National College of Ireland

Dynamic Load Balancing of Microservices in Kubernetes Clusters using Service Mesh

Research Project
MSc Cloud Computing

Abhishek Sanjay Shitole
Student ID: x19206925

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Abhishek Sanjay Shitole
Student ID:	x19206925
Programme:	MSc Cloud Computing
Year:	2022
Module:	Research Project
Supervisor:	Rashid Mijumbi
Submission Due Date:	31/01/2022
Project Title:	Dynamic Load Balancing of Microservices in Kubernetes Clusters using Service Mesh
Word Count:	4935
Page Count:	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Abhishek Shitole
Date:	30th January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Dynamic Load Balancing of Microservices in Kubernetes Clusters using Service Mesh

Abhishek Sanjay Shitole
x19206925

Abstract

As web application hosting continues to grow over the cloud, the industry has now moved towards embracing the development of cloud-native micro-services-based applications. Such type of applications is generally deployed on Kubernetes as it offers greater benefits like reduced overhead, easy management, and faster development as several teams can develop and deploy individual services together. In the case of micro-service-based applications, the overall performance of the application is dependent on the performance of individual services. But as the workload on the application continues to increase the default Kubernetes load balancing strategy fails to manage the fluctuating traffic because of its static nature and performs poorly. Also, as many applications may reside onto the very same pod of the cluster, security becomes a big concern. To overcome these challenges, this paper proposes a technique that uses service-mesh Istio to inject sidecar proxies onto every micro-service and dynamically balances the load among services by applying service-specific routing through the Istio control plane. Inter-service communication is secured by encrypting the traffic among services by means of enforcing mTLS across all services. The experimental results have proved that the proposed design outperforms the traditional approach by maintaining stability and consistency in response rate and consumes fewer resources.

1 Introduction

Cloud computing is being immensely used in web-application hosting domain to provide On-demand computational resources. It offers essential characteristics such as sharing of resources, auto-scaling, higher security, simplistic use, lower costs, better performance, and fits practically all requests required in hosting a web-application. Currently there is a growing acceptance around core principles and informally web-application designing conventions which have been embraced and deployed in various successful cloud web-applications. Such applications developed on cloud are coined as Cloud-native applications. These cloud-native applications are mainly built upon micro-services architecture, as micro-services comprise of several separate applications which are independent from each other, they are light-weight and can be easily deployed and updated together or individually with minimal impact on users Gannon et al. (2017).
The use of container-based framework in application development has received a lot of attention recently because of its light-weight nature, platform independency, improved application performance and efficient resource sharing. Microservices are bundled within a container for easy management, reduced overhead and application development becomes faster as smaller teams can work independently on distinct portions of the same application, to produce high quality results and avoid code conflicts. An easier and efficient way of managing micro-services-based applications is thru a container- orchestration platform Kubernetes. The pod in Kubernetes consists of one or more docker containers which helps to schedule and manage shared-resources of containers encapsulated in a particular pod. Every container in a pod operates on the very same VM and uses similar IP and port number and thus they can discover each other by traditional methods as localhost Zhang et al. (2019). The storage space is also shared which is available locally to the pod.
Load Balancing is of utmost importance for a web application which is cloud-native, developed using micro-services, and hosted upon containers. The incoming workloads to such applications vary unpredictably because they provide a range of services, and each service consists of activities that may consume CPU, occupy storage, or use network resources, and it is extremely difficult to predict which service will be operating at any one time. The unpredictability of workloads poses a significant challenge for load balancing. In an ideal case, the load balancer identifies underused and oversubscribed nodes and then regulates the traffic so that workloads are shifted from an overloaded node to an under burdened server without affecting the present application's continuing duties. This strategy maintains the burden whenever it hits a critical threshold. Workloads are effectively distributed across nodes using this strategy, enhancing throughput while minimizing wait and response time McDaniel et al. (2015). However, in a microservices environment, one service may be dependent on one or more other services to perform its intended functions. As a result, the overall performance of an app is dependent on the performance of all separate services.
Containerized platforms such as Kubernetes fail to address issues such as dynamic workload distribution and assure optimal performance in cases of service dependency as Kubernetes has a simple round robin strategy that requires to be updated every time in accordance with the operating situations of the application. Kubernetes uses iptables and to store load balancing rules. Adding far too many rules in iptables increases the searching speed and requests allocation time. Also, as two or more applications reside on the same pod, malicious access to the pod may leads to compromising of these applications.

