www.ijaceeonline.com ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 11, November 2022, pp. 1 – 6

An Approach to Performance Optimization of WSO2 Platform to Improve its Throughput

Stanley Makori Ondero
Department of Computer Science,
Jomo Kenyatta University of Agriculture and Technology, Kenya
Email: stanmakori@gmail.com

Wilson Cheruiyot
Department of Computer Science,
Jomo Kenyatta University of Agriculture and Technology, Kenya
Email: wilchery68@jkuat.ac.ke

Michael Kimwele
Computer School of computing and information technology,
Jomo Kenyatta University of Agriculture and Technology, Kenya
Email: mkimwele@jkuat.ac.ke

Abstract: *Application programming interface (API) performance is central to the overall success of companies that offer their services through Open APIs. [3] surveyed on maturity levels of Open API using open-source API gateway (WSO2) API platform, identifies performance as one of the issues facing adoption of open-source gateways. Although various approaches to optimization have been proposed [7], there is no academic documentation on the effectiveness of Redis server in an open-source API gateway, particularly WSO2. Furthermore, an optimization technique depends on the application and other configuration parameters. This research presents an empirical evaluation to compare two performance optimization techniques namely application level and Redis caching using open source WSO2 API gateway. By inspecting the obtained results, it was observed that for API consumers below 100, the performance is almost the same. This means that for internal consumption or low situations that have low traffic demand, it is economical to use internal application-level caching.*

Keyword: *API, Throughput, XML, JSON, JMeter, Redis*

1. INTRODUCTION

There is an accelerated growth in digital services for various industries. These are powered by a suite of web services and APIs. APIs and web services have been around since early 90s [11]. To fully realize the potential value of APIs, organizations should understand their usage, potential business models and monetization strategies should be present [6]. Leading organizations in the API economy such as WSO2, Oracle with their Fusion Middleware API Manager, Google with Apigee, Amazon Web Service with Amazon API gateway, Red hat with their API Connect and many others, have created API management platforms to expose their assets as well as sell to their partners [11]

Cite this paper:
Stanley Makori Ondero, Wilson Cheruiyot, Michael Kimwele, "An Approach to Performance Optimization of WSO2 Platform to Improve its Throughput", International Journal of Advances in Computer and Electronics Engineering, Vol. 7, No. 11, pp. 1-6, November 2022.

API management platform has multiple components. This includes an API Gateway where runtime policies such us throttling and payload size are created and enforced, a Security component for access and API key management, a Developer Portal which provides a centralized location for developers to discover, subscribe to and test APIs and generate their access keys (consumer key and secret key), an API Publisher designing and publishing APIs, an API Analytics intelligence on API usage, security threats, monitoring and streaming of traffic etc.

1.1 Top API Management gateways

According to online research firm, Gartner, the 2021 review identified the following as the top API management platforms:

i)AmazonAPI Gateway –used purely for Amazon's webservices consumption, the gateway is described to be easy to use and easy to manage.

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 11, November 2022, pp. 1 – 6

ii)Google Apigee – A commercial API management system. Can be provided both on premise and on-cloud. Provides both web and backend services. Manufactured by google.

iii)MuleSoft – Commercial API gateway which provided both API publisher and enterprise service bus. Manufactured by MuleSoft LLC.

iv)Microsoft Azure API management system – A commercial API management system that is said to be fast in push and pull requests, from Microsoft corporation.

v)IBM API Connect – Commercial API management system from IBM. Suitable for direct P2P connections and lacks ability for complex API manipulations.

vi)WSO2 – Open-source API Micro-gateway that is rich in API mediation and transformation. Provides ESB for messaging, Kafka powered stream processor for real-time streaming.

vii)TIBCO Mashery – Cloud based that was initially targeted at exposing its sister API exchange gateway services.

1.2 Challenges of API management system.

Literature has pointed out the following as the key impediments to the maximization r adoption of API management gateways.

i. Social - lack of awareness of the significance of the platforms to the enterprise, the cost of acquiring and implementing and inability to sustain them once in operation [3][9].

ii. Technical and complexity in API transformation [7].

iii. Performance of API gateways as compared to normal webservices [2][10]

This research focused on performance challenges and proposed one of the ways to mitigate the production issues faced by enterprises that have embraced the concept of exposing their digital services through API management gateways.

