Performance Evaluation of Thread Pool Configurations in the Run-time Systems of Integration Platforms

Daniela L. Freire · A. Mazzonetto · Rafael Z. Frantz · Fabricia Roos-Frantz · Sandro Sawicki · Vitor Basto-Fernandes

Abstract Companies use integration platforms to connect applications that make up their software ecosystem, which is comprised of local applications as well as cloud computing services. The run-time systems are the component of the integration platforms that have the most considerable influence on their performance. Our literature review has identified that most integration run-time systems adopt a global pool as configuration for threads. However, it is possible to configure local thread pools to increase the performance of run-time systems. This article brings a comparison between two configurations of thread pools simulating the execution of a real integration problem. The results show that the performance of the execution using the local pool configuration exceeds the performance using the global pool in high workload scenarios. These results were reviewed by rigorous statistical analysis.

Keywords Application integration · Thread pool · Optimisation · Multithread · Run-time system · Simulation

1 Introduction

With the advent of cloud computing, many applications, operating on-premises in enterprises, have been migrated and are now being offered as cloud

Daniela L. Freire
Department of Exact Sciences and Engineering, Unijuí University, Ijuí, RS, Brazil
Instituto Universitário de Lisboa (ISCTE-IUL)ISTAR-IUL, Lisbon, Portugal
Present address:
Institute of Mathematics and Computer Sciences University of Sao Paulo, Sao Carlos, SP, Brazil E-mail: danielalfreirer@icmc.usp.br

A. Mazzonetto, Rafael Z. Frantz, Fabricia Roos-Frantz, Sandro Sawicki
Department of Exact Sciences and Engineering, Unijuí University, Ijuí, RS, Brazil E-mail:

Vitor Basto-Fernandes Instituto Universitário de Lisboa (ISCTE-IUL)ISTAR-IUL, Lisbon, Portugal E-mail:

services. Software ecosystems, made up of these applications and services, have become even more heterogeneous, increasing the need for integration amongst them, so that they work in a synchronised manner and support business processes. However, many of these applications still need to be adapted to operate in the context of cloud computing, maintaining or improving the same performance, which they achieve by running locally [21, 29, 33]. Performance is a dominating user requirement in the software enterprise and is probably the most studied quality attribute in the software quality field [8].

The integration Platform as a Service (iPaaS) is an example of software provided as a cloud service. Adopting iPaaS lowers the maintenance and operations costs of local integration platforms and is, therefore, a good fit for small and mid-sized businesses that need to integrate the business processes without increasing their costs [10]. Integration platforms are specific tools that allow the design, execution and monitoring of integration processes. Such processes orchestrate applications and services such that data are synchronised and new functionalities are developed on top of what currently exists, with minimal impact [17]. Generally, integration platforms provide a domain-specific language, a development toolkit, a testing environment, a monitoring tool, and a run-time system. The domain-specific language allows the creating conceptual models for the integration process, whose level of abstraction facilities the understanding of the problem. The development toolkit consists of a set of tools for the implementation of the process, i.e., the conversion of the conceptual model into executable code. The testing environment allows performing tests in the entire integration process to mitigate or eliminate possible inadequacies in the implementation. The monitoring tool aims to monitor the operation of the integration process and detects errors in run-time. The run-time system provides all the support required to execute these integration processes. An integration process carries out a workflow made up of distinct atomic tasks that process data, encapsulated within the messages, which flow through the process. The tasks of the integration process are executed by threads, which are present in the run-time system grouped in thread pools.

Our literature review has found that most integration run-time systems adopt a single thread pool, which is usually efficient with low amounts of messages. However, when the number of messages increases, the total execution time of the processes also increases [40]. The addition of threads is the approach usually adopted, but this alternative increases the financial costs of the enterprises, especially in the context of cloud computing, whose billing system where value is proportional to the consumption of computational resources. Besides, there is a saturation point, from which the increase in the number of threads will not bring any benefit, and may even degrade the performance of the run-time system [37, 27, 30].

Recent research proposes an optimal configuration to local thread pools using Particle Swarm Optimisation (PSO) [18] to the execution of tasks of integration processes by run-time systems of integration platforms. However, this study did not compare the traditional configuration that uses a global

thread pool with the proposal configuration that uses local thread pools. Our article extends this previous work [18] to fill this lack and, thus, to help researchers and practitioners with arguments that allow a chose of the more adequate thread pools configuration model in different scenarios of message processing by integration platforms. We simulate the behaviour of a real-world integration process executed both global and local thread pools, submitted to high workloads. The results show that the local thread pools optimised provided a lower average total time of message processing than the global thread pool, been this difference directly propositional to workload. For that, we used rigorous statistical techniques to validate the results found.

The rest of this article is organised as follows: Section 2 discusses related work; Section 3 provides background information on the run-time system and the thread pool configuration models; Section 4 formulates the performance metric and objective function; Section 5 describes a study case; Section 6 reports our simulation; and, Section 7 presents our conclusions.

2 Related Work

In this section, we discuss related work that simulations and report experiences regarding performance analysis and thread pool configuration. It is possible to divide these works according to their goals, such as, performance of power transmitting systems, the performance of web-based software systems, load balancing and fault detection.