1.1 Motivation

Introducing dynamic load distribution algorithms in the ingress controller of Kubernetes to reduce the potential overheads and improve the RPS for the micro-services application are showing great results Zhang et al. (2018). Strategies to label the incoming requests A Dua et al. (2020) and Use of custom container-based load balancer Takahashi et al. (2018) on Kubernetes have proven to be effective by improving the application's portability without sacrificing the performance.
However, whenever the bulk traffic develops on the application, the requests among these micro-services expand tremendously which requires specialized routing policies to be applied for optimizing the flow of data among the services while encrypting the requests among micro-services Abidi et al. (2019). The proposed approach uses Service mesh Istio which provides dynamic load-balancing among services by applying service-specific routing policies. Service-to-service communication is secure by using mTLS which encrypts the requests among services.

1.2 Research Question

How can varying requests on an web-application using micro-services architecture and hosted on Kubernetes clusters be managed effectively using real-time load balancing techniques ?

1.3 Structure Of The Paper

This research paper is further divided into 6 sections. Section 2 presents the literature review in the sector of dynamic loading balancing in containers and micro-services. Section 3 talks about the research methodology and outline of proposed system. Section 4 explains the design specifications and architecture of the system.Section 5 discusses stepwise implementation of proposed approach.Section 6 and 7 presents the experimental results and conclusion accordingly..

2 Related Work

At the present, there is a lot of research-work & studies being carried out on Kubernetes & Micro-services.Undoubtedly developing micro-services based applications and deploying it on cloud provides great performance benefits across the board.This approach might be adopted across all entire IT industry.This section illustrates various studies

2.1 Dynamic Load balancing techniques introduced in Kubernetes

The default load distribution strategies used by Kubernetes is not suitable for handling dynamically inbound bulk requests because of slow/improper workload distribution among nodes. So, a dynamic load balancing algorithm in ingress controller of Kubernetes was introduced by Zhang et al. (2018) which assesses cpu, storage and network bandwidth of nodes and priority, resource weight of incoming requests before allocating them. The proposed algorithm also learns about service dependency among micro-services prior

distribution. Similar approach was followed by Qingyang Liu et al. (2020) to use a custom load balancer which assigns requests based on every node's current operating state and weights of incoming requests. The above provided approaches help in reducing the potential overheads and improve the RPS.

Due to the inherent design of Kubernetes, the master node carries a huge burden as it is solely responsible for updating the data and replicating it to the worker nodes. The default master selection mechanism cannot equivalently distribute the master among nodes. As a result, user requests are concentrated on a single node, causing system congestion. Nguyen and Kim (2020) proposed a leader-based consistency management strategy for state-full applications which requires provisioning of main leader container for handling user requests and dividing work among slaves. The other container to choose leader amongst the available replicas. The main container realizes its status (master or slave) and therefore will handle the incoming requests correspondingly. The receiving pod will serve the read requests regardless of its location. Parallelly, the leading unit processes only write operations. So, the write operations are transmitted to the masters if the receiver pod is a following. The author's solution helps to maintain consistency in Kubernetes cluster by distributing the number of masters equally amongst cluster nodes. As a result, bulk requests are handled faster and performance is increased.The BLD (balanced distributed leaders) algorithm proposed by T. Nguyen and Kim (2021) improves Kubernetes' standard leader election process by evenly distributing leaders among nodes. All replica pods start as followers and update the information in Leader Management Endpoint object (LMEP) and Endpoint object (EP) to become current leaders. If the criteria of BLD algorithm is satisfied they become leaders or else continue as followers and seek to become leader whenever the conditions are met. The leaders need to keep updating the information on frequent bases to be leaders. By following this algorithm, the researchers improved throughput and latency of the applications.