1.3 Problem Statement

Performance of API Management is a top requirement for the successful running of open APIs, yet it is a top challenge experienced by many Open API practitioners [9]. The messages delay in delivery, or they are dropped completely, and the request initiators must always initiate call-backs to check the status of their requests.

WSO2 platform performance declines significantly in content-based routing scenarios. The most effective and basic form of optimization for APIs is caching [15]. Storing tokens and access keys reduces database or backend roundtrip. Currently, the available caching mechanism for WSO2 API gateway is standalone server-side caching. However, the token isn't cached across multiple clusters in a distributed setup of the gateway, which means authentication must be done multiple times since the instances are load balanced. According to [13], the effectiveness of application-level caching largely depends on the application, the presence of invalid recommendations and additional configurations such as time-to-live for the caching protocol in use. Secondly, using a distributed cache via Redis or Memcached or any graph database allows the Oath2 token to be cached once and can be shared among the cluster servers and instances. Thirdly, the available optimization mechanisms have been evaluated individually; however, their effectiveness on API gateways specifically WSO2 have not been compared. This research presents an empirical evaluation to compare two performance optimization techniques namely application level and Redis caching using open source WSO2 API gateway.

1.3 Research objectives

The main objective of this research implemented an approach that will optimize WSO2 API management platform throughput and API response times.

2. LITERATURE REVIW

The literature on the performance of open-source API management platforms like Gravitee, APIMan and WSO2 has been clearly outlined. Most research is based on raw Java as API client, giving room for new research based on frameworks like SpringBoot and okhttp. The literature available compares the performance of similar platforms like Apache ServiceMix ESB, WSO2 ESB and MuleSoft ESB with related platforms, focusing on their response times. This gives room for research on individual platforms under certain conditions and targeting additional parameters like throughput and resource utilization.

[17] have compared the performance of Redis and relational databases. The approach used was by querying the two databases and document their performance. In their findings, the research found Redis to 10 times faster than the relational database under consideration. This research didn't mention the specific database it targeted, because various relational databases behave differently under different conditions. It is therefore not correct to conclude that Redis does better than all databases in all scenarios without carrying out empirical evidence. The other gaps in the research include comparing Redis with other NoSQL databases like MongoDB, resource consumption of Redis when returning responses and comparing it with internal caching for web servers and API gateway

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 11, November 2022, pp. 1 – 6

platforms. The research agrees with another research by [20] which used the same procedure and found Redis queries to be 3.3 times faster than MySQL, taking 52 milliseconds to respond compared to MySQL which takes 61 milliseconds. There is room to use the same approach but research on web-based platforms like API gateway and web servers.

[16] has created a parallel, scalable, effective, responsive and fault-tolerant framework to perform end-to-end data analytics tasks in real-time and batch-processing manner, using Twitter posts as a case study. The approach was caching posts in Redis as message broker and MongoDB for backend storage. But according to [20], Redis at its core is a caching tool and not a specialist message broker. Therefore, the findings could be different when using typical message brokers [5]. There is also room to stream Twitter posts using a specialist API gateway with a distributed message broker and a graph database as the backend.

[19] proposed a model for optimizing HBase database when working with Redis server. The model improves the ability of rapid acquisition and analysis of image data and achieve a higher retrieval efficiency about image data. There is room to use the same approach but optimize API gateway or web servers.

As seen above, there was need to research and develop an approach that improves the performance of Open source WSO2 API management platform for better throughput and response times. This area had not been covered in any of the academic work.

