On the impact of event-driven architecture on performance: An exploratory study

Hebert Cabane[a], Kleinner Farias[a]

[a]*PPGCA, University of Vale do Rio dos Sinos (Unisinos), Av. Unisinos, 950, Sao Leopoldo, RS, Brazil*

Abstract

Event-driven architecture has been widely adopted in the software industry in recent years. This adoption is motivated by the increase in software modularity, and mainly the desired performance of decomposed monolithic applications. Although it has become popular, the current literature lacks studies that demonstrate the impact of event-driven architecture on performance. Hence, developers and architects end up adopting event-driven architecture without empirical evidence, but considering "expert advice." This study, therefore, reports an empirical study on the impact of the adoption of event-driven architecture on performance. For this, the performance of an application implemented with an event-based architecture was compared with the performance of the same application implemented using monolithic architecture. The comparison was made using metrics such as CPU usage, memory, response time, throughput, and total packages sent and received. The results, supported by statistical tests, show that the monolithic architecture, compared to the event-driven architecture, consumes less computational resources and has better response times. Finally, this study reflects on the adoption of event-driven architecture, as well as points out challenges and implications that need to be considered by the scientific community in future research.

Keywords: Event-driven architecture, performance, empirical study, distributed applications, software modularity, monolithic application

1. Introduction

Over the past decade, enterprise application development has taken place in increasingly unstable business environments in the software industry. Volatile software requirements, constant changes in technology and business strategies, pressures for the fast delivery of functional software [1], and applications with tightly coupled modules are features that are always present in most projects in the real world [2]. In this unstable context, companies have been looking for alternatives to minimize problems in the development and maintenance of such applications, for example, the high maintenance cost and complexity of the applications. The adoption of event-driven architectures would be one of these alternatives, which aims to enhance maintenance by decomposing monolithic applications into independent modules that communicate through events.

Some recent works highlight the decomposition of monolithic applications into applications based on event-driven architectures [3] as a reality and necessity. In practice, the industry seeks to produce less coupled and maintenance-friendly applications aiming to meet market demands. Laigner *et al.* [4] reinforce by performing the decomposition of a monolithic application into an event-driven architecture one as they seek to tame the complexity and high cost to maintain the monolithic application. Given this trend toward the use of event-driven architecture, some works have been carried out over the past few years.

Urdangarin *et al.* [5] investigate the use of multi-objective and context-based decomposition techniques. Schipor *et al.* [6] carried out an empirical study using event-driven architecture in intelligent environment applications. Laigner *et al.* [4] present the obtained results during the decomposition of a monolithic big data application into an event-driven application. Tragatschnig *et al.* [7] carried out an empirical study to evaluate the use of change patterns in the development of event-driven applications. Djogic *et al.* [8] presented the benefits of rebuilding a monolithic application into an event-driven application. Bukhsh *et al.* [9] carried out a comparison through literature studies between service-oriented and event-driven architectures. Pienwittayasakul and Liu [10] presented the benefits of using event-driven architecture over service-oriented architecture.

Currently, the adoption of event-driven architecture has not been based on empirical evidence. Rather, it has been done following expert opinions or heuristics created based on the experience of the development teams. Today, the current literature also lacks empirical studies that report the impact of event-driven architecture on quality criteria, such as performance. This work, therefore, reports an exploratory study on the impact of event-driven architecture on performance. A realistic empirical study was carried out to evaluate the performance of two versions of the same application, one implemented using a monolithic architecture and the other implemented using an event-driven architecture. The comparison was made using metrics such as CPU usage, memory, response time, throughput, and total packages sent and received. This was done through an eight-step process organized into four phases.

The results, supported by statistical tests, show that the monolithic architecture, compared to the event-driven

architecture, consumes less computational resources and has better response times. The results obtained in this work showed that a monolithic architecture consumes computational resources more efficiently, and has better response times compared to the event-driven architecture. These results benefit professionals interested in the field of software architecture such as software architects, software developers, and researchers.

This work is organized as follows: Section 2 presents the theoretical foundation of the main concepts needed to understand this study. Section 3 3 presents a comparative analysis of related works. Section 4 outlines the study methodology. Section 5 describes the analysis of the obtained results. Finally, Section 6 presents the final considerations and upcoming works.

2. Background

This section presents the main concepts for understanding the work developed.

2.1. Decomposition of monolithic applications

Decomposition can be understood as a sequence of tasks that are performed to break a monolithic application into modules, generating a non-monolithic application [5] with several interdependent modules. Essentially, both applications are equivalent, having the same functionalities, but they are structurally distinct [5]. In an application with a monolithic architecture, all functionalities are available in a single artifact. Typically, this unit is restricted to a single executable software artifact. For example, traditional monolithic applications are typically composed of a web interface, a domain layer, and a data access layer. In a monolithic architecture, these layers are combined into a single instance of the application [5]. Generally, monolithic architectures are suitable for small applications, but they can become complex as the application grows. Initially, a small and simple application can become a complex and difficult application to maintain because all functionalities are centralized in a single artifact. Several challenges are encountered as business rules and application load increase such as difficulty in scaling the size of existing features, the complexity of developing new features, the limitation in the use of new technologies, sharing work across teams, and the release and implementation of new versions. These challenges can be minimized by applying decompositions.

2.2. Event-driven architecture

Event-driven architecture is an architectural pattern that is composed of single-purpose decoupled components that asynchronously receive, process, and transmit events [11]. The event-driven architecture advocates the communication of its components through the producer/consumer model, with producers having the responsibility to publish events and consumers to subscribe to and consume events [12]. Through event-based communication, a barrier is established that ensures the loose coupling and isolation of the components of the event-driven architecture. Through isolation it is possible to manage each component independently, making it possible to control aspects of load control, elasticity, and monitoring, among other aspects of each component individually [13].

2.3. Performance

Performance is one of the vital characteristics of software quality. If we improve it, positive effects on the software quality can be obtained, in addition to offering a better user experience. Performance can be briefly defined as the number of responses of an application while executing its functionalities in a given time interval, and by the capacity to consume resources such as CPU and memory appropriately while processing the functionality under the established conditions [14]. Resources are defined as CPU (processor) and memory (RAM). Performance is composed of the following characteristics:

- **Behavior over time:** Response measures and processing time, and transfer rates of an application during the execution of a feature.
- **Use of resources:** Measures the quantity and resource types used by an application during the execution of a feature.
- **Capacity:** Maximum limit for an application to execute a feature.

If the source code of an application is optimized, it can have good effects in terms of performance, such as reducing response time, increasing the number of responses, reducing CPU and memory usage, and network consumption [14]. Performance is also known as efficiency and is part of the quality attributes, described by ISO/IEC 25010 [15], such as functionality, efficiency, compatibility, usability, reliability, security, maintenance, and portability. Quality attributes provide a basis for measures and forms of verification to assess software quality [16].

3. Related Work

This section presents an analysis of related works addressing the use of monolithic architecture and event-driven architecture. The selected works are presented in Section 3.1. Comparative analysis and research opportunities are discussed in Section 3.2.

3.1. Related Work analysis

Laigner *et al.* (2020) [4] presented some results considering the redesign of a big data truck fleet monitoring application for an oil and gas industry. The initial architecture of the application was of the monolithic type, which was rebuilt during the project using the event-driven microservices architecture. The reconstruction aimed to facilitate the maintenance of the application due to the complexity and obsolete codes that the application had. With the event-driven architecture of microservices, some expected results

were perceived, such as greater ease of in-service maintenance and improved service failure control, but due to the complex data flow generated by the number of microservices, it was perceived as a disadvantage by the authors.

Schipor *et al.* (2019) [6] performed an empirical study using the EUPHORIA[1] software architecture developed by the authors. This new event-driven architecture aims to orchestrate communication between different devices in a smart environment. The study was conducted in three distinct smart environment scenarios using the new event-driven architecture, demonstrating in practice the use of the architecture in different types of environments. Finally, the result of the empirical study showed that the processing time of an event can be impacted by the size of messages or by the complexity of the smart environment, that is, by the number of connected consumers.

