Revista Brasileira de Computação Aplicada, April, 2019

DOI: 10.5335/rbca.v11i1.8784
Vol. 11, Nº 1, pp. 48–58
Homepage: seer.upf.br/index.php/rbca/index

ISSN 2176-6649

ORIGINAL PAPER

Towards optimisation of the number of threads in the integration platform engines using simulation models based on queueing theory

Igor G. Haugg[1], Rafael Z. Frantz[1], Fabricia Roos-Frantz[1], Sandro Sawicki[1] e Benjamim Zucolotto[1]

[1]Universidade Regional do Noroeste do Estado do Rio Grande do Sul

*{ihaugg,rzfrantz,frfrantz,sawicki,benjamim.zucolotto}@unijui.edu.br

Received: 2018-10-23. **Revised:** 2019-02-27. **Accepted:** 2019-03-05.

Abstract

The use of applications is important to support the business processes of companies. However, most of these applications are not designed to function collaboratively. An integration solution orchestrates a group of applications, allowing data and functionality reuse. The performance of an integration solution depends on the optimum configuration of the number of threads in the runtime engine provided by the integration platforms. It is common that this configuration relies on the empirical knowledge of the software engineers, and it has a direct impact on the performance of integration solutions. The optimum number of threads may be found by means of simulation models. This article presents a methodology and a tool to assist with the generation of simulation models based on queuing theory, in order to find the optimum number of threads to execute an integration solution focusing on performance improvement. We introduce a case of study to demonstrate and experiments to evaluate our proposal.

Key words: Simulation; Queueing Theory; Enterprise Application Integration; Integration Platforms; Business Process Simulation.

Resumo

O uso de aplicativos é importante para suportar os processos de negócios das empresas. No entanto, a maioria desses aplicativos não foi projetada para funcionar de maneira colaborativa. Uma solução de integração orquestra um grupo de aplicativos, permitindo a reutilização de dados e funcionalidades. O desempenho de uma solução de integração depende da configuração ideal do número de threads no motor de execução das plataformas de integração. Atualmente essa configuração depende do conhecimento empírico dos engenheiros de software e portanto tem um impacto direto no desempenho das soluções de integração. O número ideal de threads pode ser encontrado por meio de modelos de simulação. Este artigo apresenta uma metodologia e uma ferramenta para auxiliar na geração de modelos de simulação baseados na teoria das filas a fim de encontrar o número ideal de threads para executar uma solução de integração com foco na melhoria do desempenho. Apresentamos um caso de estudo para demonstrar e experimentos para avaliar nossa proposta.

Palavras-Chave: Integração de aplicativos corporativos; Plataformas de integração; Simulação; Simulação de Processos de Negócios; Teoria das filas.

1 Introduction

The use of applications is important to support the business processes of companies. Companies usually have various applications in their software ecosystem, which are developed in-house or acquired from third parties. This way, the software ecosystem becomes heterogeneous, with applications that have possibly been developed using different programming languages and data models, running on different operating systems, and, generally, have not been designed to work in a collaborative form. It is common that the business processes involve data and functionality present in distinct applications, what requires collaboration amongst these applications. Enterprise application integration (EAI) is a research field that concerns with the development of methodologies, techniques and tools to make different applications, which were not developed with the purpose of working together, to collaborate (Hohpe and Woolf; 2012). An integration solution orchestrates a group of applications, without the perception that they are being integrated, and without causing dependency in the applications with the solution, allowing data and functionality reuse (Ritter et al.; 2017).

There are several message-based integration platforms available in the market for the development of integration solutions. Amongst the platforms that represent the state-of-the-art technology are Camel (Ibsen and Anstey; 2017), Spring Integration (Pandey; 2015), Mule (Dossot et al.; 2014), Petals (Surhone et al.; 2010), Apache Flume (FLU; 2017), Apache Nifi (NIF; 2017) and Guaraná (Frantz et al.; 2016). These platforms support the *integration patterns* documented by Hohpe and Woolf (2012), which have consolidated as a reference in the integration market when creating integration platforms, and follow the architectural style of *pipes-and-filters* (Hohpe and Woolf; 2012). In an integration solution, pipes represent message channels, and filters represent atomic tasks that implement a concrete integration pattern to process encapsulated data in messages. The adoption of this architecture allows to desynchronising the tasks that make up the integration solution.

Usually, these integration platforms provide a domain-specific language, a development toolkit, an environment for testing, a monitoring system, and a runtime engine (Freire, Frantz, Roos-Frantz and Sawicki; 2019). The domain-specific language is focused on the elaboration of conceptual models with an abstraction level close to the problem domain. The development toolkit is a set of software tools that allows the transformation of the conceptual model into an executable code. The environment for testing allows trying individual parts or all the integration solution. The monitoring tool is used to follow, at execution time, the operation of the integration solution and to detect errors that may occur during the processing of messages. The runtime engine provides all the support necessary to execute these integration solutions. Therefore, its performance is directly related to the performance of the integration solutions.

These message-based integration platforms have been constructed using Java Technology (Schildt; 2017) and their runtime engine is organised around a first-in-first-out (FIFO) queue and a set of threads. The queue is used to store tasks that are ready to be executed in an integration solution, and the execution of these tasks depends on the availability of threads in the runtime engine. Threads represent computational resources within the instance of the Java Virtual Machine (Lindholm et al.; 2014) in which the runtime engine executes. They are managed by a mechanism provided in Java called Executor (Lindholm et al.; 2014), which allows to create and allocate threads and pools of threads within the runtime engine to execute an specific integration solution. The configuration of these pools is an important activity involved in the deployment and execution of integration solutions and is commonly performed by software engineers Freire, Frantz and Roos-Frantz (2019). In a simplified way, it means to define the size of the pool, i.e., the number of threads that must be used by the runtime engine to execute an integration solution. Currently, this configuration relies on the empirical knowledge of software engineers, which brings risks because a low number of threads causes the accumulation of tasks to be executed in the queue, leading to a poor performance. An over dimensioned number of threads increases computational costs with memory and CPU time involved in the process of context switching of these threads and leads to a negative impact on the performance as well (Pusukuri et al.; 2011).