Braun and Krus [11] presented an automatic approach for parallel continuous-time simulation of tightly coupled power transmitting systems destined for industrial problems. They developed an automated algorithm for partitioning the distributed system models for multi-core processors with proper load balancing and another synchronisation algorithm for running several simulation threads concurrently. Ágnes Bogárdi-Mészöly and Rövid [9] proposed mathematical models, in the form of difference equations by subspace identification, to simulate the behaviour of thread pools and queued requests to predict the performance of web-based software systems. Pasha et al. [31] presented a simulation framework for code-level energy estimation. The framework has an instruction-level power estimator module that estimates the average power consumption of individual machine instructions simulated using gate-level netlists of the target processors. In addition, it has a high-level energy estimation module parses the assembly code written for a target processor and gives the estimated energy consumed while taking the inherent data and control dependencies into consideration. Ahmad et al. [1] proposed an approach for performance testing of web applications that place the worst path of sequences of user interactions, subject to a workload model that demands high resource utilisation on the system under test. They presented an exact and an approximate method for detecting the worst path in the workload model and analysed the performance of both analytically and empirically. Altmann et al. [3] investigated the value creation for providers and users of software service

platform at different levels of interoperability under an economic approach. Their goal was to provide a knowledge of how investments in interoperability and portability impact cost, enable cost-effective service integration and create value, as well as design new strategies for optimising investments for platform providers. Bahadur et al. [4] presented a distributed framework that tunes the thread pool systems based on request arrival rate and balances the load between nodes of distributed systems. Besides, the framework adjusts the thread pool size, when its overload control mechanism detects throughput fall caused by the increase of requests. Tarvo and Reiss [38] presented an approach for design performance models of multithreaded programs using hierarchical discrete-event models, in which each tier of the model simulate a factor that impact the performance of programs and the interaction between the model tiers simulates the mutual influence of these factors on performance. Jeon and Jung [25] proposed an approach that optimises thread pool management by controlling the number of threads in the handler thread pool according to the number of received packets. They aimed to increase the performance concerning the speed of processing of the requests, by adjusts the number of threads and throughput according to the number of packets, the capacity of the queue, and retransmission time of each packet in IoT networks. Stetsenko and Dyfuchyna [36] designed a model of the multithreaded algorithm using Petri-object simulation technology based on stochastic Petri net and object-oriented approach. The model o+9-6f a thread pool was developed to prove the relationship between algorithm complexity, computing resources and parameters of the thread pool. [12] propose a model for parallel real-time tasks implemented with thread pools and with blocking synchronization mechanisms to meeting precedence constraints. The work compares the proposed analysis approaches with prior work to evaluate the schedulability due to reduction of concurrency. Berned et al. [6] propose a generic methodology for to rightly tune the number of threads according to the application, using learning algorithms in static strategics, by inferring the execution behaviour of parallel applications. The work compares the performance and the energy consumption of the execution of the applications using the number of threads found by their learning algorithm with a dynamic one.

The purpose of our article is different from the others because it aims to evaluate the performance of run-time systems of integration platforms using two distinct thread pool configuration models: global and local thread pool. In Table 1, we summarised the works related to performance analysis and thread pool configuration.

3 Background

In this section, we present an overview of integration processes and run-time systems of integration platforms. After, we describe the main thread pool configuration models: global and local.

Table 1 Related works comparison.

Work	Research field	Goal
Braun and Krus [11]	Power systems	Load balancing and synchronisation
Ágnes Bogárdi-Mészöly and Rövid [9]	Software system	Prediction performance
Pasha et al. [31]	Power systems	Prediction of power consumption
Ahmad et al. [1]	Software system	Prediction performance
Altmann et al. [3]	Software system	Interoperability
Bahadur et al. [4]	Distributed system	Load balancing
Tarvo and Reiss [38]	Software system	Prediction performance
Jeon and Jung [25]	IoT networks	Increase the performance
Stetsenko and Dyfuchyna [36]	Software system	Prediction performance.
Casini et al. [12]	Software system	Evaluation of schedulability
Berned et al. [6]	Software system	Energy consumption
[Our proposal]	EAI	Evaluation of performance

3.1 Integration Process

An integration process is a computational program that allows the exchange of data and functionalities amongst a set of applications $A = A_1, A_2, ..A_k$. Its conceptual model is a workflow composed of set of tasks $T = T_1, T_2, ..T_n$ connected by «communication channels». Data, wrapped as «messages», flow through the workflow. A message has a header and a body. The former carries data custom properties, and the latter, the payload data. A message is transformed into one or more messages in its processing in the workflow. Usually, there are more of one path that a message can follow into a workflow. Figure 1 depicts a conceptual model of an integration process. Small rectangles, identified with a letter «T», represent tasks. Arrows linking tasks represent communication channels. Rectangles outside the integration process, identified with a letter «A», represent applications. Dark small rectangles and arrows highlight one possible path to this process.

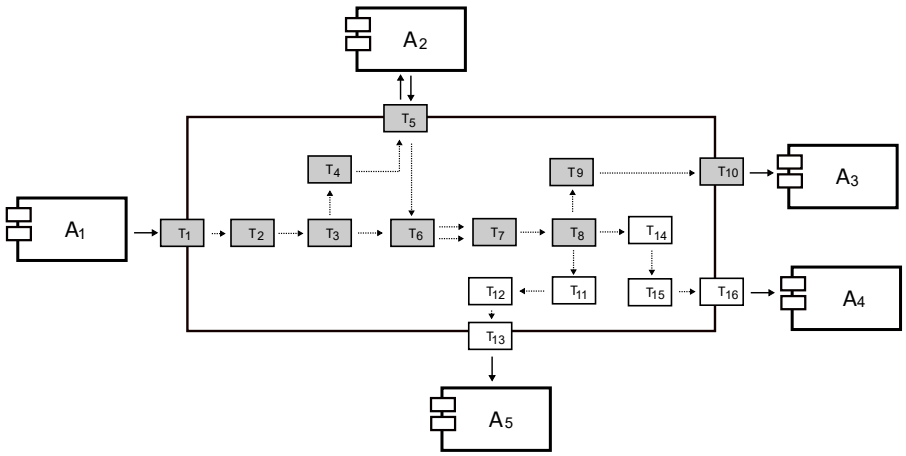


Fig. 1 Integration process conceptual model.