Tragatschnig *et al.* (2018) [7] carried out an empirical study on how change patterns can influence the performance of changes in an application with event-driven architecture. For this, the authors selected 4 applications based on real industry context and three change pattern sets. Thus, the experiment was conducted with 90 students divided into three groups, each group was in charge of executing a set of specific change patterns in the previously selected applications. With the result of the experiment, it was possible to identify that change patterns with less primitive operations require less time than change patterns with primitive operations.

Djogic *et al.* (2018) [8] presented the benefits of rebuilding an application with a monolithic service-oriented architecture into an event-driven microservices architecture, the work was done in a real-estate application. The application's reconstruction in a new architecture was performed due to the increase in the number of messages that are processed by the application and to support new integrations with it. Finally, the authors reported that the reconstruction using the event-driven microservices architecture made it possible to solve several problems that the monolithic application had, in addition to simplifying some processes such as the individual maintenance of the application components, publishing only what was changed, scalability by service and utilization of resources.

Bukhsh *et al.* (2015) [9] performed a comparative study between service-oriented architecture and event-driven architecture, an event-driven service-oriented architecture was also considered in the study. This case study addressed the use of each architecture studied in a Learning Management Application (Moodle), with this it was possible to identify the disadvantages of each architecture. Finally, the study concluded that both architectures have different characteristics, mainly how communications between services are performed, and that the choice of architecture will depend on the business of the application and how it should react to certain situations, such as transaction control, type of communication between services, data redundancy, and user responses, and other factors.

Pienwittayasakul and Liu (2014) [10] carried out a comparative study of the definitions and characteristics of service-oriented and event-driven architectures, and the relationship of the architectures to each other was also analyzed. The comparison between the architectures was performed in four categories: how the architectures react to the application's business, how each service that makes up the application is distributed, how each application service communicates, and how each architecture manages the information generated by the application. Finally, the authors concluded that service-oriented architecture and event-driven architecture interact in several areas, such as event execution, where simple or complex business operations are performed, and that event-driven architecture is not just an implementation of service-oriented architecture, both architectures are peers and complement to the business and IT context.

3.2. Comparative Analysis and Research Opportunities

The comparative analysis was performed based on criteria, because other works already published [1, 17] that used this approach, showed that this is an effective way to generate a comparison between works and identify research opportunities. With the performance of the comparative analysis, it was possible to identify points that are similar and different between the current work and the selected related works. This comparison was necessary to objectively identify research opportunities, and for this, six Comparison Criteria (CC) were selected, which are described below:

- **Exploratory Study (CC01):** Works that carried out the study in an exploratory manner.
- **Performance Attributes (CC02):** Studies that consider performance or efficiency attributes.
- **Metrics (CC03):** Studies that used metrics to analyze resource use efficiency and to assess application performance.
- **Event-Driven Architecture (CC04):** Studies where event-driven architecture was part of the central research theme.
- **Monolithic Architecture (CC05):** Studies where monolithic architecture was compared with other types of architecture.
- **Application Context (CC06):** This criterion addresses the studies that performed their research on real applications.

The result of the comparison of related work and the proposed work based on the Comparison Criteria is shown in Table 1. Analyzing the result of the comparison, the following research opportunities were identified:

- only a few works carried out the study empirically on event-driven architecture;
- no work explored the performance attributes of applications that use event-driven architecture;

[1] http://www.eed.usv.ro/mintviz/resources/Euphoria/

Table 1: Related work comparative analysis.

Related work	Comparison Criteria					
	CC1	CC2	CC3	CC4	CC5	CC6
Current work	●	●	●	●	●	●
Laigner *et al.* (2020) [4]	○	○	○	●	●	●
Schipor *et al.* (2019) [6]	●	◐	◐	●	○	○
Tragatschnig *et al.* (2018) [7]	●	○	○	●	○	◐
Djogic *et al.* (2018) [8]	○	◐	○	●	●	●
Bukhsh *et al.* (2015) [9]	○	○	○	●	●	○
Pienwittayasakul and Liu (2014) [10]	○	○	○	●	●	○

● Similar ◐ Partially similar ○ Not similar

- no work has explored the use of metrics to assess and quantify the use of event-driven architecture.

Research opportunity. Based on the research opportunities indicated, the following research opportunity was identified: carrying out empirical studies to understand the impact of event-driven architecture on performance. This research opportunity is explored in the following sections.

4. Study methodology

This section presents the main decisions underlying the experimental design of our exploratory study. Section 4.1 presents the objective and research questions. Section 4.2 introduces the formulation of hypotheses based on research questions. Section 4.3 describes the context and domain of the application used in the study. Section 4.4 presents the considered quantitative variables. Section 4.5 demonstrates the process of experiment. Finally, Section 4.6 presents the analysis procedures.

4.1. Objective and Research Questions

This study essentially seeks to grasp the effects of event-driven architecture and monolithic architecture on performance. These effects are investigated from concrete evolution scenarios involving real-world applications so that empirical results can be generated. Our experimental procedures follow empirical strategies previously successfully applied in previous studies [18, 19, 20, 21, 22]. With this in mind, the objective of this study is stated based on the GQM template [23] as follows:

Analyze *architectural styles* **for the purpose of** *investigate its effects* **with respect to** *performance* **from the perspective of** *software architect* **in context of** *software development*

In particular, this study focuses on evaluating the performance impact of using event-driven architecture and monolithic architecture. Thus, the study focuses on the following research question:

- **RQ01:** Is the performance of event-driven architecture greater than the performance of monolithic architecture?

4.2. Hypothesis formulation

We conjecture that the use of event-driven architecture can improve performance as a superior modularization of software concerns can reduce the consumption of computation resources [5]. Perhaps, a better modularization can bring performance benefits, due to the efficiency related to the number of resources an application consumes and the code needed to execute an operation under certain conditions [14]. A better modularization can reduce the use of resources, converting to better performance. We also conjecture that although event-driven architectures can provide a more systematic way to modularize architectural components, the structure to support it can require additional code and configuration files, jeopardizing performance issues. In this sense, event-driven architectures may require much more structures to support the modularizations at the expense of performance gains.

This hypothesis assesses whether the performance of event-driven architecture is different from the performance of monolithic architecture. The hypothesis is described as follows:

> **Null Hypothesis 1, H_{1-0}:** The performance of event-driven architecture is the same as the performance of monolithic architecture.
> **H_{1-0}:** Performance(event-driven architecture) = Performance(monolithic architecture)
> **Alternative Hypothesis 1, H_{1-1}:** The performance of event-driven architecture is different from the performance of monolithic architecture.
> **H_{1-1}:** Performance(event-driven architecture) ≠ Performance(monolithic architecture)

H1 was refined into a set of 6 sub-hypotheses, each sub-hypothesis is represented by a performance metric, as presented as a dependent variable in Section 4.4. The formulation of the sub-hypotheses can be seen in Table 2. By carrying out the hypothesis tests, empirical knowledge can be produced about the impacts on performance as a result of the use of event-driven architecture and monolithic architecture.

4.3. Target Application

For the execution of the experimental and exploratory study, it was defined that the independent variable (Section 4.4) should have the following possible values: event-driven architecture and monolithic architecture. Then, for

Table 2: The investigated hypotheses in our study.

