International Journal of Academic Information Systems Research (IJAISR)
ISSN: 2643-9026
Vol. 6 Issue 12, December - 2022, Pages: 4-13

Examining the Design Patterns of Microservices for Achieving Performance Quality Tactics

Abdelkareem M. Alashqar[1] and Zaki Kurdya[2]

Faculty of Information Technology, Islamic University of Gaza
P.O. Box 108, Gaza, Palestine
[1]aashgar@yahoo.com, [2]zkordya@gmail.com

Abstract: *Microservices is one of the newest architectural styles used in the field of software development. The approach of microservice architectural style focuses on designing the software as a collection of very small services, each of which handles small functionality, runs separately on its own process, and communicates with other microservices to provide comprehensive and coherent functionality. Many existing research works offer insights into applying design patterns and tactics when using microservice style without considering the achievement of software qualities. However, some research works reported on how design patterns contribute to software qualities. Yet, no study takes into consideration the performance quality in particular and how it is affected by patterns and tactics when adopting the microservice architectural style. This paper reviews and reports the most well-known design patterns that are practically used in achieving performance quality when developing microservices applications and provides qualitative analysis on how much these patterns can achieve the performance tactics as proposed in the literature. It also examines the degree to which the selected design patterns are discussed and handled in the Stack Overflow by developers and practitioners.*

Keywords— ***Microservices, Architectural Tactics, Design Patterns, Performance, Stack Overflow***

1. INTRODUCTION

The software architectural style describes the overall structure of the software being developed and it significantly affects the quality attributes in a positive or a negative way. Microservices is one of the most recent architectural styles used by developers when building software. As an architectural style, microservices permits the building of software systems as small services, each of which runs separately on a processing node and can communicate with other services to provide comprehensive and cohesive functionality [1]. While this approach of modular decomposition of the system into small services units enhances software qualities such as scalability, elasticity and deployability [2][3] it usually detriments other quality attributes such as performance in particular because of the communication overhead between the large number of microservices that construct the software. This overhead includes an increase in the interprocess communication, the context switches and the involved I/O operations [4]. Moreover, while microservices are beneficial to scalability attributes the trading-off with performance attributes must be considered [5]. Additionally, the dynamic and volatility of deploying microservice creates more challenges on performance [3] and optimizing performance in the microservice design is more challenging than monolithic design [6].

Although the microservices style has an overall impact on quality attributes, the detailed design approaches such as architectural tactics and design patterns play important roles in enhancing quality attributes directly when designing software. A tactic is a design decision related to a specific quality attribute that tends to be a general and abstract concept. A design pattern is a reusable and proven solution to a real-world design problem related to quality attributes or other.

While some research works studied how tactics and patterns affect quality in the microservices environment, to the best of our knowledge, there is no study focused on handling the performance quality in particular. In this paper, we provide an examination on how the design patterns can achieve the architectural tactics of performance when developing microservices systems. And how these patterns are handled by developers and practitioners in the community forums especially in the Stack Overflow. The results of our study will help practitioners deepen their understanding with how much the mostly used design patterns contribute in achieving performance when developing microservices systems.

The rest of this paper is organized as follows: Section 2 provides the related work. Section 3 introduces the scope and research methodology including the research questions. Section 4 provides the results and discussion. Threats to validity are stated in Section 5. Section 6 concludes the paper with a future outlook.

2. RELATED WORK

There are many research works found in the literature for studying the impact of architectural styles on quality attributes such as in [7]. The relationships between architectural styles and tactics are also handled in the existing research such as in [8].

Some research works in the literature examined the architectural tactics and design patterns when adopting microservices. The authors in [9] followed a systematic review of academia and industry to explore evidence of using architectural tactics and patterns in microservices. While they noted that there is no evidence of using architectural tactics for microservices in the academic and industrial literature, they generally documented 44 architectural patterns proposed in academia and 80 in industry, and they argued that the majority of these patterns are related to scalability, flexibility,

testability, performance, and elasticity quality attributes. An extension to this work is done by [10] where the authors conducted an actual analysis of 30 microservice projects in open source and found that a few of the architectural patterns are used in these projects most of them are considered SOA patterns.

The authors in [11] conducted a literature review for understanding and addressing quality attributes in microservices and reported 6 quality attributes which are scalability, availability, security, monitorability, performance, and testability then identified 19 tactics for addressing these quality attributes. However, the authors in [11] handled the architectural tactics related to modifiability quality attributes in particular. They provided a qualitative analysis of the degree to which the principles and design patterns of service-oriented architecture (SOA) and microservices systems can be mapped onto modifiability tactics.

The authors in [12] followed the qualitative approach using the interviews for studying the adoption of microservices in the software industry. They provided insights on the microservices characteristics, the applied technologies used and the effect of microservices on the ISO 25010 quality attributes. They reported that microservices style has a positive impact on these quality attributes.

With respect to collecting data from discussion forums related to using architectural tactics, the authors in [13] provided knowledge about the relationship between some general architectural tactics and some quality attributes by mining the information from the developers' discussions that were posted on Stack Overflow.

