

[36] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[37] Barry Boehm, Hans Dieter Rombach, and Marvin V Zelkowitz. *Foundations of empirical software engineering: the legacy of Victor R. Basili*. Springer Science & Business Media, 2005.

[38] Ivan Eduardo Metz Kühne. *Guaraná dsl in the context of municipal public administration: a case study*. Undergraduate thesis, Unijui University, 2017.

[39] Robert G Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 Winter Simulation Conference*, pages 166–183, 2010.

[40] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous java performance evaluation. *ACM SIGPLAN Notices*, 42(10):57–76, 2007.

[41] Rafael Z Frantz, Rafael Corchuelo, and Jose L Arjona. An efficient orchestration engine for the cloud. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 711–716, 2011.

[42] Gabriel A Wainer. *Modeling and discrete-eventsimulation: a practitioner's approach*. CRC press, 2009.