3. METHODOLOGY

A. Conceptual design

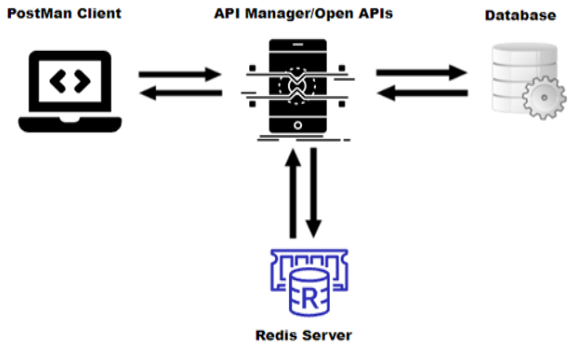

Figure 1: APIM-Conceptual model

B. Introduction.

The purpose of the paper was to develop an approach that will optimize WSO2 API management platform throughput and API response times. In this section, the research details the methodology used to accomplish the objective.

First, a basic payload was created to create a customer order and return the order details from a listening API.

```
{
 "customerName": "string",
 "address": "string",
 "MobileNo": "integer",
 "OrderQuantity": 0,
 "DeliverTo": "string"
}
```

C. Experiment setup

Redis 5.5 server was integrated with WSO2 API gateway. Jedis 3.6.0 library was used to allow WSO2 gateway to run Redis commands from the gateway and cash static and repeat requests and tokens. The internal cache of WSO2 is limited to caching tokens and in a distributed setup, each server must re-authenticate the tokens a fresh. Therefore, a fast, centralized in memory and caching platform is preferred. A basic Rest API was created and published in WSO2 portal. The test user subscribed to the published API and accessed it from Apache JMeter 5.5, which acts as API client as well as collecting performance metrics.

The datasets used for analysis were obtained by collecting performance metrics e.g., response time and throughput when a given set of concurrent users invoke the API. The concurrent users were simulated by JMeter threads and samples transactions were configured using JMeter's loop counter.

The primary pre-processing of the WSO2 API optimization approach was to allow API consumers to post their queries and receive the responses in a shorter time. This was achieved through invoking the API form Apache JMeter and then collecting the response payload and the time it takes to get response, among other metrics. The following is the layout and the components that were used in the experiment:

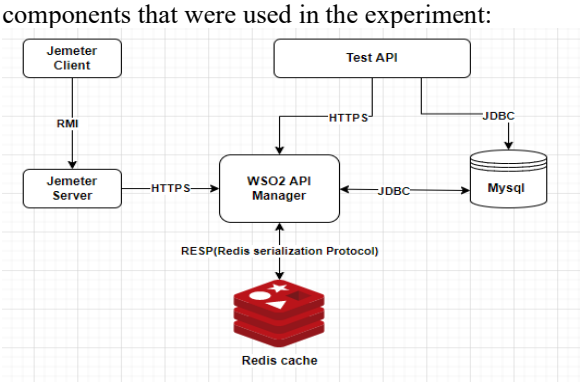

Figure 2: Experimental setup

The research follows quantitative research. This is because it collects data from available APIs. It is done int two phases: one phase will use a generic WSO2

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 11, November 2022, pp. 1 – 6

API gateway and fire API call from Apache JMeter and performance parameters including throughput, transactions per second and memory footprints. The second phase will involve introducing Redis as a caching tool and the said performance metrics be recorded. Redis was preferred over Memcached because of its simplicity of use and research shows that it is faster than most of the available caching tools.