A Dua et al. (2020) created a method to handle typical load balancing difficulties such as fair allocation of inbound traffic to cluster nodes as systems get more complex, with processors with variable capacities, loads, and configurations. The author presents a task scheduling system and creates clusters dedicated to a certain type of job (dynamic data, datacentric, etc.) by giving tags to each work in order to identify it. The approach is then tweaked to add load-balancing measures like task transfer. To assess the efficacy of this strategy, thousands of jobs with varying expected execution time and inputs were tested. All in all, Cluster processing time has been cut in half, and throughput has increased dramatically.Kimitoshi Takahashi et al. (2018) used the Linux kernel's Internet Protocol Virtual Server (IPVS) to develop a container-based software load balancer that can be run on Kubernetes cluster. The proposed method proved to be effective where Kubernetes is incompatible with the supplied load-balancer such as on-premises data-centers with physical load balancers and so movement of nodes across different environments is a problem. The author's method gets the best performance by supposing excellent overlay by selecting network operating mode and distributing processing of packets across many cores. The results conducted by researcher revealed that the proposed IPVS load balancer improved application's portability without sacrificing performance.

2.2 Load Balancing in Microservices

Chang Yi et al. (2018) investigated many coarse grain load-balancing strategies and provided an unique dynamic weighed load balancing strategy based on micro-services clustering frameworks that solves incorrect resource consumption among all nodes of cluster. Researcher claims that the fundamental source of the problem is that each node cannot process user traffic fairly, and that some nodes are functional but others aren't. As a result, the cluster's overall performance drops significantly. The author's proposed algorithm uses data from seven physical servers and allocation of resources amongst them, as well as metrics such as multiple phases among several server interconnection and the volume of memories that are used, to provide an accurate approach for dissmenating loads among various servers. When compared to other algorithms, this method has a higher throughput and a shorter reaction time.

As per Ruozhou Yu et al. (2019), the most challenging issue for microservices architectures is dependency between service instances and controlling loads on these services. The author employed a directed acyclic graph (DAG)-based technique to characterize the interconnectedness of micro services and to equalize the loads by reducing the pressure on the services with the greatest loads. This proposed approach tackles the issue of load-balancing as a means of improving QOS. Yipei Niu et al. (2018) has made important contributions to load-balancing in microservices systems. The author created a comprehensive model known as a chain orienting load-balancing algorithm (COLBA), which assists in the management of load balancing among micro-services by assessing different aspects such as diversified requests and inter competitiveness. The researchers revealed that by carrying out experiments and comparing their proposed approaches to standard load balancing strategies, total responsiveness in COLBA was reduced by 13 percent.

Rusek and Landmesser (2018) investigated many load-balancing algorithms and applied them to micro services running in virtualized containers, demonstrating how the decentralized swarm-like model surpasses other strategies. According to the authors, virtualized containers are great examples of understanding the services as they enhance the efficacy of containers which operate across different systems, and the containers in this scenario act like a bunch of agents. The analysis gives conclusions based on current knowledge as well as numerical-based simulation, demonstrating that using load dispersing ways in cloud environments surpasses other existing methodologies.'Hong Zhu et al. (2018), analysed variety of load balancing algorithms used in microservices with a particular emphasis on the influence of scheduling strategies on service capacity and scalability. The researcher found out that the round-robin approach is much more scalable and achieves high processing capabilities when compared to other approaches such as workload-aware.