In this article, we introduce a methodology that can be used to estimate the optimum number of threads that must be configured at the runtime engine to execute an integration solution seeking at best performance. This methodology is supported by a software tool named *ModelGen*, which allows for the analysis of an integration solution taking as input its conceptual model and generates a simulation model based on Queuing Theory (Klcinrock; 1975). This simulation model can be run on Simulink (Chaturvedi; 2017) to provide the optimum number of threads. We demonstrate our methodology and *ModelGen* in action through a case of study called Café (Hohpe; 2005). Café has become a benchmark in the studies and evaluation of integration platforms and it describes how customer orders are processed in a coffee shop. We have used Guaraná as the target integration platform, which means we have considered the conceptual model of Café designed in this platform and data from its runtime engine was used as input to *ModelGen*. Then, the simulation model generated and the number of threads found is tightly dependent to this integration platform. We have simulated this integration solution under different input rates for inbound messages to analyse its performance with variation on the number of threads and reported our experiments in a dedicated section.

The rest of this article has its structure organised as follows. In Section 2 we present our review on the literature for related works. Then, in Section 3 we provide a brief background on Queuing Theory, the theoretical model that abstracts the runtime engine of the message-based integration platforms considered in this article, and on the Guaraná integration platform. We then present in Section 4 our methodology and in Section 5 the supporting

tool names *ModelGen* to automate the generation of simulation models, followed by the demonstration of our proposal in a case of study in Section 6. Next, in Section 7 we present the experiments we have conducted; and, finally, in Section 8 we discuss our main conclusions.

2 Related Work

Our literature review has identified some works that look to provide an estimate on the optimum number of threads in pools, aiming at achieving best performance, without depending on the empirical knowledge of the software engineers. Muñoz and Ruspini (2014) propose a simulation method based on queuing theory an fuzzy logic to estimate the number of threads and to estimate the entrance in the queue and the service time, with values provided by software engineers. Our proposal differs from theirs in the type of queuing theory used, and in the approach to collect the entrance and service time, this values are taken as an input in our tool. The work by Ju et al. (2015) sought, through a prediction model, to determine the optimum number of threads to execute applications in heterogeneous systems and with multiple processors. The prediction model developed seeks to optimise the number of threads considering the behaviour and the characteristics of the architecture of the application. With the optimisation, the authors sought to adjust dynamically the process of mapping threads to many cores. Our proposal differs from theirs in that they sought to contribute for optimising the performance of multiple processors systems, while the present work seeks to optimise the runtime engine. Son and Wysk (2001) present a structure and an architecture for the automatic generation of simulation models, intended to be used for real-time simulation of factories control. This work differs from ours in that the architecture used by these authors was not based on a queue, and the models are built aiming only at the factories. Dancheva et al. (2016) describe the implementation of a tool for setting dynamically the number of threads in the OpenMP environment, based on the current state of the runtime. Machine learning techniques are used to find the number of threads and the decision of which number of threads will be used is defined at the time of execution. This work differs from ours in that it uses the machine learning technique to predict the optimum number of threads, while in our work it is necessary to create different simulation models, each simulation model with a specific number of threads and then execute the simulation models in the Simulink software. Jung et al. (2005) sought to maximise the performance of simultaneous multi-threaded execution (SMT) processors by using adaptive execution techniques in order to find the optimum number of threads automatically during the execution. To find the optimum number of threads, a build preprocessor generates a code that, based on dynamic feedback, determines the optimum number of threads at runtime. This work differs from ours because it seeks to maximise performance for only SMT processors, and uses dynamic execution techniques with dynamic feedback to automate the process of discovering the optimum number of threads. However, in our proposal, we try to find the optimum number of threads for the runtime engine, without considering the type of processor and to discover the number of threads it is necessary to execute computational simulations. Lee et al. (2010) present a dynamic system that automatically adjusts the number of threads in an application, in order to optimise the efficiency of the system. Using the dynamic compilation system, the authors developed a software called Thread Tailor, which combines threads by communication patterns to reduce synchronization overhead. Thread Tailor uses off-line analysis to predict the type of topics that exist at runtime and the communication patterns between them, based on the architecture, the dynamic state of the system, and the communication and synchronization relationships between threads. This work differs from ours in that the software developed by the authors seeks to reduce the overhead of thread synchronization, whereas in our proposal, we seek to find the number of threads using computational simulations.

3 Background

This section provides background information on key topics to understand our proposal.

3.1 Queuing Theory

In system modelling studies, it is common to have dimensioning problems in which the solution is complex. Usually, the objective is to dimension the correct quantity of equipment or resources, and also to optimise the system being studied. In computing systems, when there is a large accumulation of processes waiting to run in the queue, performance bottlenecks may arise. In this case, the scaling of the system is required. To assist with this process, the area of Queuing Theory arises (Klcinrock; 1975).Queues are expensive and when they become too large, performance bottlenecks may arise. A common approach to minimise performance problems generated by queues is to increase computational resources to run software systems. However, it brings costs for a company and so queue theory is an important tool to help understand and optimise the system most often avoiding this cost. A queue system consists of customers, arrival process, service process, number of servers, queue and queue discipline. Customers come from a population. The arrival process is defined by the behaviour of the customers arrivals in the system, which can be deterministic or stochastic. Service process follows the same concept as arrival process; it can be deterministic or stochastic. The server number represents the servers available to serve the customers that are waiting in queue. The queue is where customers wait until some server is available. The queue discipline defines the order in which customers are selected from the queue to the service (Prado; 2014).