4. RESULTS AND DISCUSSIONS

A. Performance without Redis cashing server

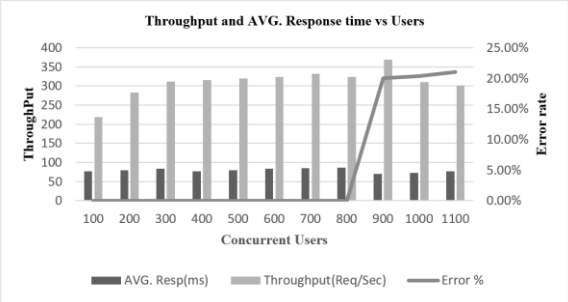

Figure 3: Performance Without Redis

B. Performance after adding Redis caching server

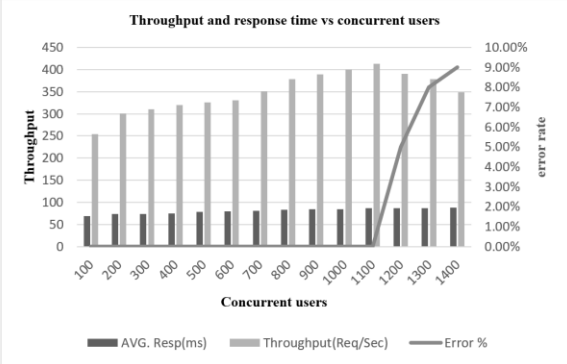

Figure 4: Performance After Adding Redis

High performannce is noted overall for Redis under high traffic API.Response times percentiles are less than 25ms up to 950 concurent users when using an external cache whereeas less than 35 for the same number of concurrent users when querying the gateway backend. It is also noted that in some cases, the gateway can peform without erros up to 1400 concurrent users. However, querying the gateway backend directly performs optiomally up to 950 concurrent users, after which, it generates backend errors, dropping new calls and thus responding faster. This is indicated by low resposne times after 950 concurrent users.

When traffic is low, that is, below 100, both approaches perform largely the same. Throughput and response are the same. Between 100 and 800 users, we notice that Redis improves response time by 5.6% and throughput by an average of 3.5%. This result is based on a personal laptop and if applied in an enterprise setup where the servers are clustered, the benefits are deemed to go higher. Secondly, the research found that without Redis, backend errors start occurring form 800 concurrent users. The backend errors imply the blocked API calls. When Redis is introduced, it was found that the API gateway can serve 1200 concurrent users without blocking API calls or generating backend errors. This is a 30% improvement from [4] who used request bundling technique which is found to be memory intensive according to [6] ,[7] and [13]and which cannot solve the unique case of distributed their party APIs.

Figure 5: Concurrency errors

5. SUMMARY

WSO2 API gateway is an open source platform that has been adopted by many organisations to expose their valuable monetary assests for third parties. Banks , telecommuications companies and gorvernment agencies adopt the platform to offer various digital services. However, the platform servers perfomance degradation under under pick time . This ranges from the complexities involving transformation of simpler Rest API to SOAP messages that are supported in the backend ,repeated user authentication using Oauth2 tokens and repeated static contents that are queires by the API consumers. This needs to be adddresed. Whereas the common approach to addressing performance bottlenecks is addition of computing resources , resouces are not infinite. It is therefore necessary for organisations to innovate various ways of addressing performabce challenges without necessarily expanding their ICT budgets. One of the solutions is to figure out on how to ensure repeat and static responses can be served without hitting the backend. In so doing, you free the backend and let it serve only new and unique requests.

Whereas most servers support internal caching mechanism, this research haa shown that internal caches perform poorly under high traffic. Similary , in a distributed setup, this is not possible because servers in the cluster do the caching individually. A central cache means that they share the tokens, keys , mediation flows and other static contents before they expire. More concurrent users mean more re-

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 11, November 2022, pp. 1 – 6

quests to the API manager gateway. Therefore, the throughput of the implemented WSO2 API Manager Gateway increases as the number of concurrent users accessing the APIs increases. The maximum throughput is obtained for 100 concurrent users for both "High performance mediation API" and "implemented WSO2 API manager," and the throughput degrades slightly after 100 concurrent users due to resource contentions in the system. The degradation point mainly depends on hardware resources.

6. CONCLUSIONS

The research has investigated the available optimization techniques namely: Memoization, caching using an external caching specialist platform and bundle splitting. Caching was chosen for this research. This addressed the first objective. The research introduced Redis server to aching tokens and repeat requests. As discussed, this has shown a performance improvement in terms of response time by 5.6% and throughput by 3.5%. This is in relative to the works of [15] who proposed a SOAP framework suitable for feature or low-end mobile devices, [14] whose work was based is a survey of the available caching mechanisms but the research doesn't give their effectiveness in various scenarios like distributed APIs. [1] has proposed mediation sequences using enterprise integration patterns, but the research falls short of the mechanisms for improving performance during mediation. This research builds a foundation for improving caching and performance during mediation operation. Although the works of [13] gives a framework for improving performance by scaling in-memory storage, the approach is too generic. For instance, the framework hasn't been tested on specialist and high throughput tools like Redis, Memcached and other proprietary tools. The framework proposed in this research is scalable, extensible and can be applied in unlimited number of caching and performance optimization platforms.