Each task implements an integration pattern, which represents an atomic operation that transforms, filters, splits, joins or routes messages. Messages arrive into a task by one or more inputs, as well as messages leave by one or more outputs of tasks, according to the operation that they implement. The tasks are arranged in order of dependence in which must be executed. Thus, a message only can be processed by a task after this message has already processed by every predecessor task of a path. After a task processes a message, it is written to the communication channel that connects this task with the next tasks of the path. The workflow usually has parts that can be executed in parallel, obeying the order of dependence in an integration process.

3.2 Run-time System

The run-time system is the component of the integration platform responsible for the execution of integration processes. Usually, it has a scheduler, task queue, thread pool, and monitors. The schedule is the main element since it manages and orchestrates the activities of the other elements. The task queue maintains the tasks that wait for threads to execute them. Thread pools are groups of threads that perform tasks. Monitors control frequency and logging system to notify about warnings and errors.

The execution model of a run-time system settles the execution way of the integration processes, sequencing tasks and allocated threads. In our article, we address the execution model task-based that treats task instances, i.e. tasks assigned to execution by a thread. In this model, a task is considered ready to be executed when there are messages in all their inputs. However, when there is no available thread, the task waits in a queue. When there is an available thread, it iteratively polls the task queue and selects a task instance to execute it.

The thread pool configuration model of a run-time system determines the way how threads are grouped. We approach two kinds of configuration model: global and local thread pool.

In the configuration model of the global thread pool, there is a single task queue and a single thread pool to all the tasks of the integration process. All threads are responsible for executing the instances of the first task, processing the incoming messages that arrive at the input channel of the first task. Only when all messages finish their processing by a task, it is that the messages begin to be processed by the subsequent task.

In the configuration model of the optimised local thread pool, there is a task queue and thread pool for every task. In the proposal of [18], the number of threads in each local thread pool is determined by an optimisation algorithm whose objective function is to minimise the average total processing time of incoming messages per unit time. Each thread pool is responsible by the execution of the instances of an only task to which it is dedicated. In this way, the initial message load are arriving at the channel of the first task is processed in the pool responsible by the first task. Thus, when a message

finishes being processed by a task, it can already be processed in the successor task as long as there are available threads in the dedicated pool to perform this next task.

4 Objective Function

Execution time and processing time are generally used as performance metrics for systems. This section, we describe mathematical models for these metrics involved in message processing in an integration process. After, we define the objective function used to minimise these times.

The processing time TP_{t_i} of a task t_i of the integration process is calculated by the sum of its execution time TE_{t_i} with its waiting time in the queue TF_{t_i}, c.f. Equation 1.

$$TP_{t_i} = TE_{t_i} + TF_{t_i} \tag{1}$$

Makespan is the most extended time interval between the start and end of processing a message within an integration solution [2], [7]. We propose an analytical model for makespan of integration processes, in each one of thread pool configuration models, considering the non-variation of the processing time of tasks. For the global thread pool model, the makespan can be calculated by Equation 2. For the local thread pool model, the makespan can be calculated by Equation 3, where $to, t_{messages}$ is the total number of message, $tot_{threads}$ is the total number of threads, Tot_{tasks} is the total number of tasks, and α is a constant equals one, when the mod of the integer division of the total number of message by the total number of threads is greater than zero, and, otherwise, α equals zero.

$$\begin{aligned} makespan_{global\ thread\ pool} &= \left(\frac{tot_{messages}}{tot_{threads}} + \alpha\right) \cdot \sum_{i=1}^{tot_{tasks}} TP_{t_i} \\ \alpha &= \begin{cases} 1, & \text{if } \frac{tot_{messages}}{tot_{threads}} > 0, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{2}$$

$$\begin{aligned} makespan_{local\ thread\ pool} &= \sum_{i=1}^{tot_{tasks}} \left(\frac{tot_{messages}}{tot_{threads(t_i)}} + \alpha_i\right) \cdot TP_{t_i} \\ \alpha_i &\begin{cases} 1, & \text{if } \frac{tot_{messages}}{tot_{threads(t_i)}} > 0, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

We define makespan average by the division of the makespan by the total number of messages, cf. Equation 4.

$$\overline{makespan} = \frac{makespan}{tot_{messages}} \tag{4}$$

The objective function seeks to minimise the makespan average and this formulation is represented by Equation 5:

$$Minimise\ \{\overline{makespan}\} \tag{5}$$

5 Study Case

In this section, we describe the case study used in our experiment. It involves a real-world integration problem whose objective is to improve the call centre application at Unijuí University, by automatisation of the charge of personal phone calls by means phones of the university. Every call has an access code that identifies who employee. The call is registered to future debt in the wages of employees. Employees can be notified about their calls and their respective charges both by e-mail and short message service (SMS).