Null Hypothesis		Alternative Hypothesis	
H_{1-0}:	Performance(eda) = Performance(mono)	H_{1-1}	Performance(eda) ≠ Performance(mono)
H_{2-0}:	CPU Usage(eda) = CPU Usage(mono)	H_{2-1}	CPU Usage(eda) ≠ CPU Usage(mono)
H_{3-0}:	Memory(eda) = Memory(mono)	H_{3-1}	Memory(eda) ≠ Memory(mono)
H_{4-0}:	Received Packages(eda) = Received Packages(mono)	H_{4-1}	Received Packages(eda) ≠ Received Packages(mono)
H_{5-0}:	Transmitted Packages(eda) = Transmitted Packages(mono)	H_{5-1}	Transmitted Packages(eda) ≠ Transmitted Packages(mono)
H_{6-0}:	Response Time(eda) = Response Time(mono)	H_{6-1}	Response Time(eda) ≠ Response Time(mono)
H_{7-0}:	Throughput(eda) = Throughput(mono)	H_{7-1}	Throughput(eda) ≠ Throughput(mono)

Legend: *EDA* event-drive architecture, *MONO* monolithic architecture

each architecture, a target application was selected for the execution of tests and data collection of selected metrics for data collection through monitoring tools. The collected data will be analyzed and submitted to hypothesis tests. Each selected application essentially corresponds to the same application having the same developed functionalities, differing only in the architectural style, that is, the application of event-driven architecture represents the decomposition of the application of monolithic architecture. The target application has the following features of an e-commerce website: product catalog, shopping cart, order closing, order history, customer registration, and authentication. The features are described in Table 3.

Table 3: Target application features.

Feature	Description
Product Catalog	Responsible for providing the list of products available for sale by the application
Shopping Cart	Responsible for temporarily storing the data and quantity of products selected for purchases by customers
Order Closing	Responsible for closing and billing Customer Orders
Order History	Responsible for providing the list of orders placed by customers
Customer base	Responsible for providing the necessary resources to register customers in the application
Authentication	Responsible for providing the necessary resources to authenticate previously registered customers

The target applications were developed and are currently maintained by the organization *Dotnet Foundation*[2], this group is also responsible for promoting the platform *.Net*[3] in the developer community. The source files of the applications are versioned using the platform *GitHub*[4], the monolithic application is located in the repository eShopOnWeb[5] and the event-driven application is located in the eShopOnContainers[6] repository. For the development of the applications, the programming language C# 9.0 and the framework .NET 5.0 were used. Table 4 presents the characteristics of each application regarding the size and amount of source code generated during development.

4.4. Measured variables and quantification method

Independent variable. The independent variable of the formulated hypothesis is the architectural type, which takes two possible values: "*event-driven architecture*" and "*monolithic architecture*". Each value of the independent variable represents an architectural style used by the target application (see Section 4.3), which is submitted to evaluation so that we can collect data regarding the dependent variable (performance).

Dependent variable. The dependent variable on the hypothesis is performance. Performance quantifies the consumption of computational resources and data related to processing time. Performance is made up of the following metrics: CPU consumption, RAM consumption, response time, throughput or responses per minute, and packet traffic sent and received over the network. The performance metrics will be collected through an application execution monitoring tool, the chosen tool for the study was *New Relic*[7]. Table 5 presents the description of each metric grouped by attributes.

4.5. Experimental design

The experimental process of this study is divided into eight steps organized into four phases (see Figure **??**). The planning phase consists of the steps of selecting target applications, selecting metrics and tools to support the experimental process, such as monitoring tools and load testing; the preparation phase consists of configuring the data collection tools for the selected metrics and configuring the tools for load testing the selected applications; the execution phase consists of the execution steps of application load tests and data collection of selected metrics; the analysis phase consists of the steps of analyzing the collected data. The steps that make up the experimental process are described below.

Step 1: Select applications. This step aims to select the target applications to meet the two architectures defined by the independent variable, that is, to select a monolithic application and an event-driven application that represents the decomposition of the monolithic application.

Step 2: Select metrics. This step aims to select the metrics that make it possible to analyze the performance aspects of

[2]https://dotnetfoundation.org
[3]https:/ /dotnet.microsoft.com
[4]https://github.com/
[5]https://github.com/dotnet- architecture/eShopOnWeb
[6]https://github.com/dotnet-architecture/eShopOnContainers
[7]https://newrelic.com

Table 4: Target application characteristics

Architecture	Language	Framework	Files	Code lines	Commented lines	Blank lines	Total
Event-driven	C# 9.0	.NET 5.0	379	16183	485	3333	20001
Monolithic	C# 9.0	.NET 5.0	158	5585	110	1129	6824

Table 5: Performance metrics.

Attributes	Metrics	Definitions
CPU	CPU Usage	Percentage of CPU (of all types, system, user, I/O) used by the process.
Memory	RAM	Total RAM used by a process.
Time	Response Time	Average duration of execution of process transactions
	Throughput (Responses per minute)	Number of transactions per minute of the process
Network	Total Packages Transmitted	Total network packages transmitted by the process
	Total Packages Received	Total network packages received by the process

each target application. The selected metrics correspond to the dependent variable.

Step 3: Select tools. This step aimed to find tools for the execution of test scenarios and the collecting of the application's data from selected metrics. The *New Relic* monitoring tool was selected to collect data from the selected metrics during the processing of application tests, as in addition to the agents that perform the capture and collection of processing data from the target applications, the tool also provides a portal for real-time visualization and monitoring of applications during processing, in addition to storing collected data for future analysis. For the execution of the test cases in each application, it was necessary to select a tool to carry out load tests in web interface applications, for that the *Apache JMeter*[8].

Step 4: Setup environments. This step had the purpose of performing the installation and configuration of the target applications for the execution of tests. Moreover, in this step, the installation of the application's database, the message service, and the configuration of the monitoring agents for the execution of the application will be done.

Step 5: Setup load tests. This step had the purpose of elaborating the load test scenarios for each target application in the *Apache JMeter* test tool. As the event-driven application is a decomposed version of the monolithic application, the test scenarios for each application had essentially the same steps, they are: 1) access the e-commerce home page; 2) login to the site; 3) access the product catalog; 4) add three random products to the cart; 5) access the cart details; 6) finish your cart; 7) confirm payment details; 8) browse order history; 9) log out of the site. As shown in Table 6, three executions were prepared for each test scenario, each execution of the test scenario simulates the simultaneous access of 10, 25, and 50 users, in this way, the behavior of each application can be observed when reacting with increased concurrent processing.

Step 6: Execute tests. In this step, the objective was to execute the test cases of each target application. With the execution of the test cases, it will be possible to collect data regarding the application execution by the monitoring tools. The execution of test cases occurred manually and was automated through the test tool *Apache JMeter*.

Step 7: Collect data. This step had the purpose of collecting the data captured by the monitoring tool *New Relic* during the execution of the test cases. For each load test run, data was collected from selected metrics grouped by metric and test scenario.

Step 8: Result analysis. In this step, the objective was to analyze the data descriptively and identify its distribution and perform hypothesis tests using the collected data. Finally, we present the discussion of results from the perspective of dependent variables.

4.6. Analysis Procedures

Descriptive analysis. Descriptive analysis was performed to analyze the distribution, dispersions, and trends such as means and medians of data collected from each selected metric for each type of architecture defined by the independent variable.

Statistical analysis. We performed the statistical analysis for testing hypotheses. The significance level for the hypothesis tests was $\alpha = 0.05$. Statistical analyzes were performed to test the hypotheses of each metric selected and in each test scenario executed. To test the H1 hypothesis and its sub-hypotheses, the Mann-Whitney Unpaired Test was applied to each set of data collected by the selected metrics. As a result, each metric will be analyzed and compared across architecture types and for each test scenario executed. The Shapiro-Wilk test was also applied to assist in the analysis of data distribution and in choosing the statistical test.

5. Result

This section analyzes the data obtained by the experimental process described in Section **??**. The findings are derived from numerical processing of the collected data and graphical representations of aspects of the obtained results. Section 5.1 introduces the descriptive analysis of the collected data. Section 5.2 presents the data obtained through the hypothesis tests.