3.2 Runtime Engine

The runtime engine is responsible for running integration solutions. Its core element is the Scheduler, which is responsible for coordinating all activities present in an execution. The Scheduler has a queue of tasks, a pool of threads, monitors and its own logging system. The queue is used to store tasks waiting to be processed and follows the FIFO discipline. The pool of threads represent a set of servers on the task queue. In this way, whenever new tasks enter in the queue, all the threads are notified, and then the competition to poll a task from the queue starts. The Scheduler must be provided with a configuration XML file, which includes information about the number of threads to be allocated to its pool, the type of monitors to be active to capture statistical data and the periodicity to capture this data, and the log file to report warnings and exceptions that may occur during the execution. The Scheduler starts by loading and parsing the configuration file, creating the indicated number of threads, activating required monitors and initialising the logging system (Frantz et al.; 2011).

3.3 Guaraná Integration Platform

Guaraná is the result of a joint research effort between academy and company to develop an innovative platform for application integration. This platform is currently in version 1.4 and was developed with the Java programming language. Guaraná follows the style of message-based integration and provides support for several integration patterns (Hohpe and Woolf; 2012) to implement a variety of atomic tasks. This platform allows the modelling, deployment, and monitoring of integration solutions using a graphical interface. This environment supports the following constructors: message, task, slot, port, and resources. Message is the information transmitted and transformed into the integration solution. Tasks are units that perform a certain processing on the messages flowing in the integration solution, such as filtering, copying, transforming, dividing, regrouping. Slots perform the connection between tasks, or between tasks and ports. Ports abstract the communication mechanism an integration solution uses to interact with the resources being integrated. Resources represent sources of information that are typically in applications or databases (Frantz et al.; 2011).

Figure 1 shows the interface of the designer where conceptual models for integration solutions can be elaborated. The modelling occurs in Panel (A), with tasks dragging from Panel (C). In Panel (A), software engineers can connect tasks, slots, and ports to form the integration solution. Panel (C) provides several tasks organised in groups, such as: router, modifier, and transformer. Panel (B) provides general functions, such as save, copy, past, print. The tasks and ports configuration takes place in Panel (D), which has elements to perform the configuration of the selected task. The graphic model developed can be exported to an Extensible Markup Language (XML) format. The resulting models

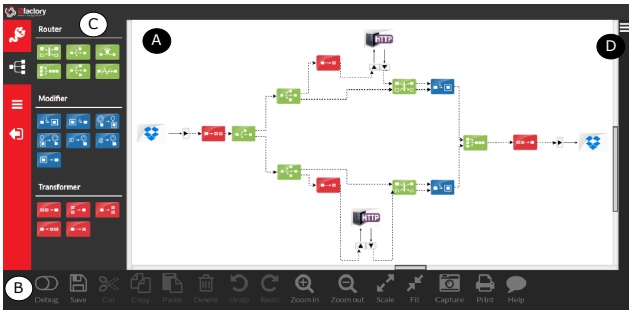

Figure 1: Guaraná user interface to design integration solutions.

obtained with Guaraná are platform-independent, so engineers do not need to have skills on a low-level integration technology when designing their solutions. Furthermore, this design can be re-used to automatically generate executable EAI solutions for different target technologies.

4 Methodology

This section introduces our methodology, which is divided into four main steps: conceptual modelling, model transformation, simulation, and analysis. Figure 2 provides an overview of this methodology.

In the conceptual modelling step, the workflow of the integration solution is developed by the software engineer using the graphic concrete syntax of the domain-specific language provided by the integration platform. This graphic model is transformed into a textual representation, based on the XML format. This feature is provided by the integration platform, which uses this format to internally store conceptual models in its repository. XML is a standard machine-readable format and is the base representation for the integration solution.

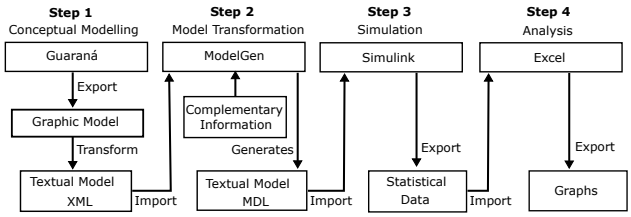

Figure 2: Overview of the proposed methodology.

In the model transformation step, the XML file is imported by *ModelGen* at its graphical interface. *ModelGen*, by means of model transformation, produces the corresponding simulation model for the target integration platform. The simulation model is represented using the Model Definition Language (MDL) format, which is the standard format used by Simulink Matlab (Higham and Higham; 2005). Simulink is a simulation tool integrated with Matlab which allows modelling, simulating, and analysing systems. Furthermore allows to model event-based systems with the SimEvents library and is the supporting software tool in the next step. The transformation of the XML model into the simulation

model occurs through the capture of all the tasks represented in the XML, so each task is transformed into an entity in the MDL model. Software engineers have also to provide complementary information to configure *ModelGen*. This information are provide through the graphical interface of the *ModelGen* tool and regards the number of threads to be experimented, the input rate for inbound messages to be simulated and the total duration time for the simulation under these rates.

The simulation step is performed using Simulink, and the software engineer only needs to import and run the simulation model generated in the previous step. There is no need to perform more settings for its operation. After executing the simulation model, results containing statistical raw data that can be exported and has to be analysed.

The last step concerns with the analysis of the raw data produced during the simulation. In our methodology we propose to perform this analysis using a spreadsheet, like Microsoft Excel (Winston; 2016), since exported data by Simulink can be imported into Excel and the graphs be straight generated for the analysis of the simulation behaviour.

5 Supporting Tool

In this section we introduce *ModelGen*, which is a Java-based software tool to automate the generation of simulation models taking as input the conceptual models exported by integration platforms.

The overview of the internal process of the tool responsible for generate the models can be visualised in the Figure 3. The process starts by reading a textual file in XML format, this file is loaded into the user interface of *ModelGen*, where additional information is also added. The textual model contains the tasks and their interconnections in the integration solution. Complementary information includes: simulation time, input rate, number of threads and execution time of each task. Simulation time is where users can inform the execution time for the simulation model generated. Input rate represents how many messages the integration solution receives for each unit of time. Number of threads determines the amount of threads that the simulation model will use in the experiments, the execution time of each task represents the total time to individually execute a task of the integration solution, and is calculated automatically by the tool.