5.1 Software Ecosystem

The software ecosystem is composed of five applications that were designed without integration concerns in mind; their data layer makes the interaction with them. Figure 2 shown the integration process and the applications: Call Centre, Human Resources System, Payroll System, Mail Server, and SMS Notifier. The Call Centre records every call every employee makes from a university belonged phone. The code is also used to correlate phone calls with the information in the Human Resources System and the Payroll System. The Human Resources System supplies personal data concerning employees, and the Payroll System computes their wages. The Mail Server and the SMS Notifier notify employees about their charges. The former provides e-mail service and the later offers short message system services. There are 16 tasks identified with T_i, where i ranges from 1 to 16. The T_1 is an input task, in which messages are firstly processed. The T_5 is a task that solicits and receives data of the Human Resources System application. The T_10, T_1 and T_16 are output tasks, in which messages are lastly processed.

5.2 Computational Model of Simulation

To simulate the execution with both thread pool configuration (global and local), we implemented two computer programs, whose source codes are publicly available for download [1]. The first program simulates the execution of the tasks through a global thread pool and the second simulates the execution of the same tasks through the optimal local thread pool model. For the global thread pool, we have implemented an algorithm that calculates the makespan by Equation 2, after, we calculates the makespan average by Equation 4. For

[1] https://github.com/gca-research-group/Simulation-ThreadPoolConfig.git

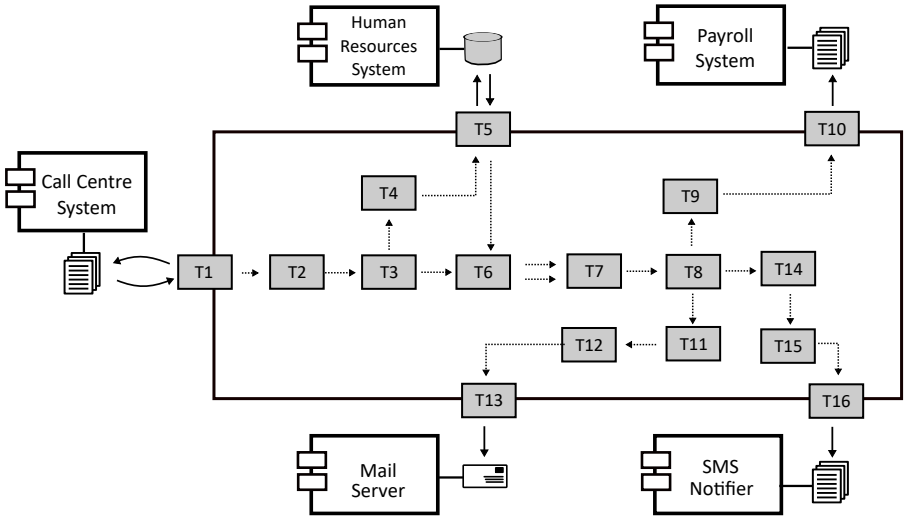


Fig. 2 Integration process real-world.

optimal local thread pool model, we use the implementation of the algorithm based on the metaheuristic PSO, which finds the best thread configuration for local pools [18], calculates the makespan by Equation 3 and the makespan average by Equation 4. In both cases, the number of threads is the same, i.e., the number of threads in the global thread pool is equal to the sum of the number of threads in the local pools. The input parameters for the two programs are (1) total number of threads, (2) the total number of incoming messages (workload), and (3) vector containing the processing times of the tasks. In this last input parameter (3), the number of columns in the vector defines the number of tasks to be performed, and the column index denotes the order of execution.

The programs generate an array with the final processing times of each message for each of the tasks of the flow. The array row index represents the executed task, and the column index represents the message. The makespan average is calculated by the arithmetic average of the final message times in the last flow task. The program graphically shows the sequence of message processing in the tasks and returns the makespan average. In the case of the optimal local thread pool model, it also returns the optimal configuration of the pools.

6 Experimental Results

In this section, we present the experiment that compares two implementations of algorithms concerning the performance of the execution in an integration process by the global and local thread pool configuration models. First, we describe the environment in which the experiment was carried out

and present the scenarios used and, the observed variables. Next, we report and discuss the results obtained in each model.

Simulation in multithreaded architectures allows the evaluation of multithreading techniques and their influence on performance as well as the investigation of the dependence between the performance and characteristics of different numbers of messages [39]. We used the protocols of procedures for controlled experiments in the engineering studies field indicated by [23], [42], and [5]. Besides, we followed the guidelines of literature, which claim that statistical theory must be used to validate results from experiments on performance [20] because the statistic deals adequately with the non-determinism factors present in computational systems, such as run-time systems [16]. Thus, we used ANOVA [41] and Scott & Knott [35] statistical techniques to validate the results. Every step of the experiment and its validation is detailed in the following sections.

6.1 Research Questions and Hypothesis

Our experiment answer the following research question:

RQ: In a high number of messages, is the performance of the execution of integration processes better using a thread pool configuration of the optimised local model than using the global model?

We provide a hypothesis that has to be confirmed or refuted by the experiment, respectively:

H: In a high number of messages, the performance of the execution of integration processes using an optimal local thread pool model is better than using the global model.

6.2 Environment and Support Tools

The experiments were conducted on a machine containing with 16 processors Intel Xeon CPU E5-4610 V4, 1.8 GHz, 32GB of RAM, and operating system Windows Server 2016 Datacenter 64-bits. The Matlab [28] software, version R2018, was used to create and execute the algorithms. The Genes [13] software, version 2015.5.0, was used to process the descriptive statistics, ANOVA and Scott & Knoot techniques for the makespan measured in this study.

6.3 Variables

The independent variables are:

`Number of threads`. The number of threads that can be distributed to the thread pools. The value for this variable was 100 threads.