[8]https://jmeter.apache.org

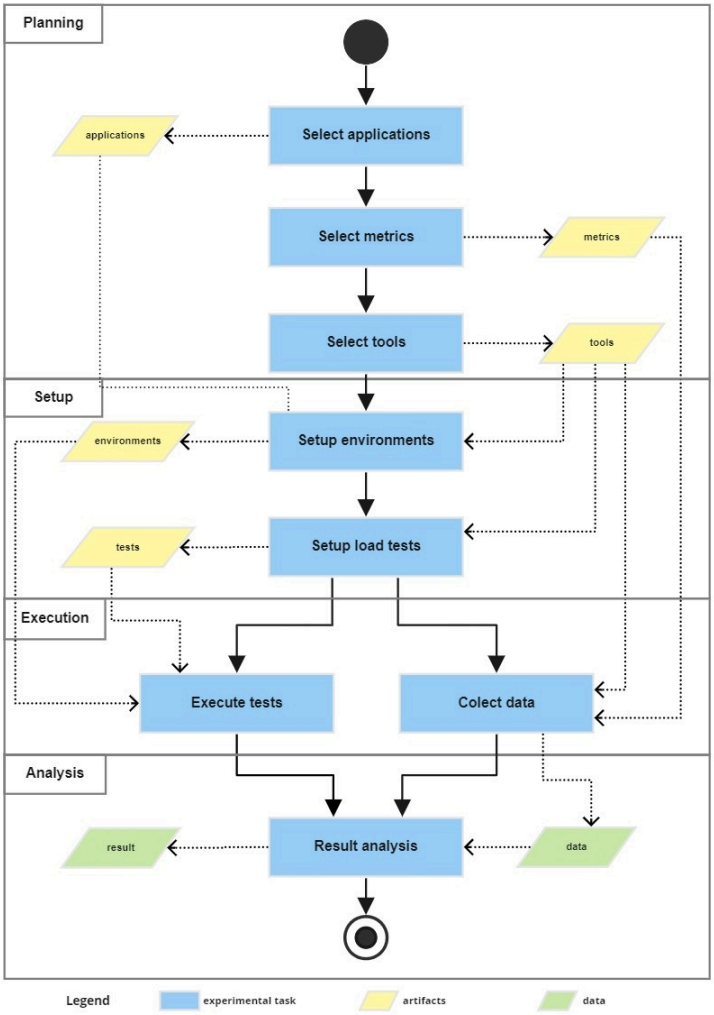


Figure 1: Experimental design

5.1. Descriptive statistics

This section describes the aspects of the collected data related to the performance of the architectural types studied in this work. For this, descriptive analysis was applied to analyze the distribution of data, trends (means, medians, etc.), and the dispersion of data sets through standard deviation. Statistical data were calculated based on 120 compositions for each selected metric (see Table 5), that is, with 60 compositions applied to event-driven architecture and 60 compositions applied to monolithic architecture. Each composition represents one minute in a continuous 60-minute timeline. Each metric analyzed has a set of 60 data compositions for each load test scenario and each architecture. When performing the analysis of the result of the descriptive statistics, it was possible to observe that the monolithic architecture had more positive effects on the event-driven architecture than the event-driven architecture on the monolithic architecture. The monolithic architecture had better values for the consumption of resources and the processing time during the empirical tests. This result is supported by the following observations for each metric (Figure **??**):

CPU Usage. The CPU usage means for each scenario of the monolithic architecture were 9.11%, 24.81%, and 38.14% and for the event-driven architecture they were 2.93%, 7.94%, and 17.54%. With these values, it is possible to see that the CPU usage by the monolithic architecture was between two and three times higher than the event-driven architecture, even considering the standard deviation of each scenario.

RAM Memory. RAM consumption means for each scenario of the monolithic architecture were 347.05MB, 349.95MB, and 344.57MB and for the event-driven architecture, they were 1723.03MB, 1721.55MB, and 1788.81MB. With these values, it is possible to verify that the event-driven architecture consumed a greater amount of RAM than the monolithic architecture, the difference in memory consumption was five times greater.

Response Time. The response time means for each scenario of the monolithic architecture were 13.31, 13.13, and 14.54 milliseconds and for the event-driven architecture, the means were 43.08, 16.68, and 16.13 milliseconds. Through these values it is possible to notice that the means are very close between the types of architectures, even considering

Table 6: Scenario of load test

Scenario	Description
1	Run load test with 10 concurrent users
2	Run load test with 25 concurrent users
3	Run load test with 50 concurrent users

Table 7: Descriptive statistics - CPU usage

Scenario	Architecture	N	Min	25	Mean	75	Max	Median	SD
1	EDA	60	1,47	2,90	2,96	4,43	13,10	4,33	2,67
	MONO	60	4,90	8,14	9,11	9,95	14,90	9,33	2,40
2	EDA	60	5,20	6,62	7,94	9,19	11,70	7,98	1,52
	MONO	60	21,20	22,88	24,81	26,60	35,00	25,13	3,09
3	EDA	60	3,90	15,89	17,54	18,52	59,30	17,78	5,93
	MONO	60	32,20	36,48	38,14	39,95	48,60	38,65	2,92

Legend: event-driven architecture (EDA), monolithic architecture (MONO), standard deviation (SD)

the standard deviation, the average values remain very close without major differences between them.

Throughput. The means responses per minute in each scenario observed for the monolithic architecture were 801.5, 1592, and 2399 requests per minute and for the event-driven architecture it had means of 2350, 5161, and 5713 requests per minute, through these values, the event-driven architecture had the highest means. This indicates that the event-driven architecture generated a higher amount of connections between client and server to handle the processing of the target application's functionalities.

Packages received per second. The means of packages received per second in each test scenario for the monolithic architecture were 30.89, 60.70, and 86.17 packages per second, while the event-driven architecture received 77.09, 152.01, and 224.54 packages per second. We observe that the event-driven architecture had the highest means and it is also noticed that the number of received packages is directly related to the number of requests per minute (throughput).

Packages transmitted per second. The means of packages transmitted per second in each test scenario for the monolithic architecture were 21.48, 42.28, and 60.98 packages per second and for the event-driven architecture they were 53.69, 104.60, and 153.66 packages per second, through these values, as well as the mean of received packages per second, it is possible to observe that the event-driven architecture obtained the highest means, and it is also noticed that the amount of transmitted packages is also directly the number of requests by minutes (Throughput).

Finally, the effects of using the monolithic and event-driven architecture are observed through two aspects: consumption of computational resources; and response time and throughput. The details of each aspect observed during descriptive statistics are presented below.

Aspect 1: Consumption of computational resources. Regarding resource consumption, the average CPU consumption was higher in the monolithic architecture than in the event-driven architecture, but this difference is smaller compared to the average memory consumption, which is lower in the monolithic architecture compared to the event-driven architecture. events. The results showed the CPU consumption was two and a half times higher in the monolithic architecture and the memory consumption five times higher in the event-driven architecture. Through this aspect, it is identified that monolithic architecture consumes fewer resources compared to event-driven architecture.

Aspect 2: Response time and throughput. Regarding performance, the response time means of the monolithic architecture were higher than the means of the event-driven architecture, the difference of the means in comparison of one architecture with the other presents very similar values, however, the dispersion of the response times of the Event-driven architecture was greater. Regarding the throughput, the average response per minute in the event-driven architecture was higher than the average in the monolithic architecture, and as well as the average response time, the event-driven architecture has greater data dispersion than the monolithic architecture. It was identified that the monolithic architecture had better response times compared to the event-driven architecture.

5.2. Hypothesis Test

Statistical tests were performed with the data collected through performance metrics to assess whether they are statistically significant. For this, it was hypothesized that the event-driven architecture has better performance than the monolithic architecture, that is, it consumes less computational resources and has better response times. To test the differences between the averages of the performance metrics, unilateral tests were performed, considering the significance level at 0.05 (p value <=0.05), as mentioned above.

The Shapiro-Wilk test was applied to evaluate the distribution of the collected data and they presented a distribution different from the normal one (p value <= 0.05) for all data sets of the executed test scenarios. So, due to data distribution, it was not possible to apply the T-Test statistical test, in this case, the non-parametric Mann-Whitney test was used as the main statistical test.