2.3 Containerized Micro-services

IntMA, a unique, scalable heuristic technique was proposed by Christina 'Joseph and Chandrasekaran (2020) for launching microservices in an interaction-aware way using connection information received from the Interactions Graph. The Author implemented the strategy on Kubernetes platform and measured efficiency using various different sample micro-services applications which are easy to host on GCP. The findings showed that proposed strategy is successful in improving micro-based system's response time and throughput. Maria Fazio et al. (2016) highlight issue related to microservices whenever they are hosted on cloud platforms like AWS, Azure, etc are dependent on various vendor

provided functionalities like load balancer, proxy server and databases. Thus, varying configurations of microservices might affect application performance and so author suggests there should be automation in selecting micro-services configuration. Additionally, the authors indicates that the most challenging part in developing micro-serviced specific models are learning and fitting of functions to monitor metrics like request arrival, CPU utilization, memory consumed, etc. Leila 'Abdollahi Vayghan et al. (2018) evaluated high availability (HA) in Kubernetes clusters by deploying web-applications based micro-services architecture and observed that initially Kubernetes behaves effectively when failure is caused by external events. But in subsequent tests, the outage observed was 5 minutes and HA isn't triggered because the requirements aren't met. Additionally, the default configuration needs to altered in such a way that it will minimize the network overhead and false positives.

'Rajavaram et al. (2019) conducted research on implementing the CI/CD pipeline and projecting the different ways where the microservices can be continuously deployed. They did this by using the Kubernetes to deploy the microservices where the application uses the high availability. It is a method where, Kubernetes is used with the help of Rudneck plugin to deploy the microservices on the cluster. According to this research, the features, and the integration with Rudneck, Kubernetes is the best practical approach for orchestration. Khaleq and Ra (2021) did their study on microservices autoscaling based on the QoS for cloud systems. The study was done to demonstrate that, the autoscaling done by the pod resource will generate high response that the generic CPU based scaling. They concluded by showing the autoscaling of the microservices in cloud with real time application where the time is main quality of service. They also concluded from the analysis that, the autoscaling is to be done with a good understanding as the behavior of the microservices vary. Abdollahi Vayghan et al. (2019) researched on previously studied the deployment of the stateless microservices with Kubernetes. In this paper, they provided a model to deploy the stateful microservices application with the help of Deployment controller and Kubernetes. They showed the availability that is provided by the Kubernetes for the stateful microservices. They provided a solution where the redirection of services is done through the healthy pods. This solution provided the handling of the failure at any platform.

2.4 Overview on Service Mesh

Lim et al. (2021) In his research, the author used a service mesh framework Istio for effectively managing services and controlling traffic in VM and container systems. An orchestrating software based on service-mesh was used in automating the proxy deployment for VM on the LCM side. The services were configured by specifying deployments, network connectivity based upon microservices structure. The author carried out experiments by programming codes with on open-source software's such as Openstack, Kubernetes and Istio and observed.

The default scheduling in Kubernetes is static in nature, to solve the issue Wojciechowski et al. (2021) offers NetMARKS which is a unique solution to Kubernetes pod scheduling that makes advantage of dynamic network measurements gathered by Istio Service Mesh. NetMARKS makes scheduling decisions based on data acquired by Prometheus. This technique enhances Kubernetes scheduling while being completely backward compatible. The author verified the approach by running it through various workloads and processing configurations. According to the findings, NetMARKS can cut application's response

time by up to 37 percent and saves almost 50 percent of cross-node bandwidth in a completely autonomous way.
When monitoring and controlling network and application traffic through service mesh istio, the application sensitive information might get exposed to external world which is a security risk. As containers consists of only one network interface, there is not possibility to separate application traffic from others. To resolve this issue, Kang et al. (2019) presented a secure coordination system for container-based three tier service traffic that uses en-/decapsulation for traffic division and encrypting. the researcher was able to successfully deliver function-level traffic monitoring and management of comprehensible metrics for functional health. It may also withstand the testing of function/proxy failures and assist containers management in re-covering impacted services and connected monitoring agents.