`Number of messages`. The total number of incoming messages (workload). The values for this variable were 10^2, 10^3, 10^4, 10^5, and 10^6.

`Thread pool models`. The model for thread pool configuration. The values for this variable were global and optimal local.

The dependent variables are:

`Makespan average`. The meantime, a message takes to be processed by all tasks that compose the longest path of the integration process.

6.4 Execution and Data Collection

The execution of the algorithms was conducted using the longest path of the integration process, highlighted in Figure 2, aiming to evaluate how the processing performs under an increased number of messages triggered by the worst-case scenario. Table 2 shows the times, in milliseconds (ms), for every task of the path, obtained from the execution of the actual implementation of the integration process and supported from the literature [22].

Table 2 Processing times of tasks.

Task	Execution Time	Waiting Time	Processing Time
t_1, t_9	0.005	2	2.005
t_2, t_{11}, t_{14}	0.003	1	1.003
t_3, t_8	0.553	1	1.553
t_4, t_7, t_{10}, t_{12}, t_{13}, t_{15}, t_{16}	0.005	1	1.005
t_5	0.005	4	4.005
t_6	0.004	1	1.004

The literature classified the experiment as a termination simulation, where the experiment output is express as a function of the initial conditions. Usually, the method of the repetitions, with a repetitions number between 20 and 30, is adopted for statistical analysis of the results. This repetitions number is sufficient to obtain a population mean, considering distributions with more extreme values that a normal distribution [34]. We used 200 different scenarios, which are synthesised, cf. Table 3. We executed the algorithms by setting the input parameters:

- Total number of threads: 100.
- Total number of incoming messages: 10^2, 10^3, 10^4, 10^5,and 10^6.
- Processing time vector: [2.005; 1.003; 1.553; 1.005; 4.005; 1.004; 1.005; 1.553; 2.005; 1.005; 1.003; 1.005; 1.005; 1.003; 1.005; 1.005]

We considered the execution algorithm and the number of tasks of the integration process, thus considered that 10 is a reasonable number to amount of solutions tested by algorithm of optimisation of the optimal local thread pool model. The execution of each algorithm was repeated 20 times for every

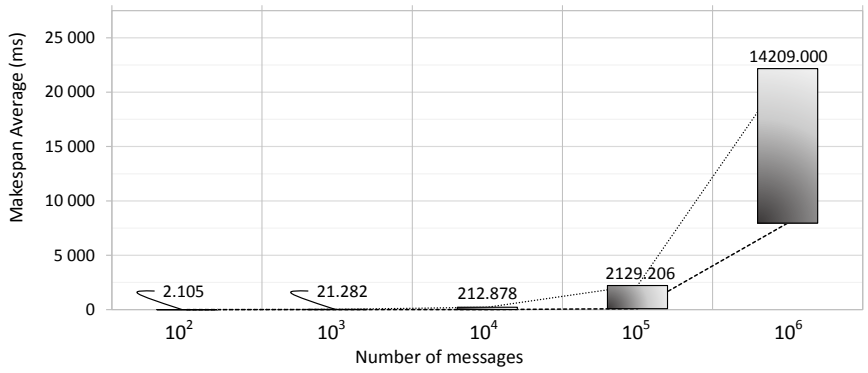
number of messages. In every execution, we collected the makespan average processing time and the time that each algorithm consumed to execution. We exported the data collected to a spreadsheet application, where we analysed them to build the mathematical model and charts.

Table 3 Scenarios tested.

Number of threads:	Number of messages:	Thread pool models:	Number of: repetitions:
«100»	«10^2, 10^3, 10^4, 10^5, 10^6»	«global and local»	«1, 2, ..., 20»
1	5	2	20
Total of scenarios:	1 x 5 x 2 x 20 =		**200**

6.5 Results

In this section, we present and discuss the results of the experiment. Line charts present the average of the makespan measured in 20 execution of each algorithm, cf. Figure 3. The x-axis represents the number of messages, and the y-axis represents the makespan average, in milliseconds (ms). The rectangles represent the difference between the makespan of using the global and local thread pool configuration. This difference was 2.105 ms for 10^2 messages, 21.282 ms for 10^3 messages, 212.878 ms for 10^4 messages, 2129.206 ms for 10^5 messages, and 14209 ms for 10^6 messages. The table below chart of Figure 3 shows the average of the makespan measured in each number of messages. In the first line are the averages of the makespan measured using the global thread pool configuration and in the second line are the averages of the makespan measured using the local thread pool configuration optimised.



Global	2.216	22.169	221.69	2216.9	22169
Local	0.111	0.886	8.81	87.6	7960

Fig. 3 Makespan average.

The variance analysis statistical technique allows the evaluation of the difference of the variations in the makespan average, which derived from thread pool configuration models and the ones that caused by other random factors, named error. Table 4 presents the analysis of variance of the makespan average.