A. Areas of future research

This research focused on a passthrough API of same payload size. Future research will focus on XML to Json transformation in which caching of mediation headers and other static mediation contents is needed, and payloads are of varying length. Future research to also assess performance gains in a distributed setup where the gateway is load balanced using available load balancers like and enable the servers in a cluster to share the cache, compared to a scenario where the servers are using internal cache.

B. Abbreviations

API- Application programming interface.
Redis- an open source in-memory caching platform.
ICT- Information and communications technology.
SOAP- Simple Object Access Protocol.
Rest- Representation State Transfer.
WS02- An Open-source API gateway.
XML- Extensible Markup language.
JMeter- an open-source performance testing platform.
Oauth2- Open authentication standard version 2. For authenticating two sites without sharing password.
AVG- Average.
Req/Sec- Requests per second.
RMI- Remote method invocation.
JDBC- Java database connectivity.

7. DECLARATIONS

C. Ethics approval and consent to participate.

I hereby authorize the publisher to engage all kinds of ethical checks on the paper.

D. Consent for publication

I hereby authorize the publisher to publish this paper

E. Availability of data materials

The data obtained from the experiment has been supplied through online attachments. I will be available to provide any missing data artifacts.

F. Competing interests.

This is my original work and there are no competing interests whatsoever.

G. Funding

No funding nor sponsorship of any kind has been applied in this research.

H. Authors's contributions.

I, Stanley Ondero, is the author of the main author. Prof Wilson and Dr. Michael also played a significant role in guiding me through the research, giving me feedbacks and corrections.

I. Acknolwledgements

I acknowledge my supervisors listed above for their immense contributions in the research.

REFERENCES

[1]. Cloud, W. A. (2022, 1 26). *API mediation and sequence*. Retrieved from Add Sample mediation sequence: https://cloud.docs.wso2.com/en/latest/learn/message-mediation/add-a-sample-mediation-sequence/

[2]. Coleman. (2016). *APIs: Building A Connected Business in the App.*

[3]. E. E. (2020, August 27). Open Banking. *Technology and innovation: Bnaking APIs*, pp. 33-41. Retrieved from Investopedia: https://www.investopedia.com/terms/o/open-banking.asp

[4]. El, M., & Amine. (2021). Evaluation of API Request Bundling and its Impact on Performance of Microservice Architectures. *IEEE International Conference on Services*

ISSN: 2456 - 3935
International Journal of Advances in Computer and Electronics Engineering
Volume: 7 Issue: 11, November 2022, pp. 1 – 6

Computing (SCC 2021), (pp. 3-6). Virtual conference. doi:10.5281/zenodo.5087467

[5]. GUO FU, Y. Z. (2020). A Fair Comparison of Message Queuing. 1-3. doi:10.1109/ACCESS.2020.3046503

[6]. IBM Institute for Business Value. (2016). Evolution of the API economy. In I. I. Value, *Adopting new business models to drive future innovation* (pp. 1-2). Somers, NY 10589: IBM.

[7]. Kavita, D. H., & Rodd, S. (2021). Optimal web service composition using hybrid optimization algorithm in cloud environment. *Proceedings of the International Conference on Innovative Computing & Communication (ICICC) 2021.* doi:https://dx.doi.org/10.2139/ssrn.3834957

[8]. Martin, & R., A. (2021). PyPortfolioOpt: portfolio optimization in Python. *Journal of Open Source Software*, 6-61. doi:https://doi.org/10.21105/joss.03066

[9]. Max, M., Michiel, O., & Slinger, J. (2021). Focus Area Maturity Model for API Management. *Journal of Computer Science and Technology*, 11-130.

[10]. MCkenzie, & Cameron. (2020, April 1). *open API (public API)*. Retrieved from Tech Target Network.

[11]. Mifan, Careem. (2017, JUNE). Building an API Strategy Using an Enterprise API Marketplace., (pp. 45-53).