2.5 Summary of Literature Review

Reference	Algorithm/ Framework	Approach	Advantages	Limitations
Zhang et al. (2018)	Endpoint Assessment	Assess endpoints according to service dependency	Improved response time	High Overhead
Nguyen and Kim (2021)	Balanced Leader Distribution Algorithm	Evenly distribute the leaders throughout nodes in the cluster	Enhances the throughput	Increased Latency
Dua et al. (2020)	Scheduling and Migration Algorithm	Label tasks according to sizes	Increased Throughput	High Response Processing Time
This Research	Service Mesh Istio	Define routing policies in YAML files	Low Response Time, improved security	Increases complexity

Table 2.1: Summary of Literature Review and Research Niche.

After reviewing many techniques and algorithms offered by researchers for dynamically balancing incoming traffic across Kubernetes clusters, the majority of researchers recommended new algorithms to be deployed over Kubernetes proxy or by replacing the proxy

with a bespoke load balancer. Their primary focus was always on appropriately dispersing the load by utilizing every resource to its best extent. Almost all of the studies failed to pay attention to specific features such as service dependency and imposing security on micro-services within a cluster, detailed request tracking, and monitoring important metrics such as HTTP failures, agent status, latency, and resource consumption, and centrally adjusting load-balancing policies without modifying the actual code.

To address the aforementioned concerns, my proposed approach considers service dependency across micro-services, and as a result, the deployment types and routing information for each service to be hosted on Istio are provided for better load-balancing and resource utilization. Additionally, security is given by authenticating each service and encrypting traffic between them using mTLS, which is enforced by centrally applying a PeerAuthentication security policy to all services. Kiali tracks all incoming requests and their actions, and Prometheus monitors essential metrics such as HTTP failures, individual service status, latency, and resource usage, which are shown in the Grafana dashboard.

3 Methodology

This section describes methodology of the research. Research flow is explained in 3.1 and various tools and technologies used in this research are discussed in 3.2.

This research contributes towards dynamic load balancing of micro-services in Kubernetes clusters by making use of open-source service mesh framework Istio. The proposed solution in this paper solves issues in Kubernetes such as limited load balancing, tracking service latencies problems and missing security among services. Variable Incoming requests are evenly distributed among all available services with mTLS enabled across every micro-service which provides extra layer of security. The proposed system also achieves better performance and lower response time when compared with Kubernetes. In a nutshell, the proposed solution fills a gap in the Kubernetes ecosystem which was previously lacking.

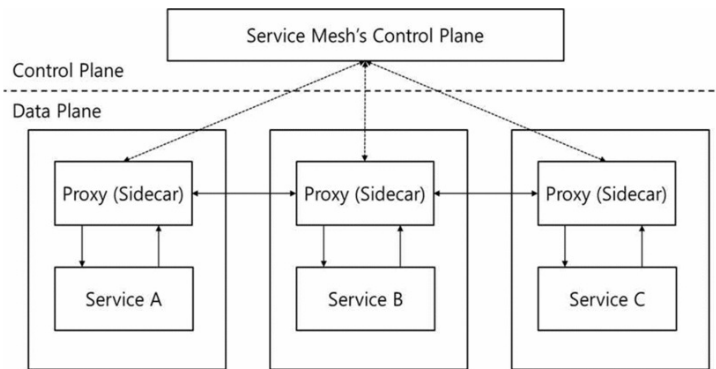

Figure 1: Basic Architecture of Service Mesh(Cha and Kim (2021))

Figure 3 shows how service mesh separates the data plane and control plane and isolates individual services. Furthermore, sidecars which act like proxies are injected in every service and are responsible for forwarding requests to other sidecar. All the routing policies for proxies, security functions, service registration and other function are handled by control plane.

3.1 Research Flow

All activities in this research were conducted on a 3-node Kubernetes cluster deployed on Google Kubernetes Engine (GKE). The application based on micro-services is taken from Istio website Istio (2021) and modified to fit the requirements before deploying it onto the cluster. Istio is then installed on top of the Kubernetes container and then the YAML file for virtual service is created along with defining load balancing and security policies before injecting the sidecar proxies onto pods. Prior to exposing the application to external traffic, Kiali, Prometheus and Grafana are deployed onto Istio for monitoring and recording metrics of the application. Load testing on micro-services application is performed using Locust tool.