Hypothesis (H1) and its sub-hypotheses (see Table 2) were tested to evaluate RQ1 in the test scenarios (see Table 6) executed. Table 13 contains the values of the independent comparison between the event-driven architecture and the

Table 8: Descriptive statistics - RAM Memory

Scenario	Architecture	N	Min	25	Mean	75	Max	Median	SD
1	EDA	60	1656,24	1683,07	1723,03	1889,70	1941,85	1772,41	104,02
	MONO	60	335,99	340,78	347,05	391,19	684,29	378,96	67,12
2	EDA	60	1672,69	1701,61	1721,55	1732,82	1750,75	1717,59	21,01
	MONO	60	338,26	346,04	349,95	352,48	392,29	351,79	9,86
3	EDA	60	1726,06	1764,84	1788,81	1856,11	2049,38	1811,47	74,82
	MONO	60	328,51	339,12	344,57	348,19	357,42	343,85	6,15

Legend: event-driven architecture (EDA), monolithic architecture (MONO), standard deviation (SD)

Table 9: Descriptive statistics - Response Time (milliseconds)

Scenario	Architecture	N	Min	25	Mean	75	Max	Median	SD
1	EDA	60	18,51	35,83	43,08	43,08	49,33	37,67	10,86
	MONO	60	13,20	13,20	13,31	14,51	14,51	13,84	0,66
2	EDA	60	16,22	16,68	16,68	16,68	17,09	16,68	0,08
	MONO	60	13,13	13,13	13,13	13,16	13,16	13,14	0,02
3	EDA	60	16,13	16,13	16,13	22,80	22,80	18,71	3,25
	MONO	60	14,54	14,54	14,54	14,54	15,74	14,56	0,15

Legend: event-driven architecture (EDA), monolithic architecture (MONO), standard deviation (SD)

monolithic architecture in each test scenario. The p values in bold represent statistically significant results, i.e., *p-value* < 0.05. These values indicate the rejection of the respective null hypothesis.

The main finding is that monolithic architecture had higher mean CPU processing values than event-driven architecture. But, on the other hand, it was also demonstrated that the event-driven architecture had higher consumption values of more memory than the monolithic architecture, the difference is much higher than the difference in CPU consumption. It was also demonstrated that the monolithic architecture had better mean response times compared to the event-driven architecture. However, the event-driven architecture had better averages in the throughput of responses per minute, but it needed to receive and send a higher amount of data packages.

This can indicate the increase in the throughput of responses per minute, as the event-driven architecture requires a virtual bus to communicate with the modules'communication. Thus, the null hypothesis is rejected and this result is supported by the following observations:

CPU Usage. It was hypothesized that the event-driven architecture has a higher or equal CPU processing consumption than the monolithic architecture, but when analyzing the statistical test results, we observe that the *p-value* is lower than the z and below the 0.05 significance level. Therefore, the null hypothesis of no difference in CPU consumption between event-driven architecture and monolithic architecture can be rejected. That is, there is enough evidence to prove that the differences in CPU consumption between event-driven architecture and monolithic architecture are statistically different. Table 13 depicts the mean CPU usage rank of the monolithic architecture is higher than the event-driven architecture.

The *Mann-Whitney U* statistical test was performed to determine if there were differences in CPU consumption between an application with event-driven architecture and another application with monolithic architecture in the three test scenarios. Distributions of CPU consumption values for event-driven and monolithic architectures were similar, as assessed by visual inspection. The average CPU consumption was statistically significantly higher in the application of monolithic architecture for all three test scenarios (9.33, 25.13, and 38.65) than in the application of event-driven architecture (4.33, 7.98, and 17.78), U=3282, 3600 and 3540, z=7.776, 9.445 and 9.130, p=0.001, 0.001 and 0.001.

RAM Memory. It was hypothesized that the event-driven architecture has a RAM consumption greater than or equal to the monolithic architecture, however, when analyzing the results of the statistical test, it is possible to observe that the *p-value* is lower than the z and below the 0.05 significance level. Therefore, the null hypothesis of no difference in RAM consumption by event-driven architecture and monolithic architecture can be rejected. In other words, there is enough evidence to demonstrate that the differences in RAM consumption between event-driven architecture and monolithic architecture are statistically different. Table 13 depicts the average RAM consumption rank of the event-driven architecture is higher than that of the monolithic architecture.

The *Mann-Whitney U* statistical test was performed to determine if there were differences in RAM consumption between an event-driven architecture application and another application with monolithic architecture in the three test scenarios. The distributions of RAM consumption values for event-driven and monolithic architectures were similar, as assessed by visual inspection. The average RAM consumption was statistically significantly higher in the event-driven architecture application for all three test scenarios (1772.41, 1717.59, and 1811.47) than in the monolithic architecture application (378.96, 351 .79, and 343.85), U=3600, 3600 and 3600, z=9.445, 9.445 and 9.445, p=0.001, 0.001 and 0.001.

Response Time. It was hypothesized that the event-

Table 10: Descriptive statistics - Throughput

Scenario	Architecture	N	Min	25	Mean	75	Max	Median	SD
1	EDA	60	2350,00	2350,00	2350,00	2501,00	2505,00	2398,87	71,01
	MONO	60	789,00	789,00	801,50	803,00	803,00	796,18	7,02
2	EDA	60	5161,00	5161,00	5161,00	5161,00	5637,00	5169,97	61,84
	MONO	60	1592,00	1592,00	1592,00	1611,00	1611,00	1598,63	9,12
3	EDA	60	3638,00	3638,00	5713,00	5713,00	5713,00	4908,40	1012,55
	MONO	60	1859,00	2399,00	2399,00	2399,00	2399,00	2390,00	69,71

Legend: event-driven architecture (EDA), monolithic architecture (MONO), standard deviation (SD)

Table 11: Descriptive statistics - packages received per second

Scenario	Architecture	N	Min	25	Mean	75	Max	Median	SD
1	EDA	60	67,75	73,62	77,09	80,25	83,68	76,78	4,12
	MONO	60	27,22	30,01	30,89	32,58	36,67	31,19	2,13
2	EDA	60	141,50	148,06	152,01	158,08	175,38	152,82	6,46
	MONO	60	56,44	59,31	60,70	62,24	65,80	60,82	2,31
3	EDA	60	162,36	217,58	224,54	230,95	479,72	228,44	35,09
	MONO	60	80,55	84,17	86,04	88,49	93,49	86,24	2,91

Legend: event-driven architecture (EDA), monolithic architecture (MONO), standard deviation (SD)

driven architecture has a Response Time less than or equal to the monolithic architecture, but when analyzing the results of the statistical test, it is possible to observe that the *p-value* is inferior to the z and below the 0.05 significance level. Therefore, the null hypothesis of no response time difference between event-driven architecture and monolithic architecture can be rejected. That is, there is enough evidence to prove that the Response Time differences between event-driven architecture and monolithic architecture are statistically different. Table 13 depicts the average Response Time rank of event-driven architecture is higher than the one produced by monolithic architecture.

The *Mann-Whitney U* statistical test was performed to determine if there were differences in Response Time between an application with event-driven architecture and another application with monolithic architecture in the three test scenarios. The distributions of Response Time values for the event-driven and monolithic architectures were similar, as assessed by visual inspection. The mean Response Time was statistically significantly higher when applying event-driven architecture for all three test scenarios (37.67, 16.68, and 18.71) than when applying monolithic architecture (13.84, 13, 14, and 14.56), U=3600, 3600 and 3600, z=9.445, 9.445 and 9.445, p=0.001, 0.001 and 0.001.