The authors in [14] conducted a performance analysis related to query response time, efficient hardware usage, hosting costs, and packet loss rate when applying the API Gateway, Chain of Responsibility and Asynchronous Messaging design patterns in the software industry.

As stated in the previous work and to the best of our knowledge, no study has focused particularly on addressing the performance patterns and how they can meet the architectural tactics for achieving the performance quality attribute when using the microservices architectural style.

3. The Scope and Methodology

This research aims at studying the design patterns of microservices for achieving the architectural performance tactics, and hence the goal of our study is framed using the following research questions:

RQ1: What are the design patterns that are mostly used in microservice architecture as extracted from the literature? To what extent can these design patterns be fitted to performance architectural tactics?

RQ2: To what extent are the selected design patterns from RQ1 used by developers and practitioners as extracted from the Stack Overflow discussion forum?

RQ3: What general statements about achieving performance quality attribute in microservices can be inferred from the answers of RQ1 and RQ2?

For answering the first part of RQ1 we followed the literature review to explore the most commonly used design patterns in microservices for achieving performance quality. For this purpose, we reviewed the most recent related sources including textbooks, published papers and white papers. As a result of our review we selected nine design patterns that are shown in Table 1 where the description of these patterns will be provided in Section 4.

Table 1: Performance design patterns for microservices architecture

#	Design Patterns
1	Timeout
2	Circuit Breaker
3	Service Mesh
4	Throttling
5	Asynchronous Communication
6	Bulkhead
7	Map-Reduce
8	Load Balancer
9	CQRS

Then to answer the second part of RQ1 we followed an analysis process based on our understanding, intuition and experience to determine to what extent each of the selected design patterns can achieve the performance architectural tactics. We used performance tactics from [15] which are depicted in Fig. 1.

For answering RQ2 we examined the Stack Overflow as a popular discussion forum used by developers and practitioners in order to determine the degree to which the selected design patterns from RQ1 are used. For this purpose, we followed the steps of data collection, processing and analysis that depicted in Fig. 2. At the beginning we collected the posts published in Stack Overflow that tagged with "microservices" term. An example of a tagged Stack Overflow post that handles the usage of a design pattern for achieving performance is shown in Fig. 3. Then we filtered out the collected posts to select only those that include the performance quality based on predetermined related performance terms. After that the data is preprocessed to remove null values, merge the comments of the posts to their questions and answers and convert all of the text to lowercase. Finally, the degree of using design patterns for achieving performance quality is examined and discussed.

We answered RQ3 based on the results of RQ1 and RQ2 and provided general statements about the achievement of performance quality in microservices systems.

International Journal of Academic Information Systems Research (IJAISR)
ISSN: 2643-9026
Vol. 6 Issue 12, December - 2022, Pages: 4-13

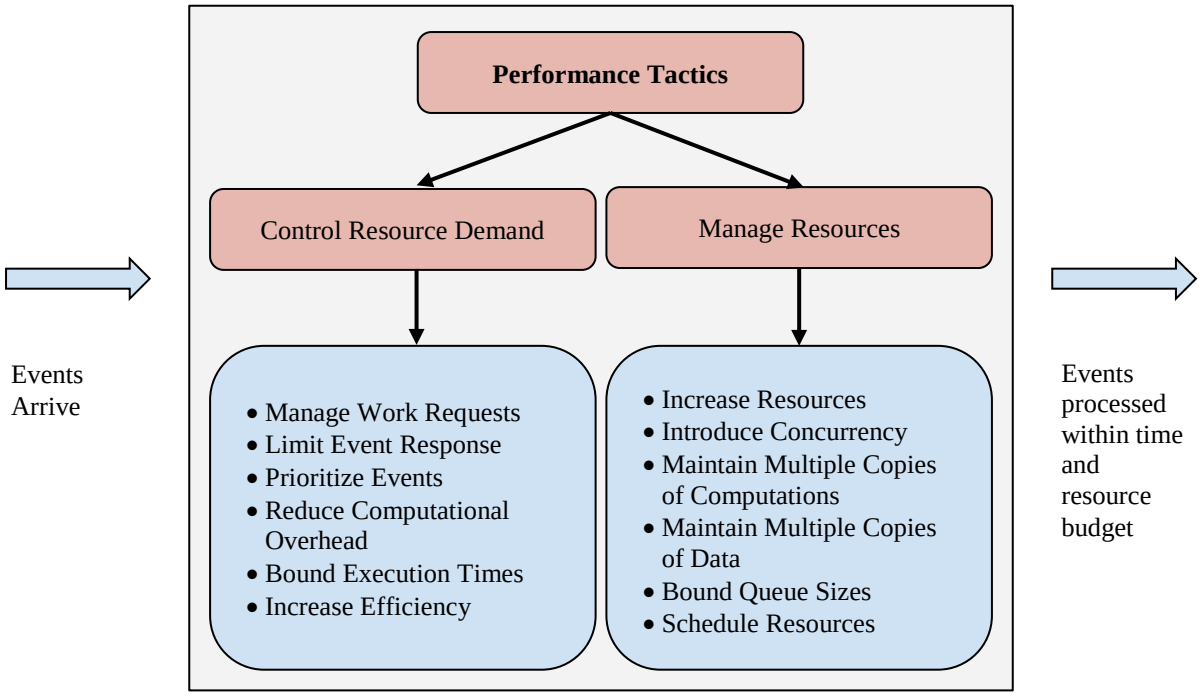

Fig. 1. *Performance architectural tactics*

Fig. 2. *The main steps for examining the design patterns in Stack Overflow*

International Journal of Academic Information Systems Research (IJAISR)
ISSN: 2643-9026
Vol. 6 Issue 12, December - 2022, Pages: 4-13