The total of results was calculated upon the multiplication of the number of repetitions, by the number of configuration models, cf. Equation 6. In our case, $\mathsf{Total\ results} = 20 \cdot 2 = 40$. The total of results minus 1 calculates the total freedom degree, cf. Equation 7. In our case, $df_{total} = 40 - 1 = 39$. The degree of freedom of the configuration models is calculated by the number of possible values of configuration models tested subtracting 1, cf. Equation 8. In our case, $df_{models} = 2 - 1 = 1$. The degree of freedom of error is calculated by the difference between total freedom degree and the degree of freedom of models, cf. Equation 9. In our case, $df_{error} = 39 - 1 = 38$.

$$\mathsf{Total\ results} = \mathsf{repetitions} \cdot \mathsf{number\ of\ models} \tag{6}$$

$$df_{total} = \mathsf{total\ results} - 1 \tag{7}$$

$$df_{models} = \mathsf{number\ of\ models} - 1 \tag{8}$$

$$df_{error} = df_{total} - df_{models} \tag{9}$$

The analysis of variance of the makespan average shows the average square of 4412.04 for the thread pool configurations models and 200.83 for error. The overall average was equal to 39.99 seconds, and the coefficient of variation (cv) was 35.43 %. Coefficient of variation is a standardised measure of the dispersion of a probability or frequency distribution, which shows the spread of the variability of the data relative to the mean. This measure is capable of comparing results from different works involving the same variable-response, making it possible to quantify the precision of the research [26], [19]. Parameters of the experiment, such as the number of repetitions and experimental design influence the experimental error But, in similar conditions, the lower the coefficient of variation, the more accurate is the experiment with lower [19].

Table 4 Variance analysis of the makespan average.

Sources of variation	**Degree of freedom**	**Average square** *Number of messages*				
		10^2	10^3	10^4	10^5	10^6
Thread pool model	1	44.31 †	4529.33 †	453170.98 †	45335191.48 †	2018956810 †
Error	38	0.0001	0.0182	1.4809	183.96	2375092.10
Total	39					
Overall average		1.16	11.52	115.25	1152.29	15064.50
Coefficient of variation (%)		0.85	1.17	1.05	1.17	10.23

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

Scott & Knoot technique is widely adopted in performance experiments because it is simple. The literature considers this technique rigorous because it only considers relevant differences between the experimented alternatives. In this case, the alternatives are the two models of thread pool configuration. The Level of significance is a parameter used in the Scott & Knott algorithms to create the groups of averages, which the default value is 0.05. This parameter helps to measure the relevance of the difference between the alternatives [24]. Usually, this technique is used when the analysis of variance indicates significant statistical between the dependent variable. It allowed the identification of the effect in the use of every model of thread pool configuration in the makespan average, cf. Table 5. For every number of messages, there are two columns: «Average» and «Group». «Average» presents the makespan average in the 20 repetitions. «Group» classifies the makespan averages, where groups of the different letter are statistically different between themselves. There were two groups «a» and «b». In «a» group is the thread pool model with the highest makespan average and, in «b» group is the thread pool model with the lowest makespan average.

Table 5 Average of makespan by Scott & Knott technique.

Thread pool model	**Makespan average & Group** *Number of messages*									
	10^2		10^3		10^4		10^5		10^6	
Global	2.2169	a	22.169	a	221.69	a	2216.90	a	22169	a
Local	0.119	b	0.8868	b	8.8119	b	87.69	b	7960	b

Error level of 5% by the Scott & Knoot technique.

6.6 Discussion and Comparison

Analysing all numbers of messages tested, we realised that the makespan average achieved with local thread pool configuration optimised model is lower than that obtained with the global thread pool. Figure 3 shows that as greater the number of messages as greater is this difference between the makespan averages. Figure 4 shows the difference of makespan average as a function of the number of messages and an approximation found by linear regression, represented by a dotted line. This approximation is an exponential function, described by Equation 10, where $tot_{messages}$ is the total number of message. This equation indicates that this difference exponentially grows when the number of messages grows. The correlation coefficient, better known as R^2, allows the determination of the degree of linear correlation of variables, considering regression analysis. Thus, the closer the R^2 is of the 1, the exacter the mathematical model will be. For Equation 10, the R^2 equals 0.9986 indicates the mathematical model is a reliable approximation.

$$\overline{makespan} = 0.248\ e^{2.224 \cdot tot_{messages}} \tag{10}$$

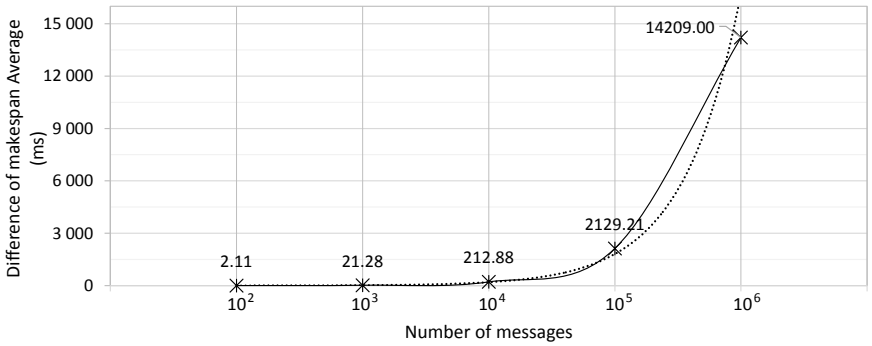


Fig. 4 Difference makespan average.

The analysis of variance reported confirmed that the local thread pool optimised generates a significant improvement in makespan average in every scenario tested. The low values for coefficients of variation indicated the adequacy and reliability of the experiment. In The Scott & Knott averages comparison technique, the lowest makespan averages occurred with the local thread pool optimised in every scenario tested, cf. Table 5. The largest difference between makespan was 14209 ms when the number of messages was 10^6. In this table, groups of different letters indicate that there is a difference statistically between them. The makespan average using the global thread pool, which belongs group «a», is different statistically compare to makespan average using local thread pool optimised, which belongs to group «b».