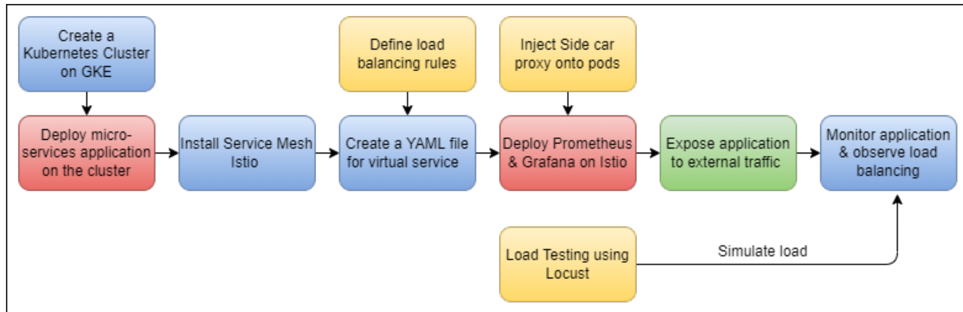

Figure 2: Flow Diagram

3.2 Tools and Technologies Used

This research was carried out on Google Kubernetes Engine with machine type e2-standard-4 (Ubuntu). A three-node cluster was provisioned with one leader and two follower nodes. The micro-services web-application is developed in Python language. The following technologies and tools were employed:

- Kubectl: This research used kubectl as a command-line interface for communicating with and managing pods in the Kubernetes cluster. All the deployments and configurations were also applied using this entity.
- Istio: Service Mesh tool Istio with version 1.12.1 is used in the study to manage microservice communication and data sharing. After installation, istio was used to inject proxies in Kubernetes pods and for routing traffic to the relevant service while enforcing policies.
- Istioctl: The study used Istioctl for debugging and diagonalizing installation issues within service mesh. It is also used to launch dashboards for kiali and grafana when deployed in istio-system.
- Kiali: This research uses kiali for visualizing inbound traffic on service mesh and for identifying service health issues quickly. It builds a topology of the entire system and displays real-time traffic analytics on it.
- Prometheus Grafana: The combination of both open-source tools is used in research for real-time monitoring of incoming HTTP requests and cluster resources. While

Prometheus fetches data from Istio's ingress controller other sources, Grafana Dashboard displays detailed insights like response time, CPU usage, memory utilization for inbound requests, clusters, nodes, and pods.

- Locust: It is a performance testing tool that has been used in research for generating load over the application so that the load balancing capabilities of the proposed system can be tested.

4 Design Specification

In this section, the design specifications of the project are discussed. Subsection 4.1, describes the required system configuration for running the project, and system architecture is explained in Subsection 4.2.

4.1 Required Specifications for System

Google Kubernetes Engine used in this research is bundled with pre-configured docker version 20.10.11 and kubernetes version 1.21.5.Kubernetes uses kubectl for managing all operation and deployments within a container.Generally kubernetes places micro-services on different pods for loose coupling and independent scaling.Micro-services can be restarted with ease in case of unforeseen failures.Below Tables 2 & 3 depicts cluster configuration and software configurations required to run this project.

Kubernetes Cluster on GKE	
Machine type	e2-standard-4
No of Nodes	3
Operating System	Debian Version 10
Total vCPU	12
Total Memory	48 GB
Cost	$0.134012/hr per node

Table 2: Required Cluster Configuration.

Software	Description
Service Mesh	Istio 1.12.1
Kiali	Requires Istio installed
Locust	Requires Cluster with 3 node
Manifest language	YAML

Table 3: Required Software Configuration.