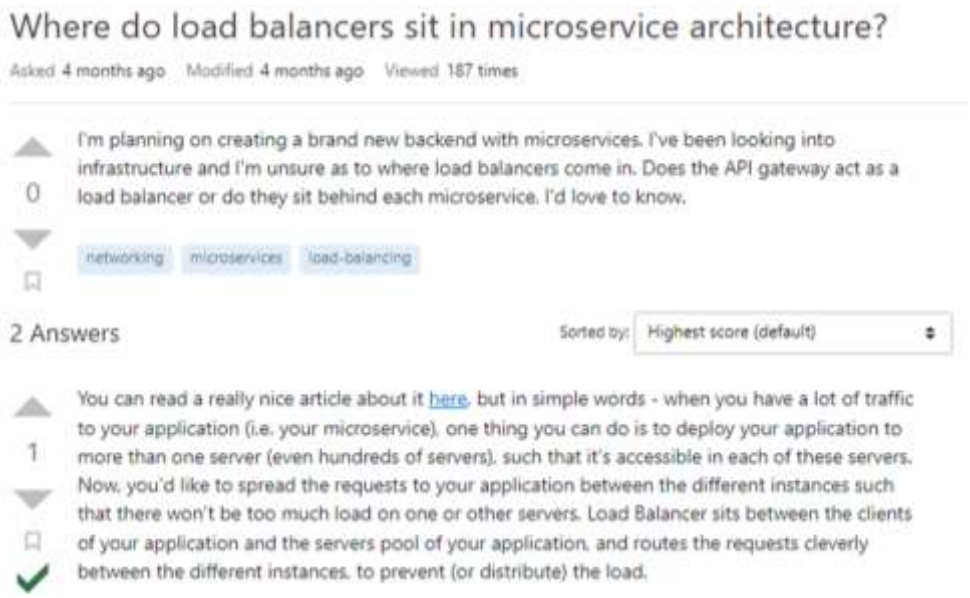

Fig. 3. *Example of Stack overflow post on microservices*

4. Results and Discussion

For answering the first part of RQ1 to determine the design patterns for microservices, and while the microservices field is very young in software development, we summarized nine well-known design patterns from the most prominent sources [2][15][16][17]. These patterns are shown in Table 1 and described in the next subsections.

- **Timeout**

In this pattern the timeout is a maximum period of time that is allowed for a calling microservice to wait for a requested microservice, where the requested microservice is considered a failed one if it does not respond within this given period of time. The timeout pattern is also used whether the microservice meets its timing constraints [15][17].

- **Circuit Breaker**

The circuit breaker acts as an electrical circuit breaker in preventing the access of failed microservice without the delays of related timeouts. Any microservice call is routed via the circuit breaker, which immediately responds with a failure status when the requested microservice does not respond several times. Thus, preventing the calling microservice from waiting and retrying for a response from the faulty microservice. The circuit breaker periodically sends a request to the failed microservice and when it detects that the microservice is responding, it resets the circuit and routes any future calls to this newly recovered microservice [15][17].

- **Service Mesh**

The service mesh includes a sidecar that accompanies each microservice and acts as a proxy for addressing application-independent concerns such as communication and monitoring. The sidecar is executed alongside each microservice to handle all interservice communication and coordination. Also, the elements of the microservice and the sidecar are often packaged and deployed together on a processing node called a pod, which improves performance by reducing the overhead of remote communication via the network [2][15].

- **Throttling**

The throttling pattern involves placing an intermediary called a throttle in front of a microservice to monitor the rate of requests coming to this microservice to determine whether these requests can be serviced. This guarantees the microservice to continue operating even when the demand reaches an extreme level [15].

- **Asynchronous Communication**

In asynchronous communication between microservices, the sender microservice sends a message and does not wait for a response from the receiver microservice. The sending messages are put in a queue to be processed by consumer microservice. In the case of asynchronous communication, the threads of the sender microservice will be released just after sending the request so that they can be utilized by other processing in the system and they can be notified when a response to the original request arrives. The request message in asynchronous communication can be processed by multiple receivers. The asynchronous mechanism permits for delivering messages between different microservices concurrently [4].

- **Bulkhead**

In the bulkhead pattern use isolated pools (also called thread pools) when connecting to different microservices. Having a dedicated pool for each individual microservice will reduce the bad impact caused by a failed microservice on the other microservices. And therefore, the system will continue to provide functionality depending on the other successful microservices [18].