6.7 Threats to Validity

According to Cruzes and ben Othman [14], validity threats are presented in any empirical researches. We sought to identify the possible causes of disturbance in the experiment and tried to mitigate these threats. We discuss the validity threats, separating into constructor, conclusion, internal, and external validity.

6.7.1 Constructor Validity

To deal with constructor validity threats, we studied previous articles [18] and based our experiment based on the procedures from empirical software engineering presented by Jedlitschka and Pfahl [23], Wohlin et al. [42], and Basili et al. [5]. We gather information about the execution environment, supporting tools, variables, execution and data collection. Then, we design the experiment with 200 different scenarios and used statistical techniques to validate the experimental results.

6.7.2 Conclusion Validity

According to Feldt and Magazinius [15], conclusion validity refers to the assurance that the treatment used in the experiment is the cause of the actual outcome observed. To obtain this assurance, we resorted statistical techniques and, then, we verified the performance measures observed in our experiment is associated with the number of messages and also with algorithms used.

6.7.3 Internal Validity

Internal validity copes uncertain factors or not measured in order to their effects do not impact the outcome of the treatment [15]. We mitigated these factors in the execution time of the algorithm, performing the experiment with the machine set on security mode, using minimal features and disconnected from the Internet during all executions.

6.7.4 External Validity

External validity refers to expand the results beyond the ambit of our study [15]. The experiment can be used to compare other scenarios with other integration processes, different numbers of messages to generalise the results. To generalise the results, in future work, we intend to perform our experiment with an extensive data set.

7 Conclusion

Enterprises have taken advantages of services offered by cloud computing, amongst them, integration Platform as a Service (iPaaS) that are specialised software tools that allow for keeping the consistency and the synchronism of the data on all the applications. The iPaaS represents a new option for small and mid-sized businesses that need to integrate their business processes without increasing their costs [10]. However, integration platforms need to be adapted to tackle environments involve the large volume, velocity, and variety of data from several different sources and types of devices [32]. The run-time system is the component of the integration platforms responsible for running integration processes so, it is directly related to the performance of the integration platforms.

The efficiency of the run-time system of the integration platforms is directly related to the scheduling algorithm of the tasks and the allocation of threads to execute them. An inefficient algorithm leads to an increase in execution time, thus degrading the performance of the execution of integration processes. This article aims to analyse the behaviour of thread pool configuration models, at high workloads. Our experiment compared two models of thread pool configuration: global and local optimised. In the first, there is a single thread pool for all tasks. In the second, there is a thread pool for each task, and

the algorithm finds the best distribution of the threads to the local pools, taking into account the processing time of the task. We used a case study with a real-world integration process in the scenarios that varied the numbers of messages.

The results showed that, in high workload scenarios, the optimised local thread pool model performs better than the global thread pool model. This assessment can help save costs for the companies, especially ones that use cloud services, due to the pay-as-you-go charging model of the cloud, where companies pay for the use time of the services. We evaluated the models by the makespan average and the results validated by the statistical techniques ANOVA and Scott & Knott. We sum up the main conclusions from the results found in our experiment, answered our research questions and validate our hypothesis. Regarding the conclusions:

- The makespan average using global thread pool configuration can be analytically calculated by Equation 2.
- The makespan average using local thread pool configuration can be analytically calculated by Equation 3.
- The local thread pool configuration optimised generated lower makespan average than global thread pool configuration.
- An exponential equation represents the difference of makespan average between two models of thread pool configuration as a function of the number of messages. Thus, the makespan average tends to infinite when the number of messages is huge.
- The results of the experiment are valid according to the statistical techniques: ANOVA and Scott & Knoot.

Results of the experiment confirmed our hypothesis, and we answered our research question, following as.

- **RQ:** The makespan average obtained with local thread pool optimised configuration model in run-time systems of integration platforms was 2.8 times lower than the makespan average obtained with a global thread pool, with a workload of 10^6 messages. Thus, the performance of the execution of integration processes better using local thread pool optimised.

8 Acknowledgements

This work was supported by the Brazilian Co-ordination Board for the Improvement of University Personnel (CAPES) under grants 73318345415 and the Research Support Foundation of the State of Rio Grande do Sul (FAPERGS) under grant 17/2551-0001206-2.