4.2 System Architecture

The architecture of this research is composed of three important components: A bookstore application based on microservice architecture, Kubernetes clusters, and the service mesh Istio. Every task is aided, acquired, is carried out on the Kubernetes cluster is provided by Google Kubernetes Engine, a PaaS platform. The high-level design of our proposed model is depicted in Figure 4.
The microservices ProductPage, Details, Ratings, and Review work collaboratively to provide a tenuously connected bookshop application that operates on Kubernetes containers.Envoy proxy is injected as sidecar onto every pod that runs a service. All the routing policies for proxies, security functions, service registration, etc defined by control plane are enforced onto these sidecars and they perform actions accordingly. The control plane gathers telemetry data and functions as the brain of Istio.
A plethora of monitoring and controlling capabilities using Prometheus and Grafana are introduced to the core operational control-plane for detailed monitoring of cluster resources and the incoming requests on the system. Kiali provides visualization of real-time inbound traffic and creates a topology for better analytics and identifying health issues within the model.In comparison to conventional methods of deploying an application, the proposed system does it in a lot easier and more efficient manner.

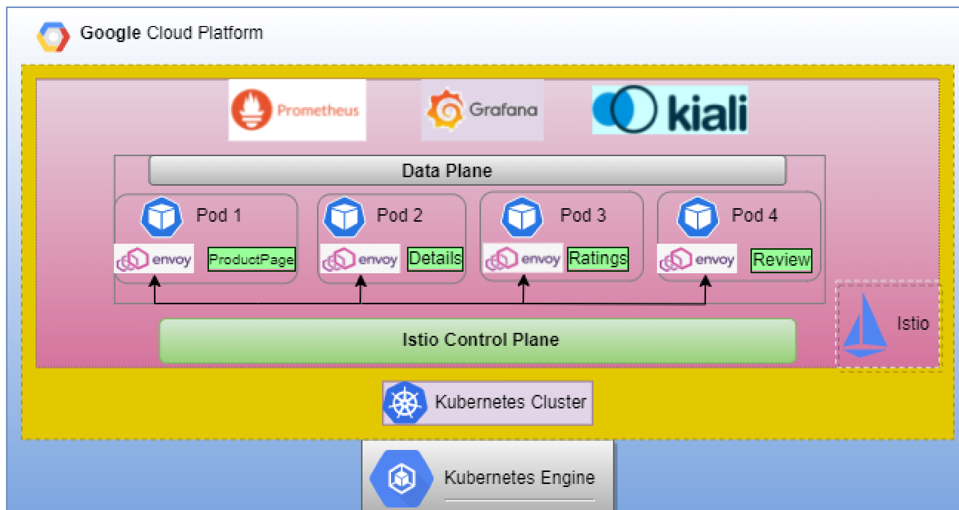

Figure 3: High Level Design of proposed model

5 Implementation

The proposed system's implementation was finished with the order depicted in Figure 3 Flow Diagram. The bookstore application using micro-services architecture runs on three nodes Kubernetes cluster. The Istio-system namespace is created for keeping Istio configurations separate. On this namespace, the network configs (YAML file) are applied on the ingress-gateway. The ingresshost, ingressport, and secureingressport are configured in Istio-ingressgateway. This gateway plays an important role of a load balancer by distributing the incoming load among the various application services. Firewall rules are created in Google Cloud Platform in order to allow the ingress port and secure

ingress port which will be used by Istio-ingressgateway. All the dynamic routing and security policies that are to be applied among individual services are written in a YAML file named as VirtualService and Enable-mTLS respectively. It is then applied onto every sidecar present in every pod through control plane. Locust tool generates load on the bookstore application by simulating users that are predefined by us. Thus, we use this tool for performance testing of the application and the proposed system. As shown in Figure 5, Kiali displays the topology diagram of the system based on the real-time incoming workloads over various services. It also highlights important metrics like bandwidth, throughput, Request per second, and the amount of load distributed to individual services. Additional details like resource consumption, gRPC requests, CPU usage, memory usage, etc can be seen after logging onto Grafana Dashboard.