- **Map-Reduce**

The map-reduce pattern is designed particularly to provide high performance when sorting and analyzing large data. In the first step, the software in map-reduce is allocated to multiple nodes of processing that run in parallel to sort the data. In the second step, two functions called map and reduce are invoked for additional processing in the sorted data. The map function takes the data and a key as input then it hashes the data into buckets based on the key. The map function is also used to filter the data to determine which data will be considered for further processing. The data mapping is achieved by multiple map instances that run in parallel where each instance accesses a different part of data. In the third and final step, the mapped buckets are shuffled in order to be processed by multiple reduce instances that run in parallel where the number of instances are similar to those in map step. The reduce step performs heavy analysis on the data buckets and produces summarized output such as averages where the amount of data output of the reduce step is always smaller than the input data [15][19].

- **Load Balancer**

The load balancer acts as a mediator that handles all requests that come from client services and determines which service from a pool of service providers will respond to these requests. The load balancer applies a scheduling algorithm to balance the load among the pool of service providers. The scheduling algorithm takes into consideration the waiting requests for a service provider and the load on each service provider as well [2][15].

- **Command query responsibility segregation (CQRS)**

The CQRS applies the separation of concern concept. It separates the command operations from the query operations that are performed on data. The command operations represent writing data which includes creating, updating, and deleting data whereas the query operations represent reading data which includes retrieving data operation. The command operations are done asynchronously in a normalized database while the query operations are done synchronously in a separate database. When the command operations are performed, the system publishes events so that the separate database used by the query operations will be updated automatically to be consistent with the main database used by the command operations. The CQRS implements query operations more efficiently, especially when retrieving data from multiple microservices. Moreover, it avoids the slowness of database operations when there is contention on a large number of reading and writing operations on the system [16].

To answer the second part of RQ2 we used the list of performance tactics shown in Fig. 1 to analyze how much these tactics can be realized by the selected design patterns we have determined and explained previously in this section. Table 2 summarizes the relationships between the architectural tactics and the design patterns when achieving performance quality in microservices. As shown in the table some design patterns are able to achieve a considerable number of tactics, such as "Load balancer" and "Map-Reduce" where 8 of the 12 tactics were realized by the "Load Balancer". Each of "Service Mesh", "Bulkhead", "Map-Reduce" and "CQRS" realizes a number of 4 tactics. The next sections provide our analysis results on how each of the selected patterns contributes in realizing the architectural performance tactics.

Table 2: The relationships between performance design patterns and performance tactics

	Timeout	Circuit Breaker	Service Mesh	Throttling	Asynchronous Com.	Bulkhead	Map-Reduce	Load Balancer	CQRS
Control Resource Demand									
Manage Work Requests	√	√	√		√	√			
Limit Event Response		√	√	√					
Prioritize Events								√	
Reduce Computational Overhead			√				√		
Bound Execution Times									
Increase Efficiency							√	√	√
Mange Resources									
Increase Resources								√	
Introduce Concurrency					√	√	√	√	√
Maintain Multiple Copies of Computations			√		√	√	√	√	√
Maintain Multiple Copies of Data								√	√
Bound Queue Sizes						√		√	
Schedule Resources								√	

It is noticed that the "Load Balancer" performs all of the "Manage Resources" tactics category. Since the "Load Balancer" acts as a mediator between client requests and a pool of servers that respond to these requests, the servers can indeed work concurrently and hence the "Introduce Concurrency" tactic is achieved. Also, the tactics of both "Maintain Multiple Copies of Computations" and "Maintain Multiple Copies of Data" can be realized through the existing servers. Moreover, additional servers can be added easily so the "Increase Resources" tactic is applied, any requests can be controlled so the "Bound Queue Sizes" tactic is realized and these requests can be scheduled so the "Schedule Resources" tactic is realized as well. Additionally, the "Load Balancer" permits for "Prioritize Events" tactic because any event can be checked for priority before forwarding it to the dedicated server so that the low-priority requests can be discarded to free resources for the high-priority requests. The efficiency of a server resources can be enhanced and hence the "Increase Efficiency" tactic is achieved by the "Load Balancer".

In "Map-Reduce" design pattern, sorting the huge amount of data is done by multiple processing nodes that run in parallel and hence the tactic "Introduce Concurrency" is realized. In the analysis step, the function of data mapping is processed by multiple instances that run concurrently on different portions of data to produce multiple buckets of transformed data. Then these buckets of data are processed for data reduction by different concurrent processing instances where the number of these instances is similar to the number of data buckets. And hence in addition to achieving "Introduce Concurrency" tactic, the "Maintain Multiple Copies of Computations" is totally realized by the "Map-Reduce" pattern. Since each portion of data is processed locally in a specific processing node with independence from other nodes, then the "Reduce Computational Overhead" tactic is realized. Moreover, the "Increase Efficiency" tactic can be achieved by enhancing the algorithms of "map" and "reduce" functions.