References

1. Ahmad T, Truscan D, Porres I (2018) Identifying worst-case user scenarios for performance testing of web applications using markov-chain workload models. Future Generation Computer Systems 87:910–920
2. Ali S, Maciejewski AA, Siegel HJ, Kim JK (2003) Definition of a robustness metric for resource allocation. In: Parallel and Distributed Processing Symposium (PDPS), pp 10–21
3. Altmann J, Ángel Bañares J, Petri I (2018) Economics of computing services: A literature survey about technologies for an economy of fungible cloud services. Future Generation Computer Systems 87:828 – 830
4. Bahadur F, Umar AI, Khurshid F (2018) Dynamic tuning and overload management of thread pool system. International Journal of Advanced Computer Science and Applications 9:444–450
5. Basili VR, Rombach D, Kitchenham KSB, Selby D, Pfahl RW (2007) Empirical Software Engineering Issues. Springer Berlin/Heidelberg
6. Berned G, Rossi FD, Luizelli MC, Beck ACS, Lorenzon AF (2020) Decreasing the learning cost of offline parallel application optimization strategies. In: Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp 144–151
7. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K (2005) Task scheduling strategies for workflow-based applications in grids. In: Cluster Computing and the Grid (CCGrid), vol 2, pp 759–767
8. Bogado V, Gonnet S, Leone H (2014) Modeling and simulation of software architecture in discrete event system specification for quality evaluation. Simulation 90:290–319
9. Ágnes Bogárdi-Mészöly, Rövid A (2016) Performance modeling of web-based software systems with subspace identification. Acta Polytechnica Hungarica 13:27–41
10. Brahmi Z, Gharbi C (2014) Temporal reconfiguration-based orchestration engine in the cloud computing. In: International Conference on Business Information Systems (ICBIS), pp 73–85
11. Braun R, Krus P (2016) Multi-threaded distributed system simulations using the transmission line element method. Simulation 92:921–930
12. Casini D, Biondi A, Buttazzo G (2019) Analyzing parallel real-time tasks implemented with thread pools. In: ACM/IEEE Design Automation Conference (DAC), pp 1–6
13. Cruz CD (2006) Programa Genes - Estatística Experimental e Matrizes. Editora Universidade Federal de Viçosa
14. Cruzes DS, ben Othman L (2017) Threats to validity in empirical software security research. In: Empirical Research for Sof. Security, pp 295–320
15. Feldt R, Magazinius A (2010) Validity threats in empirical software engineering research-an initial survey. In: Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), pp 374–379
16. Frantz RZ, Corchuelo R, Arjona JL (2011) An efficient orchestration engine for the cloud. In: Int. Conf. on Cloud Computing Technology and

Science (CloudCom), pp 711–716
17. Frantz RZ, Corchuelo R, Roos-Frantz F (2016) On the design of a maintainable software development kit to implement integration solutions. Journal of Systems and Software 111:89–104
18. Freire DL, Frantz RZ, Roos-Frantz F (2019) Towards optimal thread pool configuration for run-time systems of integration platforms. International Journal of Computer Applications in Technology 60:1–18
19. Garcia CH (1989) Tabelas para classificação do coeficiente de variação. IPEF
20. Georges A, Buytaert D, Eeckhout L (2007) Statistically rigorous java performance evaluation. ACM SIGPLAN Notices 42:57–76
21. Harman M, Lakhotia K, Singer J, White DR, Yoo S (2013) Cloud engineering is search based software engineering too. Journal of Systems and Software 86:2225–2241
22. Haugg IG, Frantz RZ, Roos-Frantz F, Sawicki S, Zucolotto B (2019) Towards optimisation of the number of threads in the integration platform engines using simulation models based on queueing theory. Revista Brasileira de Computação Aplicada 11:48–58
23. Jedlitschka A, Pfahl D (2005) Reporting guidelines for controlled experiments in software engineering. In: Int. Sym. Empirical Soft. Engineering (ESEM), pp 95–104
24. Jelihovschi E, Faria J (2014) Scottknott: A package for performing the scott-knott clustering algorithm in r. Tendências em Matemática Aplicada e Computacional 15(1):3–17
25. Jeon S, Jung I (2018) Experimental evaluation of improved IoT middleware for flexible performance and efficient connectivity. Ad Hoc Networks 70:61–72
26. Kalil EB (1977) Princípios de técnica experimental com animais. ESALQ
27. Lee J, Wu H, Ravichandran M, Clark N (2010) Thread tailor: dynamically weaving threads together for efficient, adaptive parallel applications. ACM SIGARCH Computer Architecture News 38:270–279
28. Leonard NE, Levine WS (1995) Using MATLAB to analyze and design Control Systems. Benjamin-Cummings Publishing Company
29. Linthicum DS (2017) Cloud computing changes data integration forever: What's needed right now. IEEE Cloud Computing 4:50–53
30. Lorenzon AF, Cera MC, Beck ACS (2016) Investigating different general-purpose and embedded multicores to achieve optimal trade-offs between performance and energy. Journal of Parallel and Distributed Computing 95:107–123
31. Pasha MA, Gul U, Mushahar M, Masud S (2017) A simulation framework for code-level energy estimation of embedded soft-core processors. Simulation 93:809–823
32. Ritter D, May N, Sachs K, Rinderle-Ma S (2016) Benchmarking integration pattern implementations. In: International Conference on Distributed and Event-based Systems (DEBS), pp 125–136

33. Ritter D, Dann J, May N, Rinderle-Ma S (2017) Hardware accelerated application integration processing: Industry paper. In: International Conference on Distributed and Event-based Systems (DEBS), pp 215–226
34. Sargent RG (2013) Verification and validation of simulation models. J of simulation 7:12–24
35. Scott AJ, Knott M (1974) A cluster analysis method for grouping means in the analysis of variance. Biometrics 30(3):507–512
36. Stetsenko IV, Dyfuchyna O (2019) Thread pool parameters tuning using simulation. In: International Conference on Computer Science, Engineering and Education Applications (ICCSEEA), pp 78–89
37. Suleman MA, Qureshi MK, Patt YN (2008) Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on cmps. ACM Sigplan Notices 43:277–286
38. Tarvo A, Reiss SP (2018) Automatic performance prediction of multithreaded programs: a simulation approach. Automated Software Engineering 25:101–155
39. Vlassov V, Ayani R, Thorelli LE (1997) Modeling and simulation of multithreaded architectures. Simulation 68:219–230
40. van der Weij W, Bhulai S, van der Mei R (2009) Dynamic thread assignment in web server performance optimization. Performance Evaluation 66:301–310
41. Wilkinson GN, Rogers CE (1973) Symbolic description of factorial models for analysis of variance. Journal of the Royal Statistical Society Series C 22(3):392–399
42. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer Science & Business Media