Throughout. It was hypothesized that the event-driven architecture has a response rate per minute greater than or equal to the monolithic architecture, but when analyzing the results of the statistical test, it is possible to observe that the *p-value* is lower à z and below the 0.05 significance level. Therefore, the null hypothesis of no flow difference by event-driven architecture and monolithic architecture can be rejected. In other words, there is enough evidence to prove that the flow differences between event-driven architecture and monolithic architecture are statistically different. Table 13 depicts the average flow rate rank of the event-driven architecture is higher than that of the monolithic architecture.

The *Mann-Whitney U* statistical test was performed to determine if there were differences in the Response Rate per minute between an event-driven architecture application and another application with monolithic architecture in the three test scenarios. The distributions of Response Flow Rates per Minute for the event-driven and monolithic architectures were similar, as assessed by visual inspection. Average Response Rate Per Minute was statistically significantly higher when applying event-driven architecture for all three test scenarios (2398.87, 5169.97, and 4908.40) than when applying monolithic architecture (796.18, 1598.63, and 2390.00), U=3600, 3600 and 3600, z=9.445, 9.445 and 9.445, p=0.001, 0.001 and 0.001.

Packages received per second. It was hypothesized that the event-driven architecture has a frequency of receiving packages per second higher than or equal to the monolithic architecture, but when analyzing the results of the statistical test, it is possible to observe that the *p-value* is lower à z and below the 0.05 significance level. Therefore, the null hypothesis of no difference in packages received per second by event-driven architecture and monolithic architecture can be rejected. In other words, there is enough evidence to demonstrate that the differences in the number of packages received per second between the event-driven architecture and the monolithic architecture are statistically different. Table 13 depicts the average flow rate rank of the event-driven architecture is higher than that of the monolithic architecture.

The *Mann-Whitney U* statistical test was performed to determine if there were differences in the Receive Package per second between an event-driven architecture application and another application with a monolithic architecture in the three test scenarios. The distributions of Received packages Per Second values for the event-driven and monolithic architectures were similar, as assessed by visual inspection. The average Received packages per Second was statistically significantly higher in the event-driven architecture application for all three test scenarios (76.78, 152.82, and 228.44) than in the monolithic architecture application (31.19, 60.82 and 86.24), U=3600, 3600 and 3600, z=9.445, 9.445 and 9.445, p=0.001, 0.001 and 0.001.

Table 12: Descriptive statistics - Packages sent per second

Scenario	Architecture	N	Min	25	Mean	75	Max	Median	SD
1	EDA	60	47,46	51,47	53,69	55,69	58,19	53,45	2,70
	MONO	60	18,93	20,63	21,48	22,36	24,72	21,53	1,33
2	EDA	60	97,29	101,91	104,60	108,12	123,16	105,10	4,57
	MONO	60	39,53	41,27	42,28	43,41	45,92	42,31	1,53
3	EDA	60	104,27	149,54	153,66	157,76	310,19	155,93	21,99
	MONO	60	57,29	59,80	60,98	62,49	65,23	61,10	1,91

Legend: event-driven architecture (EDA), monolithic architecture (MONO), standard deviation (SD)

Table 13: Mann-Whitney test

Scenario	**Statistics**	**CPU percentage**	**Memory RAM**	**Response time**	**Throughput**	**Packages received**	**Packages transmitted**
1	*U'*	3.282	3.600	3.600	3.600	3.600	3.600
	U	318	0	0	0	0	0
	Z	7,776	9,445	9,445	9,445	9,445	9,445
	p-value	**0,001**	**0,001**	**0,001**	**0,001**	**0,001**	**0,001**
	Rank sum EDA	2148	5430	5430	5430	5430	5430
	Rank sum MONO	5112	1830	1830	1830	1830	1830
	Mean Rank EDA	35,8	90,5	90,5	90,5	90,5	90,5
	Mean Rank MONO	85,2	30,5	30,5	30,5	30,5	30,5
2	*U'*	3.600	3.600	3.600	3.600	3.600	3.600
	U	0	0	0	0	0	0
	Z	9,445	9,445	9,445	9,445	9,445	9,445
	p-value	**0,001**	**0,001**	**0,001**	**0,001**	**0,001**	**0,001**
	Rank sum EDA	1830	5430	5430	5430	5430	5430
	Rank sum MONO	5430	1830	1830	1830	1830	1830
	Mean Rank EDA	30,5	90,5	90,5	90,5	90,5	90,5
	Mean Rank MONO	90,5	30,5	30,5	30,5	30,5	30,5
3	*U'*	3.540	3.600	3.600	3.600	3.600	3.600
	U	60	0	0	0	0	0
	Z	9,130	9,445	9,445	9,445	9,445	9,445
	p-value	**0,001**	**0,001**	**0,001**	**0,001**	**0,001**	**0,001**
	Rank sum EDA	1890	5430	5430	5430	5430	5430
	Rank sum MONO	5370	1830	1830	1830	1830	1830
	Mean Rank EDA	31,5	90,5	90,5	90,5	90,5	90,5
	Mean Rank MONO	89,5	30,5	30,5	30,5	30,5	30,5

Legend: event-driven architecture (EDA), monolithic architecture (MONO), standard deviation (SD)

Packages transmitted per second. It was hypothesized that the event-driven architecture has a frequency of sending packages per second higher than or equal to the monolithic architecture, but when analyzing the results of the statistical test, it is possible to observe that the *p-value* is lower à *z* and below the 0.05 significance level. Therefore, the null hypothesis of no difference in packages sent per second by event-driven architecture and monolithic architecture can be rejected. In other words, there is enough evidence to prove that the differences in the number of packages sent per second between the event-driven architecture and the monolithic architecture are statistically different. Table 13 depicts the average flow rate rank of the event-driven architecture is higher than that of the monolithic architecture.

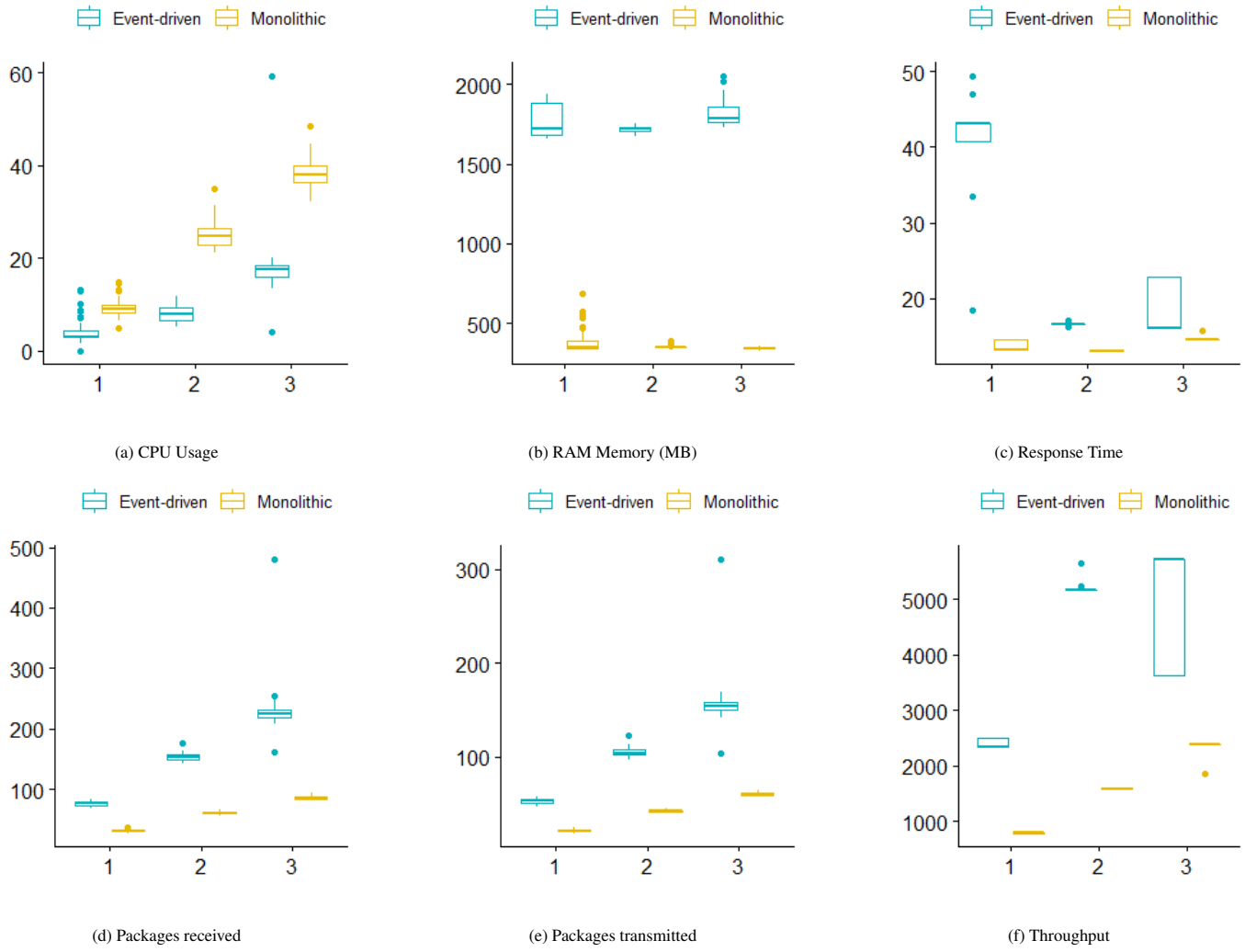
The *Mann-Whitney U* statistical test was performed to determine if there were differences in the Sending packages per second between an application with event-driven architecture and another application with monolithic architecture in the three test scenarios. The distributions of the packages Sent Per Second values for the event-driven and monolithic architectures were similar, as assessed by visual inspection. The average of packages Sent per Second was statistically significantly higher when applying event-driven architecture for all three test scenarios (53.45, 105.10, and 155.93) than when applying monolithic architecture (21.53, 42.31, and 61.10), U=3600, 3600 and 3600, z=9.445, 9.445 and 9.445, p=0.001, 0.001 and 0.001.