The "CQRS" design pattern permits for the separation of data processing responsibilities into read and write operations. This separation includes conceptual means as well as physical means. As a result, it achieves the "Introduce Concurrency" tactic where the two types of read and write operation can be done concurrently with the possibility of controlling the coming data processing requests. Also, the "CQRS" applies the requests of read operations to be done on a separate database and hence it realizes the tactic of "Maintain Multiple Copies of Data". Since each separate database can be processed on a separate processing node, and the "CQRS" also permits for creating multiple instances for the same type of operations, the "Maintain Multiple Copies of Computations" is carefully performed. Additionally, the "CQRS" permits for enhancing servicing the data operations through programming algorithms and hence it realizes the "Increase Efficiency" tactic.

In the "Bulkhead" each coming request is checked before forwarded it to the server so the "Manage Work Requests" tactic is realized. The result of checking the request is placing it into a pool that acts as a queue for a dedicated microservice which means that microservice has its own controlled pool of requests so the "Bound Queue Sizes" tactic is realized. While each microservice runs independently from other microservices then the "Introduce Concurrency" tactic is achieved. Additionally, different instances for the same microservice can run on multiple servers so the "Maintain Multiple Copies of Computations" tactic is considerably realized.

The existence of the collocated sidecar alongside the microservice in the "Service Mesh" pattern can handle all the remote communication with other microservices and hence it applies the "Reduce Computational Overhead" tactic. The microservice and its collocated sidecar are processed separately and hence the "Maintain Multiple Copies of Computations" is realized. Additionally, the "Service Mesh" pattern has the capability of controlling the delivery of microservice requests so it applies the "Manage Work Requests" tactic and controls the coming requests as well so it realizes the "Limit Event Response" tactic.

The "Asynchronous Communication" pattern permits the requester microservice to send a request without waiting for the server to respond. The server notifies the requested microservice by calling back upon completion so the two microservices can work separately and concurrently and hence the "Introduce Concurrency" and "Maintain Multiple Copies of Computations" tactics are realized. Queueing the sending requests before servicing them permits for the "Manage Work Requests" tactic.

The "Circuit Breaker" design pattern carefully applies the "Manage Work Requests" since it can determine whether the calling microservice requests a failure microservice or not. It can determine the failed microservice and responds immediately with a failure status for the requests of this failed microservice. Not forwarding the requests for the failed microservice means applying the "Limit Event Response" tactic.

Controlling the waiting time for calling a microservice in the "Timeout" pattern realizes the "Manage Work Requests" tactic. And controlling the coming request for a microservice in the "Throttling" design pattern means that the "Limit Event Response" tactic is achieved.

For answering RQ2 to determine the degree to which the selected design patterns are used by developers and practitioners when addressing the performance, we followed the steps of data processing and analysis that shown in Fig. 2 and previously explained in Section 3.

In the "Collecting Data" step, we used the community posts published by developers and practitioners in the Stack Overflow that handled the development of microservices systems. We collected all posts published in the Stack Overflow until October 4, 2022 that were mainly tagged by the "microservices" keyword [20]. The number of collected posts reached 8433 where each post has 11 elements of data which are the "Number of votes", "Number of answers", "Number of views", "Question title", Question short description", "Question tags", "Owned user", "User details", "Date", "Question text" and "Answers". The text of "Question text" and "Answers" elements also include the comments that may

be augmented to them. Although these elements are important for guiding us to the right collected data, we are interested only in the questions, their answers and comments for the purpose of this study. Then we applied two filtering steps on the original data. First, we filtered out the original collected data by selecting only the posts that handled the performance quality attribute. We selected any post that includes at least one of the following terms related to performance quality attribute:

"performance", "perform", "performing" , "latency", "deadline", "delay", "efficiency", "efficient", "throughput", "response time", "waiting time", "processing time", "execution time", "loading time", "blocked time", "resource utilization", "resource consumption", "resource usage", "resource contention", "memory occupancy", "capacity" , "load".

As a result, the number of posts that include the performance terms becomes 1467. Second, from the 1467 posts we selected only the posts that handled only at least one of the 9 examined design patterns in this study. For applying the automatic selection of posts that handled the design patterns we adopted the related and equivalent terms in addition to different writing formats for each design pattern as shown in Table 3. After this filtering process the number of selected posts becomes 967.

Table 3: Related terms and equivalent written formats of the design patterns

#	Design Patterns	Equivalent Terms
1	Timeout	timeout, time-out, time out
2	Circuit Breaker	circuit breaker, circuit-breaker, circuitbreaker
3	Service Mesh	service mesh, service-mesh
4	Throttling	throttling
5	Asynchronous Communication	asynchronous communication, asynchronous integration, asynchronous-communication, asynchronous-integration
6	Bulkheads	bulkhead, bulk head, thread pool, thread-pool, threadpool
7	Map-Reduce	mapreduce, map-reduce, map reduce|map and reduce
8	Load Balancer	load balancer, load-balancer, loadbalancer
9	CQRS	cqrs, command query responsibility segregation

Then we automatically [20] find the occurrence for each design pattern that is handled when discussing the achievement of performance quality in microservices. It is important to denote that when manually reviewing the majority of these posts, although the participants are interested in achieving the performance quality by considering the design patterns in microservices, they sometimes ask for more information about the patterns, how these the design patterns are applied, how to fix the errors they encountered when implementing the patterns.