Internally, on model generation, *ModelGen* performs the transformation of each task received in the textual model into the Model Definition Language (MDL) format, that is, each task is transformed into a block called entity. The model transformed into the MDL format is the model equivalent to the integration solution model. To performs the simulation, additional data that the user has informed are needed. Therefore, in this step the tool adds the execution times of each task, the input rate, the total simulation time and the number of threads.

In addition, the MDL model of the runtime engine is also added, this model is fixed independent of the integration solution received, because it contains elements that do not need to be modified. While

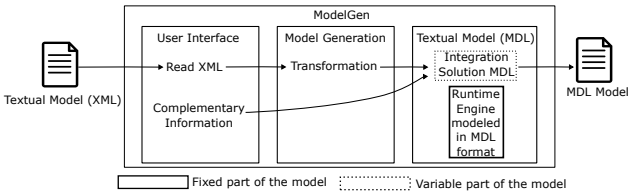

Figure 3: Overview of *ModelGen*.

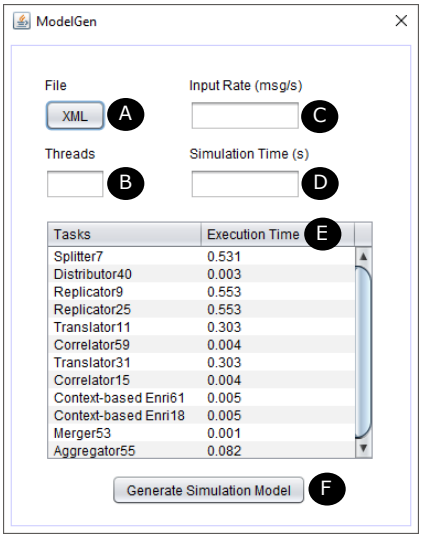

Figure 4: User interface of *ModelGen*.

the other parts are dependent on each integration solution.

Finally, the generated model is formed by the MDL model of the integration solution, in conjunction with the MDL model of the execution engine.

The main user interface of *ModelGen* can be seen in Figure 4. Component (A) has the function of opening a file selector in which the user can search the XML file containing the conceptual model of the target integration solution. In (B), it is possible to select the number of threads to be configured at the runtime engine to execute the integration solution. In (C), the input rate for inbound messages to the integration solution has to be provided. This value corresponds to messages per second. Field (D) allows for the specification of the simulation time, i.e., the amount of time to run the simulation.

After reading the XML, data is automatically displayed in (E), which is composed of the columns: tasks and execution time. Tasks represent the list of tasks present in the integration solution. The execution time represent the time that each task takes to execute in the simulation. This values have to be provided by software engineers and generally can be obtained by means of the monitoring tools provided by the integration platforms. *ModelGen* is able to store these values for an integration solution, and automatically fill this editable column in its future executions. Button Generate Simulation Model (F) generates the fully configured simulation file in MDL format. This file has to be imported and run in Simulink with no need of any further settings by the software engineer.

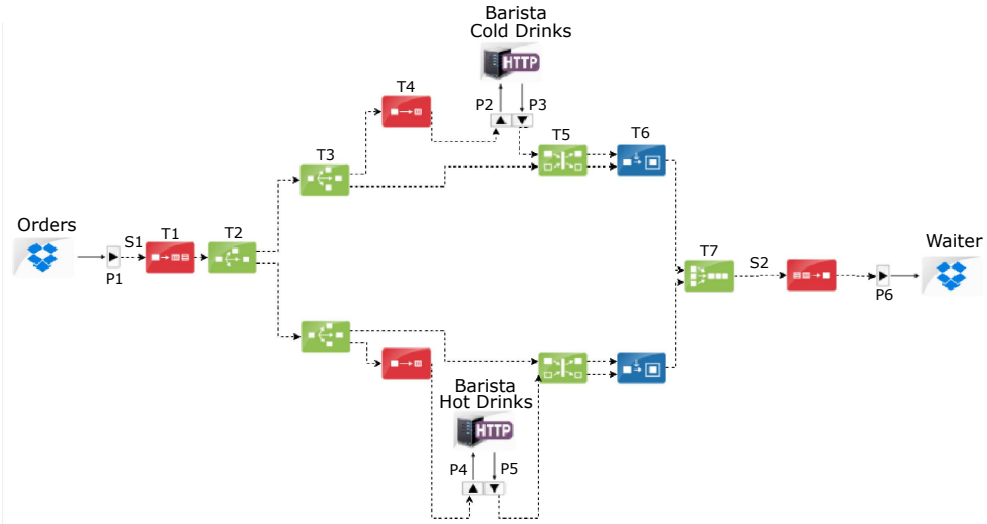

Figure 5: Conceptual model for the Café integration solution.

6 Case of Study

This section presents a case of study based on the Café (Hohpe; 2005) integration problem to demonstrate our proposal. Café illustrates how requests are processed in a coffee shop and has been used in the area of enterprise application integration to demonstrate the application and viability of proposals.

6.1 Context of the Integration Solution

The process begins with a customer making a request for the cashier, who then registers the order in the system and adds it to a queue of orders. An order can include hot and cold drinks which are prepared by different baristas. When all drinks that correspond to the same order have been prepared, they will be delivered by the waiter. Each order has a tray associated with it, which is used to deliver it to the customer.

6.2 Conceptual Model

The integration solution must be able to receive requests from the queue of requests, send requests for the baristas to prepare the appropriate drinks and notify the waiter when an order is completed. Figure 5 presents a conceptual model for the integration solution designed using the domain-specific language provided by Guaraná integration platform.