5.3. Discussion

Based on the observations of the results obtained by the descriptive analysis and by the hypothesis tests, the following subjects were identified for discussion:

CPU Usage. The monolithic architecture showed higher CPU consumption than the event-driven architecture, as the intensity of the load tests increased, it was possible to observe the increase in CPU consumption in both architectures, but the monolithic architecture remained with higher consumption in all test scenarios. The monolithic application has only one module responsible for processing all the application's functionalities, different from the event-driven architecture that has several modules, each one responsible for processing functionality or part of the functionality, so we speculate that due to this characteristic of the monolithic architecture, there is a processing overhead by the application causing high CPU consumption. This can be an important fact in the contribution of decision-making for reconstructions of monolithic applications, as in real applications the high consumption may be limiting the processing and reducing the application's performance. These results can contribute together with comparative studies such as the works carried out by [10] and [9], as these works carried out based on in the literature a survey of the advantages and disadvantages of using each architecture, but without empirical evidence.

RAM Memory. Event-driven architecture showed higher



(a) CPU Usage

(b) RAM Memory (MB)

(c) Response Time

(d) Packages received

(e) Packages transmitted

(f) Throughput

Figure 2: Box-plot diagrams.

RAM consumption compared to monolithic architecture and obtained the highest consumption values in all applied test scenarios. Unlike CPU consumption, RAM consumption values did not show a very significant increase as the load increased during the tests performed. The application with event-driven architecture has 9 sub-applications for processing all the functionality of the target application (see section 4.3), different from the monolithic architecture, which is composed of only one application. We speculate that the increase in RAM consumption by the event-driven architecture is due to the number of objects needed for the execution of each sub-application, as we saw in Table 4, the event-driven application has a larger amount of files or classes implemented than the monolithic application. We must also consider that each application has a memory consumption for allocating the resources needed to start and run the application itself, if we assume that to start an application 50MB of RAM is needed, then the 9 sub-applications of the event-driven application have a minimum consumption of 450MB, which is higher than the total RAM consumption of the monolithic application.

Response Time. The monolithic architecture obtained lower response times than the event-driven architecture, that is, it obtained the best values and due to this, it ends up being more performative. What differs between the architectures is the way of communication between the application modules, in the monolithic architecture, all modules are in the same application, unlike the monolithic architecture, where the modules are sub-applications that communicate through a bus. It is believed that this communication bus between the modules of the event-driven architecture can impact the application response time, but to confirm this statement, it will be necessary to apply new tests and analyze new metrics that help evidence this behavior.

Throughput. The event-driven architecture achieved better response-per-minute throughput values than the monolithic architecture. Although the application of event-driven architecture has obtained less performing response time values, it presented higher values regarding the number of responses per minute than the application of monolithic architecture. An indication of the increase in the number of responses per minute is that the metric is also counting the communication responses

between the modules of the event-driven application and not just the responses that were sent to the client.

5.4. Challenges and Implications

This section presents the challenges and implications that were derived after analyzing the collected results.

Challenge 1: Effects of modularization on performance. Modularization is a manifestation of the separation of interests, that is, it is the division or organization of the software into modules that are integrated to meet one or more functionalities [24]. In future research, it is suggested to evaluate the effects caused by modulation on the performance of event-driven and monolithic architectures, as the communication between event-driven architecture modules is done through an event bus, unlike the monolithic architecture, that modularization can be measured through some groups of metrics, they are: separation of interests, coupling, cohesion and size [25].

Challenge 2: Analyze performance during development. Unlike the tests and collected data performed for this work, in which each selected target application has all functionalities already implemented, it is recommended to apply similar tests to collect performance data in each version of the event-driven and monolithic application.

Implication 1: Refactoring or decomposition indicators. The results imply looking for factors that make it possible to indicate the moment when a monolithic application should be decomposed into an event-driven application or vice-versa. As seen earlier, the monolithic architecture consumes a higher percentage of CPU compared to the event-driven architecture, so we speculate that this might be a factor to be evaluated during the decomposition decision. For RAM, which, unlike CPU consumption, in the event-driven architecture, the consumption is higher compared to the monolithic architecture, it is also speculated that this may be another factor for the decision to refactor the event-driven architecture in a monolithic architecture.

To exemplify the use of refactoring or decomposition indicators, Figure **??** and Figure **??** demonstrate the evolutionary cycle of a given application through versions released over time. For each new version of the application, exploratory tests can be applied to assess changes in CPU and RAM consumption as a result of changes applied during development. The results will be analyzed and compared with indicators that warn about the increase in resource consumption or the need to refactor the application due to excess consumption.

6. Conclusions and Future Work

This work presented an empirical study to evaluate the performance of directed architecture and monolithic architecture. During the empirical experiment, an application with an event-driven architecture and another with a monolithic architecture were submitted to tests for data collection using previously selected metrics. The consumption of CPU and RAM, number and response times, and the number of packages transferred and received by the network of each application were analyzed during the tests to observe the effects of each architecture. Analyzing the results, it is possible to verify in which aspects each architecture has better performance.

From the results of the data obtained in this study, it was observed that applications with monolithic architecture have low consumption of resources such as CPU and RAM compared to applications with event-driven architecture, but applications with event-driven architecture have better amounts and response times. Through these results, this study contributes empirical knowledge to help decision-making in choosing the appropriate architecture for software development and also in the decomposition of monolithic applications into event-driven applications with evidence of the execution of an application using two distinct architectures.

To make this work have a more complete approach, some future work will be carried out: (1) consider new metrics to assess other performance-related aspects; (2) consider modularization metrics to analyze possible effects on performance; (3) collect data from new applications from other contexts and/or languages and frameworks used in development. This work is an initial study of the effects on performance caused by event-driven architecture, serving as support for further studies related to the topic.

References

[1] M. Rubert, K. Farias, On the effects of continuous delivery on code quality: A case study in industry, Computer Standards and Interfaces (2021) 103588.