[12]. Nebro, A., Pérez-Abad, J., Aldana-Martin, J., & García-Nieto, J. (2021). Evolving a Multi-objective Optimization Framework. In E. Osaba, & X. (. Yang, *Applied Optimization and Swarm Intelligence. Springer Tracts in Nature-Inspired Computing* (pp. 175-198). Singapore: Springer. doi:https://doi.org/10.1007/978-981-16-0662-5_9

[13]. Romulo, M., & Ingrid, N. (2022, July 30). A Comparative Study of Application-level Caching Recommendations at the Method Level. (2-9, Ed.) *Empirical Software Engineering, 88*. doi:https://doi.org/10.1007/s10664-021-10089-z

[14]. S. Chen, X. T. (2016). Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis. *2016 IEEE Trustcom/BigDataSE/ISPA,*, (pp. 1660-1667). doi:DOI: 10.1109/TrustCom.2016.0255

[15]. Schubotz M, S. A.-P. (2022, April 22). Caching and Reproducibility: Making Data Science Experiments Faster and FAIRer. *National center for biotechnology information*. doi:https://doi.org/10.3389%2Ffrma.2022.861944

[16]. Singh, R. K., & Verma, H. K. (2022, April). Redis-Based Messaging Queue and Cache-Enabled Parallel Processing Social Media Analytics Framework. *The Computer Journal*, 843–857. doi:https://doi.org/10.1093/comjnl/bxaa114

[17]. Spal, G., & Kaur, J. (2018). In-Memory Data processing using Redis Database. *International Journal of Computer Applications*, 1-5.

[18]. Talaam, O., George, O., & Mike, K. (2018). A Simple Object Access Protocol(SOAP) Messaging for Mobile devices in android Platform. *International Journal for advance in computer science and technology*, 48-58.

[19]. Zhou, L., Lu, B., Zhang, S., & Qi, L. (2020, July). Data Cache Optimization Model Based on HBase and Redis., (pp. 31-35). doi:https://doi.org/10.1145/3414274.3414279

[20]. Zulfa, M. I., Fadli, A., & W, W. ,. (2020, April). Strategi caching aplikasi berbasis in-memory menggunakan Redis server untuk mempercepat akses data relasional. *Jurnal Teknologi dan Sistem Komputer*, 157-163. doi:https://doi.org/10.14710/jtsiskom.8.2.2020.157-163

Authors Biography

Stanley Makori Ondero, is a student at Jomo Kenyatta University of Agriculture and Technology. He completed his B.sc in computer Science form Egerton university in 2011. His research interests are artificial intelligence, distributed systems and programming language design.

Prof. Wilson Cheruiyot, is a professor, School of computer science and information technology at Jomo Kenyatta University of Agriculture and Technology. He completed his B.sc in computer Science form the same university in 1994. Thereafter, he completed master's in computer applications technology from Central South University of technology, China, in 2002 and Doctor of Philosophy in computer technology, specializing in intelligent agents, in 2012. His research interests are artificial intelligence, distributed and internet databases, systems and communication systems.

Dr. Michael Kimwele is a Lecturer in the Department of Computing, Jomo Kenyatta University of Agriculture and technology (JKUAT). He holds a BSc. Mathematics and Computer Science-First Class Honors from JKUAT (2002), a master's in information technology management from University of Sunderland UK (2006) and a Doctorate in Information Technology from JKUAT (2012). At present, he is the Associate Chairman, Department of Information Technology, JKUAT-Nairobi Campus. Dr. Kimwele has authored a commendable number of research papers in international/national conference/journals and supervises postgraduate students in Computer Science/Information Technology. His research interests include Information systems management, Information Technology Security, Electronic Commerce, Mobile Computing, Social implications of computer applications, Human Computer Interaction, and Computer Ethics.

Cite this paper:

Stanley Makori Ondero, Wilson Cheruiyot, Michael Kimwele, "An Approach to Performance Optimization of WSO2 Platform to Improve its Throughput", International Journal of Advances in Computer and Electronics Engineering, Vol. 7, No. 11, pp. 1-6, November 2022.