The integration solution starts at input port $P1$, which waits for new customer requests. Each order results in a message with the drinks to be prepared and the messages generated are added to slot $S1$. Task $T1$ is used to separate each message in several other messages so that it is possible to send the request to the correct barista. This means that the part of the message that contains the request for a hot drink is sent to the barista for hot drinks, and in the same way with cold drinks. After that, the messages are sent to task $T2$, which sends the messages to the correct destination. Task $T3$ replicates the messages to Cold Drinks Barista application, so that one copy can be sent to the barista and another copy to task $T5$, which correlates the barista's response to the copy on hold. Task $T6$ enriches the copy on hold with the information returned by the Cold Drinks Barista application. Task $T4$ transforms the messages into the format necessary for the Cold Drinks Barista application to understand them. Messages that are sent to the Hot Drinks Barista application behave the same way. After the preparation of the drink, the barista messages are gathered in a single slot $S2$ by the $T7$ merger task. The drinks prepared are then withdrawn from this slot and reassembled to a single message again, so that the output port $P6$ writes the resulting message to Waiter application.

6.3 Simulation Model

The simulation model automatically generated by *ModelGen* for Café integration solution is presented in Figure 6. This model is organised into a set of entities, a queue, a server, data extraction blocks, and graphics generators. For each integration solution, the simulation model varies the entities block, since they represent specific tasks in the integration solution.

Every entity has an attribute that must be configured with a value that determines the interval between the generation of two instances of that entity. In the simulation model, this value is calculated from the input rate provided at the user interface of *ModelGen*. In addition, two other attributes have to be configured in the entity blocks, they are: type and time of service. Attribute type is used to distinguish between one entity and another, and it also allows entities to be identified in other blocks of the simulation system. The attribute time represents the time that the servers will take to execute the entity. Both attributes are configured at *ModelGen*. Entities generated by entity blocks are sent to the Entity Input Switch block. This block has the function of sending the entities to the Work Queue block, which represents the task queue. The queue has an infinite size and it is

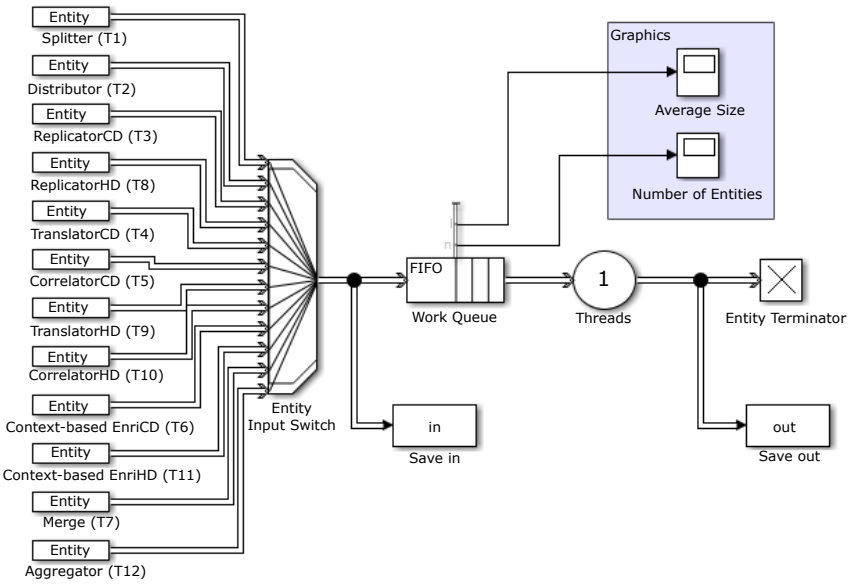

Figure 6: Simulation model generated by *ModelGen*.

possible to generate several types of graphs, which represent system statistics. This queue follows the FIFO discipline.

Threads block represents the servers, i.e., the threads available for the execution of the integration solution. The time that the entities remain in this block is previously defined in the attribute time. After entities are executed, this block sends the entities to the output, which occurs in the Entity Terminator block. The simulation model also includes two blocks called Save in and Save out, which are used to monitor the number of input and output messages in the model.

7 Experiments

In this section we report on the experiments we have conducted to evaluate our proposal. The conceptual model for the Café integration solution introduced in Figure 5 was implemented at Guaraná integration platform and run to collect execution data from the runtime engine to this integration solution. A textual model representation for the conceptual model was exported from Guaraná and taken as input by *ModelGen*, which have generated a corresponding simulation model to be performed in Simulink Matlab. In the following sections we describe our experiments and analyse the results found with Café running at Guaraná integration platform and its corresponding simulation model running at Simulink Matlab.

7.1 Research Question

We have selected three research questions we want to answer with our experiments. They are:

RQ1: What is the optimum number of threads to be configured in the runtime engine of Guaraná integration platform to run the Café solution under a given message input rate?

RQ2: Does the optimum number of threads to execute an integration solution increases by increasing the message input rate?

RQ3: By increasing the number of threads, the performance of the runtime engine increases as well?

7.2 Environment

The experiments were carried out on a machine equipped with 16 processors Intel Xeon CPU E5-4610 V4, 1.8 GHz, 32GB of RAM, and operating system Windows Server 2016 Datacenter 64-bits. Java SE version 8.0 update 152 was installed and is required to run Guaraná integration platform version 1.4. Simulink Matlab version R2016a was also installed to run the simulation. To minimise interferences, no other software was installed at the machine and it was disconnected from the Internet.