Fig. 4 depicts the number of posts in each of which a design pattern is discussed when handling the performance quality terms. The design patterns are ordered from left to right in descending order based on the number of posts that handle a design pattern.

It is clear that the “Load Balancer” is the most frequently discussed design pattern and the “Map-Reduce” is the least frequently discussed one. It is obvious that the “Load Balancer” is heavily discussed by developers and practitioners for achieving performance quality in microservices since 479 of the total performance posts handled this design pattern. The “Timeout” and “CQRS” patterns are discussed with high consideration but the “Map-Reduce” and “Throttling” are discussed very little. The “Circuit Breaker” and “Bulkhead” are moderately considered by participants when discussing the performance design patterns in microservices.

It is important to denote that the design pattern can be discussed individually in the published posts or discussed alongside other design patterns. For instance, as depicted in the UpSet plot in Fig. 5, from the total of 479 posts that handled the “Load Balancer” 361 of them discussed this pattern individually while 41 of them also discussed the “Timeout” pattern. In other words, 41 of the published posts that discussed the “Load Balancer” also discussed the “Timeout” as well. Moreover, there are published posts that handle 3 design patterns or more at the same time. For instance, 6 published posts discussed the “Timeout”, “Circuit Breaker” and “Bulkhead” patterns at the same time while 1 post discussed the “CQRS”, “Circuit Breaker”, “Bulkhead” and “Asynchronous” patterns simultaneously.

International Journal of Academic Information Systems Research (IJAISR)
ISSN: 2643-9026
Vol. 6 Issue 12, December - 2022, Pages: 4-13

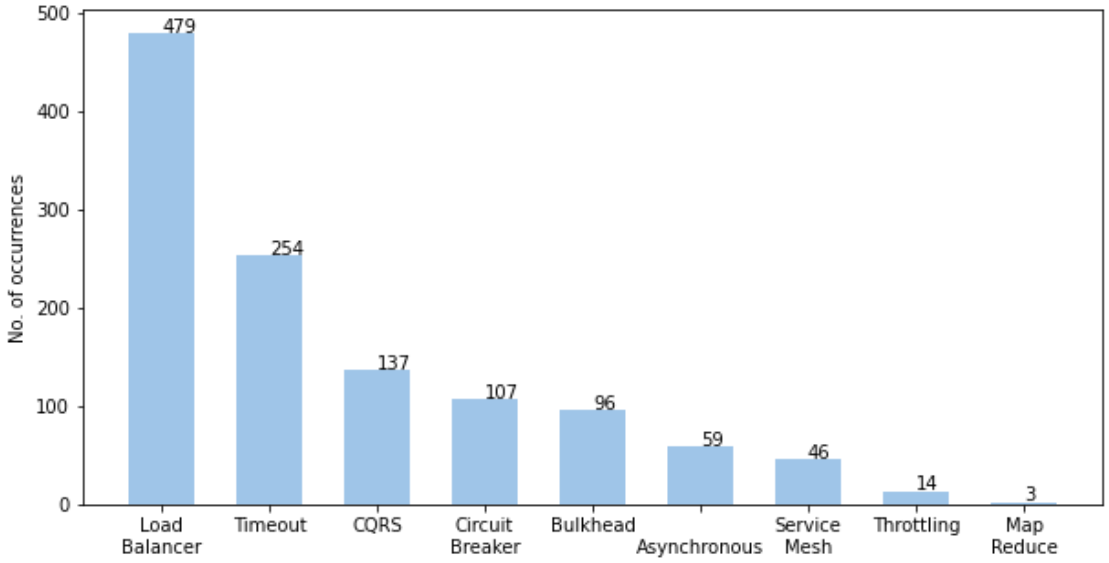

Fig. 4. *The occurrence of contribution of design patterns in achieving performance quality*