[2] A. Oliveira, V. Bischoff, L. J. Gonçales, K. Farias, M. Segalotto, Brcode: An interpretive model-driven engineering approach for enterprise applications, Computers in Industry 96 (2018) 86–97.

[3] B. Stopford, Designing Event-Driven Systems, 1st Edition, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2018.

[4] R. Laigner, M. Kalinowski, P. Diniz, L. Barros, C. Cassino, M. Lemos, D. Arruda, S. Lifschitz, Y. Zhou, From a monolithic big data system to a microservices event-driven architecture, 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (2020) 213–220doi:10.1109/SEAA51224.2020.00045.

[5] R. G. Urdangarin, K. Farias, J. Barbosa, Mon4aware: A multi-objective and context-aware approach to decompose monolithic applications, XVII Brazilian Symposium on Information Systems (SBSI 2021), June 7–10, 2021, Uberlândia, Brazil (Jun. 2021). doi:https://doi.org/10.1145/3466933.3466949.

[6] O.-A. Schipor, R.-D. Vatavu, J. Vanderdonckt, Euphoria: A scalable, event-driven architecture for designing interactions across heterogeneous devices in smart environments, Information and Software Technology Volume 109, May 2019, Pages 43-59 (2019) 43–59doi:https://doi.org/10.1016/j.infsof.2019.01.006.
URL `www.elsevier.com/locate/infsof`

[7] S. Tragatschnig, S. Stevanetic, U. Zdun, Supporting the evolution of event-driven service-oriented architectures using change patterns, Information and Software Technology Volume 100, August 2018, Pages 133-146 (2018) 133–146doi:https://doi.org/10.1016/j.infsof.2018.04.005.
URL `www.elsevier.com/locate/infsof`

[8] E. Djogic, S. Ribic, D. Donko, Monolithic to microservices redesign of event driven integration platform, MIPRO 2018, May 21-25, 2018, Opatija Croatia (2018) 1411–1414.

[9] Z. A. Bukhsh, M. van Sinderen, P. M. Singh, Soa and eda: A comparative study: Similarities, differences and conceptual guidelines on their usage, 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE) (Jul. 2015). doi:10.5220/0005539802130220.

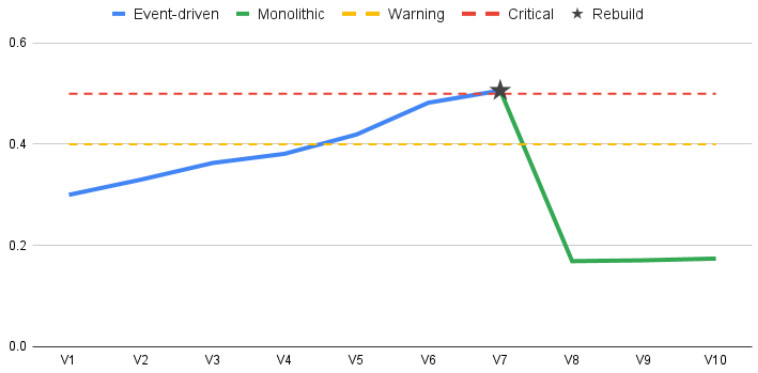


Figure 3: CPU implication

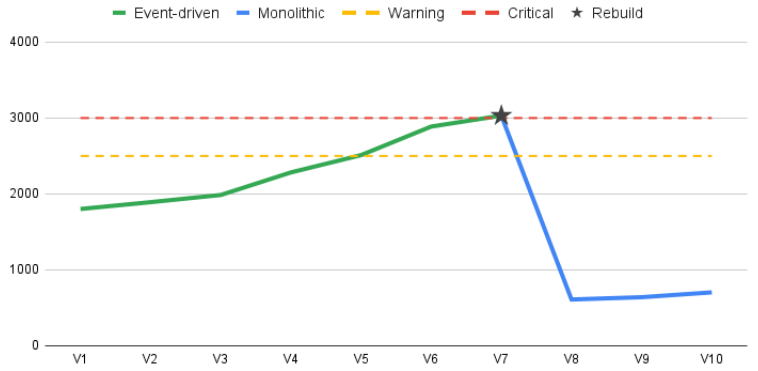


Figure 4: Memory implication

[10] C. Pienwittayasakul, Y. Liu, Comparative study on service-oriented architecture and event-driven architecture, Proceedings of the International conference on Computing Technology and Information Management, Dubai, UAE, 2014 (2014) 397–405doi:10.1.1.1020.1824.

[11] M. Richards, Software Architecture Patterns, O'Reilly Media, Inc., 2015.

[12] H. Falatiuk, M. Shirokopetleva, Z. Dudar, Investigation of architecture and technology stack for e-archive system, 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC ST) (October 2019). doi:https://doi.org/10.1109/PICST47496.2019.9061407.

[13] J. Boner, D. Farley, R. Kuhn, M. Thompson, The reactive manifesto, disponível em: <https://www.reactivemanifesto.org>. Acesso em: 18 novembro 2021 (2014).

[14] A. Kaur, P. S. Grover, A. Dixit, Performance efficiency assessment for software systems, Hoda M., Chauhan N., Quadri S., Srivastava P. (eds) Software Engineering. Advances in Intelligent Systems and Computing, vol 731 (2019) 83–92doi:https://doi.org/10.1007/978-981-10-8848-3$_8$.

[15] ISO - International Organization for Standardization, ISO/IEC 25010:2011: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models (2011).

[16] R. S. Pressman, B. R. Maxim, Engenharia de software: uma abordagem profissional, eighth Edition, AMGH Editora Ltda, Porto Alegre, 2016.

[17] R. D. Vieira, K. Farias, Usage of psychophysiological data as an improvement in the context of software engineering: A systematic mapping study, SBSI'20: XVI Brazilian Symposium on Information Systems (2020) 1–8doi:https://doi.org/10.1145/3411564.3411580.

[18] K. Farias, A. Garcia, C. Lucena, Effects of stability on model composition effort: an exploratory study, Softw Syst Model (2014) 13:1473–1494 (2013) 13:1473—-1494doi:https://doi.org/10.1007/s10270-012-0308-2.

[19] K. Farias, A. Garcia, J. Whittle, C. von Flach Garcia Chavez, C. Lucena, Evaluating the effort of composing design models: a controlled experiment, Softw Syst Model (2015) 14:1349–1365 (2014) 14:1349—-1365doi:https://doi.org/10.1007/978-3-642-33666-9$_4$3.

[20] K. Farias, A. Garcia, C. Lucena, Evaluating the effects of stability on model composition effort: an exploratory study, in: VIII Experimental Software Engineering Latin American Workshop collocated at XIV Iberoamerican Conference on Software Engineering, Rio de Janeiro, Citeseer, 2011.

[21] L. F. D'Avila, K. Farias, J. L. V. Barbosa, Effects of contextual information on maintenance effort: a controlled experiment, Journal of Systems and Software 159 (2020) 110443.

[22] K. Farias, A. Garcia, C. Lucena, Evaluating the impact of aspects on inconsistency detection effort: a controlled experiment, in: International Conference on Model Driven Engineering Languages and Systems, Springer, 2012, pp. 219–234.

[23] V. R. B.-G. Caldiera, H. D. Rombach, Goal question metric paradigm, Encyclopedia of software engineering 1 (528-532) (1994) 6.

[24] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, M. Mazzara, From monolithic to microservices: An experience report from the banking domain, IEEE Software (Volume: 35, Issue: 3, May/June 2018) (2018) 50–55doi:https://doi.org/10.1109/MS.2018.2141026.

[25] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, A. von Staa, Modularizing design patterns with aspects: A quantitative study, Rashid A., Aksit M. (eds) Transactions on Aspect-Oriented Software Development I. Lecture Notes in Computer Science, vol 3880 (2006) 36–74doi:https://doi.org/10.1007/11687061$_2$.