7.3 Variables

In this section we present the dependent and independent variables considered in our experiments, which are the following:

- Dependent: `number of messages processed`. This variable represents the number of messages processed by the Café integration solution and delivered by its exit port `P6` to the Waiter. This variable was measured in the execution of the actual integration solution at Guaraná and in the execution of the corresponding simulation model at Simulink Matlab.
- Independent: `running time`, `input rate`, `number of threads`. Running time represents the amount of time each experiment with the Café integration solution run at Guaraná and the time considered for running the simulation. We have considered a fix amount of 120 seconds for this variable all over our experiments. Input rate represents the number of messages which have been injected to

Threads	5000 msg/s		6000 msg/s		7000 msg/s		8000 msg/s		9000 msg/s		10000 msg/s	
	Plat.	Sim.	Plat.	Sim.	Plat.	Sim.	Plat.	Sim.	Plat.	Sim.	Plat.	Sim.
1	29738	18685	29786	20586	28901	19667	27650	17559	17320	21658	13646	20422
3	**46450**	**79267**	**65822**	**89961**	52084	58459	43023	71873	32746	72178	25847	75322
6	40225	72726	46200	81064	**66614**	**82217**	**83618**	**75601**	**35479**	**79025**	28308	**81500**
9	33598	67633	49322	67642	63253	67643	81525	67648	31768	60460	**31767**	73366
12	34669	69354	49642	60237	61618	73145	77828	57856	24544	56132	27182	54828
15	34984	64341	50897	62138	62451	73540	65722	62155	24500	58026	26695	57862
18	35439	61525	50149	74186	63238	70816	67154	64284	24499	59134	26013	61249
21	36165	59830	49184	69766	62484	65212	67159	64434	24499	55430	29482	55430
24	36119	59365	48680	70735	62246	64223	68592	61909	24449	56584	29187	55156

Table 1: Number of messages processed at the integration platform and the simulation.

the integration solution per second. The input rates considered were 5000, 6000, 7000, 8000, 9000, and 10000 messages per second (msg/s). With rates lower than 5000 msg/s the integration solution processes all injected messages using only one thread, so they were not considered. Number of threads represent the number of threads available to run the runtime engine, the number of threads chosen for the experiments were: 1, 3, 6, 9, 12, 15, 18, 21, and 24. The combination of input rates and the number of threads allowed us to measure the number of messages processed in 54 different experiments.

7.4 Execution and Data Collection

In both, Guaraná and Simulink Matlab, each experiment out of the 54 was repeated 25 times. According to Grinstead and Snell (2012), when an experiment is repeated a large number of times, as the number of repetitions increases, the sample mean of the variables approach the population mean. Usually, the population mean is found with approximately 20–30 repetitions (Sargent; 2010). Outliers in these executions were removed using Tukey method (Tukey; 1977). A first set of 225 executions were used to warm-up the Java Virtual Machine, and their values for the dependent variable discarded, since according to Pinto et al. (2014), the first executions tend to be considerably slower than the later ones since the Java Just-in-time compiler collects data and decide on the possible optimisations.

Data was collected at Guaraná integration platform using active monitors, which after the running time establish for the experiment stored the number of processed messages at log files. These log files were imported into Excel, allowing their visualisation and graph generation. The data collection for the executions of the simulation model in Simulink Matlab occurs in a similar way by means of building blocks that represent monitors.

7.5 Results

The results regarding the number of messages processed by Café running at Guaraná integration platform and its corresponding simulation model running at Simulink Matlab are provided in Table 1. Each value in this table is the average computed for the 25 repetitions in each combination of input rate and thread number, excluding possible outliers. The Guaraná integration platform results are flagged as "Plat" and the simulation results are flagged as "Sim". Values highlighted in bold represent the highest number of messages processes in each input rate.

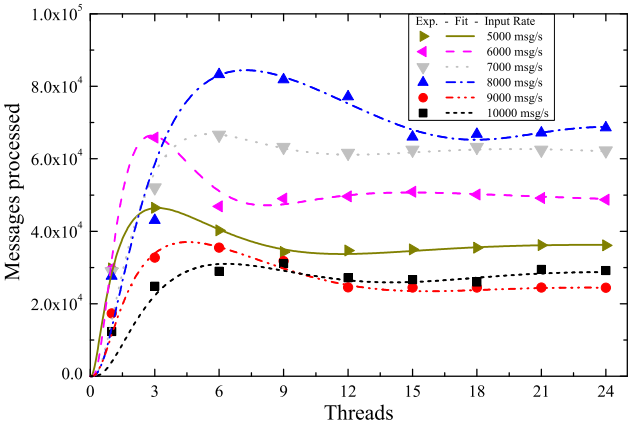

Figure 7: Number of messages processed by the integration platform.

The average number of messages processed by Café running at Guaraná integration platform in the 54 experiments is presented in Figure 7. The x-axis represents the number of threads, and the y-axis the number of messages processed in each input rate. Figure 8 presents the average number of messages processed in the simulation. The x-axis represents the number of threads, and the y-axis the number of messages processed. Fits of the experimental data shown by lines are also present (more detail below).

Analysing experimental data from Figures 7 and 8 were observed peaks of messages processed by specific threads. With input rate of 5000 msg/s, the highest number of messages processed with the integration platform and simulation were obtained using 3 threads, 46450 and 79267, respectively. The same happens with input rate of 6000 msg/s, the highest number of messages processed is still obtained with 3 threads, counting 65822 messages for the integration platform and 89961 messages in the simulation. For input rates of 7000, 8000, and 9000 msg/s, the highest number of messages processed was always reached using 6 threads. With this number of thread and an input rate of 7000 msg/s, 66614 messages were processed by the integration platform and 82217 messages in the

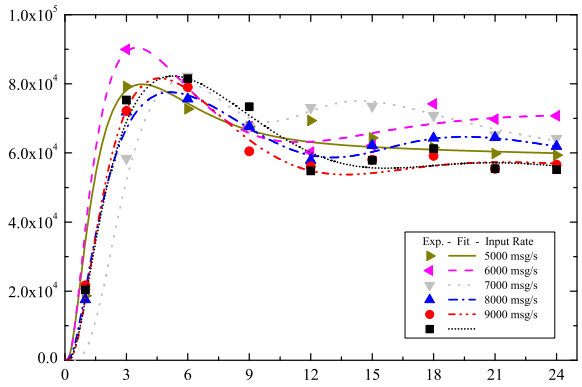

Figure 8: Number of messages processed in the simulation.

simulation; with an input rate of 8000 msg/s, 83618 messages were processed by the integration platform and 75601 in the simulation; and, with an input rate of 9000 msg/s, 35479 messages were processed by the integration platform and 79025 messages in the simulation.