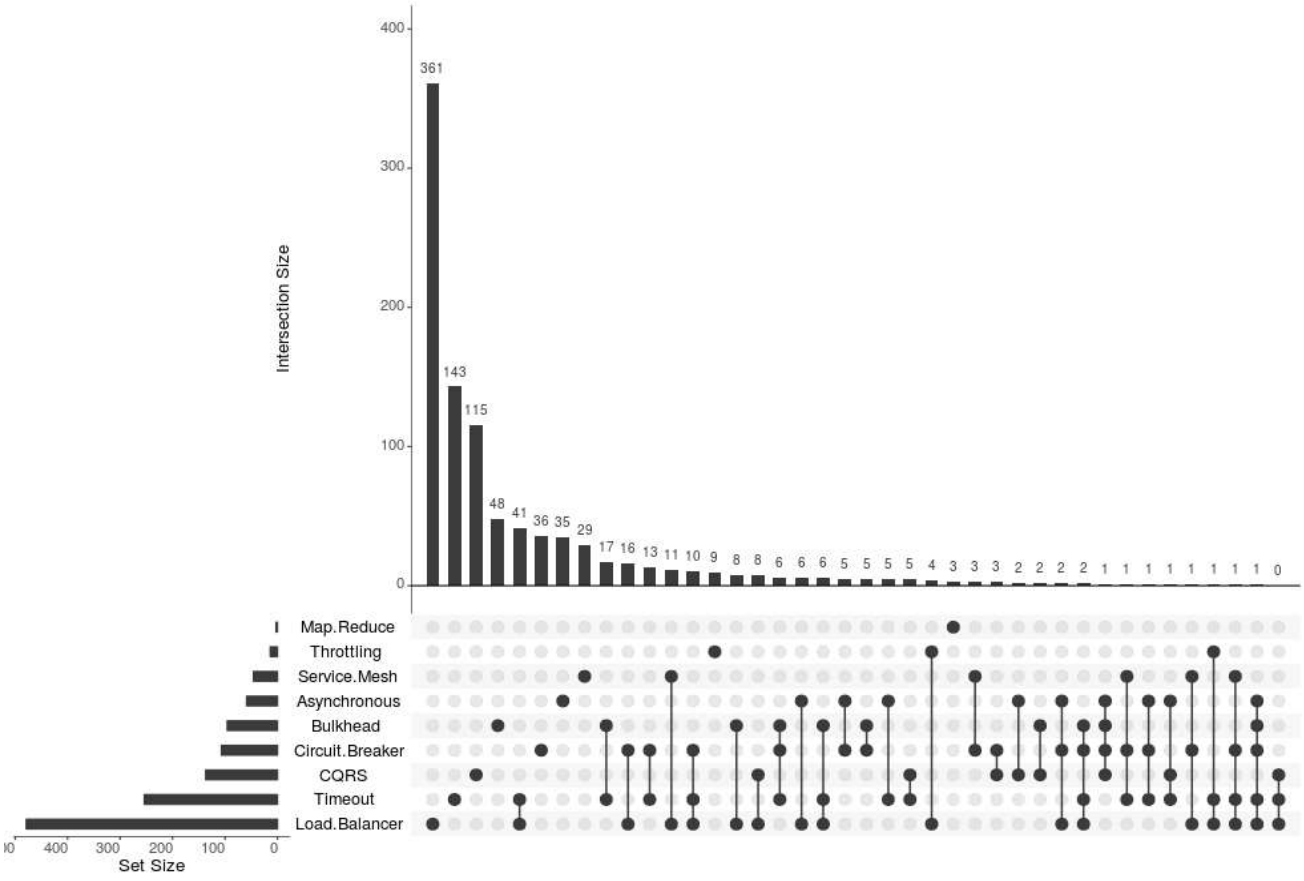

Fig. 5. *UpSet plot showing the intersections between design patterns via Stack Overflow posts*

For answering RQ3, when interpreting the results obtained from answering RQ1 and RQ2, we found that some design patterns are handled heavily by developers and practitioners when discussing the development issues related to achieving performance quality in microservices systems such as “Load

balancer", "Timeout" and "CQRS", while some design patterns are discussed very little such as "Map-Reduce" and "Throttling". Although the most frequently discussed design pattern such as "Load Balancer" has the capability of achieving a considerable number of tactics, the "Map-Reduce" design pattern also contributes in achieving a considerable number of tactics but it is not highly discussed by the Stack Overflow participants. It is clear that the "Map-Reduce" is more suitable to be used in systems that process large amounts of data which is rarely used in the microservices architecture since each microservice manages its own data storage.

Moreover, the highly discussed design pattern (e.g. "Timeout") does not mean that this pattern realizes a larger number of performance tactics and vice versa. It appears that a considerable number of highly discussed design patterns are known to developers because they are familiar with these patterns because they mostly used them in preceding types of architectural styles such as SOA. Also, we argue that the "Timeout" design pattern can be easily implemented and integrated with existing platforms with comparison to other design patterns.

So, the occurrence of discussing a design pattern is not always a key indicator for its capability in achieving the performance tactics. Additional issues that can contribute to the usage of a design pattern include its popularity between practitioners, ease of implementation, availability of tools that implement the pattern.

When looking at the realization of tactics by the selected design patterns, it is clear that the realization is more focused on the "Manage Resources" category of performance tactics. And this category that includes 6 tactics is completely achieved by the "Load Balancer" pattern, while 4 of these 6 tactics are fitted by the "CQRS", "Bulkhead" and "Map-Reduce" patterns. Also, we found that the "Bound Execution Times" tactic is not explicitly applied by any of the selected design patterns.

5. Threats to Validity

There are several potential limitations and threats to validity that can affect the results of this study.

The microservices is considered a younger architectural style with comparison to other styles where the concepts, theories and practices related to it are still not completely mature and this may affect the validity of results in making some results related temporality to the current state of microservices especially for the design patterns. To mitigate this threat, we searched for considerable and recent resources to provide a comprehensive view for the selected design patterns.