When experimenting the integration platform and the simulation with an input rate of 10000 msg/s, the highest number of messages processed by the integration platform was reached with the use of 9 threads, counting 31767 messages. However, in the simulation the highest number of messages processed was reached using 6 threads, counting 81500 messages. In this input rate, the optimum number of threads required to run the integration solution differs, probably due to external interference on the environment in which the integration platform runs. Although we have repeated the execution, since this repetition is automatically programmed and execution only takes 120 seconds, and execution of other software process at the operating system may caused this interference. In all other input rates experimented, the optimum number of threads was always the same at the integration platform execution and the simulation.

The lines in Figures 7 and 8 were obtained fitting the experimental data using the sum of two distributions. The first one was a log-normal function used to identify the peak of processing of messages. The second one was a logistic function used to characterise the saturation of processing of messages. The parameters of each function were restricted at data range observing the trend empirically. A scan of all set possibilities was performed in order to obtain those resulted in the smallest difference between experimental and fitted data. The most efficient Threads obtained by the fit were (Plat. and Sim. for input rate): 3 and 4 for 5000 msg/s, 3 and 4 for 6000 msg/s, 6 and 6 for 7000 msg/s, 7 and 5 for 8000 msg/s, 5 and 5 for 9000 msg/s, and 6 and 5 for 10000 msg/s.

7.6 Discussion

Our experiments with the integration platform and the simulation model demonstrate there is a partial relation between the number of threads and the number of messages processed. In every input rate considered, increasing the number of threads has a positive impact on the number of messages processed up to a certain number of threads. Then, keeping increasing threads do not result in the increasing of processed messages as well. Adding more threads after that number of threads in which the highest performance in terms of message processing was found, leads to a deterioration of the overall performance, i.e., a reduction of the number of messages processed in every input rate experimented. This behaviour may be related with the time that the Java Virtual Machine takes to perform the context switch and manage the threads, such as saving local data and the program pointer of the current thread execution, and loading the local data and the program pointer of the next thread to be executed (Pusukuri et al.; 2011). Keeping increasing the number of threads shows this degradation persists also up to a certain number of threads and then our experiments demonstrate that finally, from this point by increasing the number of threads there is no more positive or negative impact on the performance.

It was also observed that by increasing the input rate, the number of messages processes tend to decrease. This occurs because there will be more occurrences of tasks from the beginning of the integration flow in the FIFO queue to be executed by the threads. This behaviour also leads to an increment of the amount of time a message takes to complete the integration flow.

Despite the number of messages processed differs in the execution of Café at Guaraná integration platform and its corresponding simulation model running at Simulink Matlab, it was possible to validate the simulation model since in every input rate the number of threads that give the best performance is the same for the integration platform and the simulation.

It is observed that the experimental results of more efficient Thread did not match perfectly with the adjusted results but indicated the behaviour. We argue that a challenge is to get more processed message data in function on the number of threads. Thus the fits can express with more reality the experimental data. Nevertheless, the fits are presented as the beginning of the search for a mathematical model able to foresee the behaviour of the number of messages processed as a function of the threads.

7.7 Threats to Validity

The integration platform we have considered in our experiments requires Java Virtual Machine to run, and this machine runs over Windows operating system. Thus, any system process that runs at the same time that the Java Virtual Machine process is likely to impact on the running experiments. This can be more critical if the concurrent system process runs and requires computing resources, such as disk access or demands high processing CPU. These will have a direct impact on the number of messages processed and consequently affects the optimum number of threads. Simulation performed with Simulink Matlab is less likely to be influenced by

external system processes, since the simulation runs on a virtual environment and does not have interference in the actual runtime environment such as for the Java Virtual Machine. The simulation model could also be improved by considering other aspects, chiefly related with the actual hardware in which the integration platforms runs, which probably affect the number of messages processed. The results obtained are also dependent on the integration solution and the integration platform, so it is not possible to generalise the results to other solutions and to other platforms. The approach we have taken in this article is static, which means we analyse the integration solution for a given integration platform not during its execution. Our proposal works on data that must be extracted from a previous execution of the integration solution. This characteristics make the proposal tightly coupled with the hardware in which the integration platform is running and so the data extracted is valid only for the optimisation of the number of threads in that hardware and with that integration platform.

8 Conclusions

With the growth in the use of enterprise applications in companies, there is an increasing need for integration amongst these applications. This integration can be performed through different platforms. The integration platforms run integration solutions in runtime engines. Runtime engines are important for the performance of the solutions, and therefore, finding the optimum number of threads for the execution of these engines has great importance. In this article we have presented a methodology and a tool to assist with the generation of simulation models based on queuing theory, in order to find the optimum number of threads to execute an integration solution focusing performance improvement. The use of this simulation model can be used to empower software engineers when configuring the runtime engine of their integration platform to improve the performance of an integration solution and replaces the need of empirical knowledge, which may bring risks. *ModelGen* also allows to know the limits of an integration solution, for example, if the software engineer has a limited hardware available our approach will assist him to find out the maximum supported input rate for a given integration solution running on that hardware.

Experiments were conducted running Café integration solution at Guaraná integration platform and its corresponding simulation model at Simulink Matlab. The methodology and our tool support o generate simulation models to represent the behaviour of a runtime engine of an integration platform was validated and the optimum number of threads to run and get the best performance on an integration solution was found the same in the experiments with the integration platform and in the simulation. Thus, from now on, *ModelGen* can be used to forecast the optimum number of threads for integration solutions at Guaraná integration platforms in a static way. Other experiments can be conducted to analyse the applicability of *ModelGen* for other integration platforms, such as Camel, Spring Integration, Mule, Petals, Apache Flume, and Apache Nifi. A dynamic approach has also to be investigated, since it would make our tool hardware and platform independent. Maybe this is the most interesting and challenging step forward to improve *ModelGen*.