The qualitative analysis on how much each design pattern can achieve the performance quality attributes are based on the authors judgments which possibly provides subjectively biased results that may affect its reproducibility.

Collecting the data from only Stack Overflow affects the generalization of the results and findings of this research. Nonetheless this threat is mitigated since Stack Overflow is considered the largest and the most popular community forum used by developers and practitioners especially for the new approaches of software development. Although other sources of information like Stack Exchange and GitHub can strengthen the results of this research, adopting these sources in collecting data can be an extended future work.

6. Conclusion and Future Work

In this paper we provided an examination on how much the design patterns of microservices can achieve performance quality tactics. For this purpose, we first searched the literature for the most known performance design patterns used in microservices systems and provided an analysis on how each of the selected design patterns can achieve the performance tactics. Second, we studied the degree to which the selected design patterns can be discussed by developers and practitioners in the Stack Overflow.

Based on our findings, some design patterns realize a large number of performance tactics such as "Load Balancer" and are also handled heavily by developers and practitioners in the Stack Overflow. However, some design patterns achieve a limited number of performance tactics but they are also highly considered by developers and participants in the Stack Overflow such as "Timeout". Other design patterns such as "Bulkhead" achieve performance tactics and are handled by practitioners moderately. So, the capability of a design pattern in achieving performance tactics is not always the main reason for its popularity when discussed by practitioners.

While software developers discuss some design patterns largely via the community forums such as the Stack Overflow, they should pay more attention to other design patterns, especially those that contribute in the realization of performance tactics when developing microservices.

An extension to this work is to extract information from community forums other than the Stack Overflow. The extracted information will also include, in addition to design patterns, the performance tactics. Also, use machine learning techniques for extracting information and mining knowledge for achieving the performance quality through the design patterns.

7. References

[1] Fowler, M. (2014). Microservices: a definition of this new architectural term. MartinFowler. com, 25(14), 14-26.

[2] Richardson, C. (2017). Microservices Patterns. Manning Publications.

[3] Heinrich, R., Hoorn, A. V., Knoche, H., Li, F., Lwakatare, L. E., Pahl, C., Schulte, S., & Wettinger, J. (2017). Performance engineering for microservices: research challenges and directions. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, 223-226.

[4] Christudas, B. (2019). Practical Microservices Architectural Patterns: Event-Based Java

Microservices with Spring Boot and Spring Cloud. Apress.

[5] Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. Information and software technology, 131(106449).

[6] Zeng, R., Hou, X., Zhang, L., Li, C., Zheng, W., & Guo, M. (2022). Performance optimization for cloud computing systems in the microservice era: state-of-the-art and research opportunities. Frontiers of Computer Science, 16(6), 1-13.

[7] Alashqar, A., El-Bakry, H., & Abo Elfetouh, A. (2017). A framework for selecting architectural tactics using fuzzy measures. nternational Journal of Software Engineering and Knowledge Engineering, 27(03), 475-498.

[8] Harrison, N. B., & Avgeriou, P. (2010). How do architecture patterns and tactics interact? A model and annotation. Journal of Systems and Software, 83(10), 1735-1758.

[9] Osses, F., Márquez, G., & Astudillo, H. (2018). Exploration of academic and industrial evidence about architectural tactics and patterns in microservices. Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings.

[10] Márquez, G., & Astudillo, H. (2018). Actual Use of Architectural Patterns in Microservices-based Open Source Projects. In 2018 25th Asia-Pacific Software Engineering Conference (APSEC), 31-40.

[11] Bogner, J., Wagner, S., & Zimmermann, A. (2019). Using architectural modifiability tactics to examine evolution qualities of Service- and Microservice-Based Systems. SICS Software-Intensive Cyber-Physical Systems, 34(2), 141-149.

[12] Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2019). Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality. In 2019 IEEE international conference on software architecture companion (ICSA-C), 187-195.

[13] Bi, T., Liang, P., Tang, A., & Xia, X. (2021). Mining architecture tactics and quality attributes knowledge in Stack Overflow. Journal of Systems and Software, 180(111005).

[14] Akbulut, A., & Perros, H. G. (2019). Performance Analysis of Microservices Design Patterns. IEEE Internet Computing, 23(6), 19-27.

[15] Bass, L., Clements, P., & Kazman, R. (2022). Software architecture in practice. Addison-Wesley.

[16] Pacheco, V. F. (2018). Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices. Packt Publishing Ltd.

[17] Sommerville, I. (2020). Engineering software products. Pearson.

[18] "Bulkhead pattern - Azure Architecture Center | Microsoft Learn." https://learn.microsoft.com/en-us/azure/architecture/patterns/bulkhead. Accessed November 29, 2022.

[19] Miner, D., & Shook, A. (2012). MapReduce design patterns: building effective algorithms and analytics for Hadoop and other systems. O'Reilly Media, Inc.

[20] Examining the Design Patterns of Microservices for Achieving Performance Quality Tactics. (2022). Retrieved November 17, 2022, from https://colab.research.google.com/drive/1dOmmWOYxvHNshxr7baGDmkKosWl4lpUj#scrollTo=OizCjPUQ9U-o.