Acknowledgements

This work was partially supported by the Brazilian Co-ordination Board for the Improvement of University Personnel (CAPES) under grant number 88881.119518/2016-01, the Research Support Foundation of the State of Rio Grande do Sul (FAPERGS) under grant number 17/2551-0001206-2, and the Brazilian National Council for Scientific and Technological Development (CNPq).

References

Chaturvedi, D. (2017). *Modeling and simulation of systems using MATLAB and Simulink*, CRC Press.

Dancheva, T., Gusev, M., Zdravevski, V. and Ristov, S. (2016). An OpenMP runtime profiler/configuration tool for dynamic optimization of the number of threads, *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 192–197. http://dx.doi.org/10.1109/MIPRO.2016.7522136.

Dossot, D., D'Emic, J. and Romero, V. (2014). *Mule in action*, Manning.

FLU (2017). Apache Flume, https://flume.apache.org. Last accessed on 03/09/2018.

Frantz, R. Z., Corchuelo, R. and Roos-Frantz, F. (2016). On the design of a maintainable software development kit to implement integration solutions, *The Journal of Systems and Software* **111**(1): 89–104. http://dx.doi.org/10.1016/j.jss.2015.08.044.

Frantz, R. Z., Quintero, A. M. R. and Corchuelo, R. (2011). A domain-specific language to design enterprise application integration solutions, *International Journal of Cooperative Information Systems* **20**(02): 143–176. http://dx.doi.org/10.1142/S0218843011002225.

Freire, D. L., Frantz, R. Z. and Roos-Frantz, F. (2019). (in-press). Ranking enterprise application integration platforms from a performance perspective: An experience report, *Software: Practice and Experience* pp. 1–21. http://dx.doi.org/10.1002/spe.2679.

Freire, D. L., Frantz, R. Z., Roos-Frantz, F. and Sawicki, S. (2019). Survey on the run-time systems of enterprise application integration platforms focusing on performance, *Software: Practice and Experience* **49**(3): 341–360. http://dx.doi.org/10.1002/spe.2670.

Grinstead, C. M. and Snell, J. L. (2012). *Introduction to probability*, American Mathematical Soc.

Higham, D. and Higham, N. (2005). *MATLAB guide*, Society for Industrial and Applied Mathematics - SIAM.

Hohpe, G. (2005). Your coffee shop doesn't use two-phase commit [asynchronous messaging architecture], *IEEE software* **22**(2): 64–66. http://dx.doi.org/10.1109/MS.2005.52.

Hohpe, G. and Woolf, B. (2012). *Enterprise integration patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional.

Ibsen, C. and Anstey, J. (2017). *Camel in action*, Manning Publications Co.

Ju, T., Wu, W., Chen, H., Zhu, Z. and Dong, X. (2015). Thread Count Prediction Model: Dynamically adjusting threads for heterogeneous many-core systems, *IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 456–464. http://dx.doi.org/10.1109/ICPADS.2015.64.

Jung, C., Lim, D., Lee, J. and Han, S. (2005). Adaptive execution techniques for smt multiprocessor architectures, *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 236–246. http://dx.doi.org/10.1145/1065944.1065976.

Klcinrock, L. (1975). *Queueing systems, vol. 1: theory*, New York: Wiley.

Lee, J., Wu, H., Ravichandran, M. and Clark, N. (2010). Thread tailor: dynamically weaving threads together for efficient, adaptive parallel applications, *ACM SIGARCH Computer Architecture News*, pp. 270–279. http://dx.doi.org/10.1145/1816038.1815996.

Lindholm, T., Yellin, F., Bracha, G. and Buckley, A. (2014). *The Java virtual machine specification*, Pearson Education.

Muñoz, E. and Ruspini, E. (2014). Simulation of fuzzy queueing systems with a variable number of servers, arrival rate, and service rate, *IEEE Transactions on Fuzzy Systems* **22**(4): 892–903. http://dx.doi.org/10.1109/TFUZZ.2013.2278407.

NIF (2017). Apache Foundation, https://nifi.apache.org. Last accessed on 03/09/2018.

Pandey, C. (2015). *Spring Integration Essentials*, Packt Publishing Ltd.

Pinto, G., Castor, F. and Liu, Y. D. (2014). Understanding energy behaviors of thread management constructs, *ACM SIGPLAN Notices*, pp. 345–360. http://dx.doi.org/10.1145/2660193.2660235.

Prado, D. (2014). *Teoria das Filas e da Simulação*, Editora de Desenvolvimento Gerencial.

Pusukuri, K. K., Gupta, R. and Bhuyan, L. N. (2011). Thread reinforcer: Dynamically determining number of threads via os level monitoring, *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 116–125. http://dx.doi.org/10.1109/IISWC.2011.6114208.

Ritter, D., May, N. and Rinderle-Ma, S. (2017). Patterns for emerging application integration scenarios: A survey, *Information Systems* **67**(Supplement C): 36–57. http://dx.doi.org/10.1016/j.is.2017.03.003.

Sargent, R. (2010). *A new statistical procedure for validation of simulation and stochastic models*, L.C. Smith College of Engineering and Computer Science.

Schildt, H. (2017). *Java: a beginner's guide*, McGraw-Hill.

Son, Y. J. and Wysk, R. (2001). Automatic simulation model generation for simulation-based, real-time shop floor control, *Computers in Industry* **45**(3): 291–308. http://dx.doi.org/10.1016/S0166-3615(01)00086-0.

Surhone, L., Timpledon, M. and Marseken, S. (2010). *Petals Enterprise Service Bus (Esb)*, Betascript Publishing.

Tukey, J. W. (1977). *Exploratory data analysis*, Mass. Reading.

Winston, W. (2016). *Microsoft Excel data analysis and business modeling*, Microsoft press.