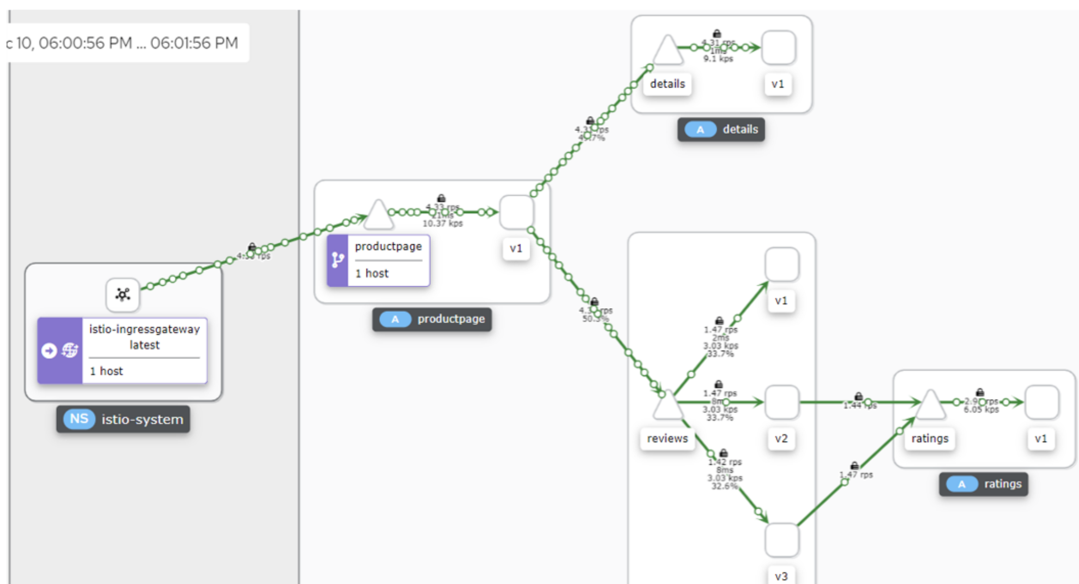

Figure 4: Real-time traffic Visualization thru Kiali

6 Evaluation

In this section, the performance evaluation is performed by conducting various experiments on the proposed system and comparing these results with the default Kubernetes system. The proposed system uses Google Kubernetes engine to run Kubernetes three-node cluster and to host the micro-services web-application for evaluation and experiments. The Default Kubernetes system which has been used for comparison was developed with the exact same configurations as that of the proposed system.
The experiments were conducted by generating load for five minutes under three different scenarios (as shown in below table 4). Using the load testing tool locust, we simulate users and the amount of incoming requests per second on the micro-serviced based web application. Metrics such as response time, cpu utilization, and memory consumption were observed and recorded with the help of Kiali and Grafana.

Scenario	Total Users	Requests per second
1	1000	100
2	3000	300
3	5000	500

Table 4: Scenarios for experiments

6.1 Experiment 1

For the first experiment, we have defined total user and requests per second according to scenario 1 as shown in Table 4.From below figure, it can be observed the average response time (green line) and the 95th percentile response time (Maximum Response time for 95% requests) for both the systems (proposed default) is nearly the same. But, the cluster with default kubernetes system consumes more CPU and memory (30% & 10.9%) as compared to proposed system.

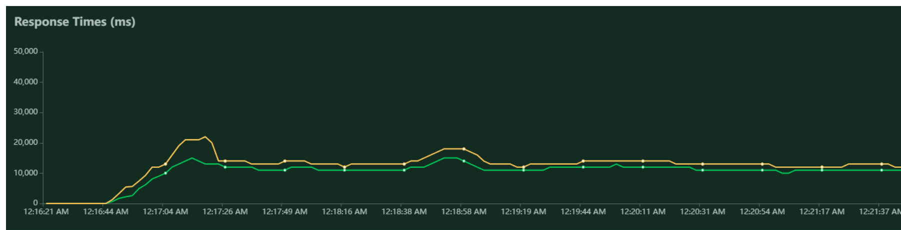

Figure 5: For Kubernetes-based system

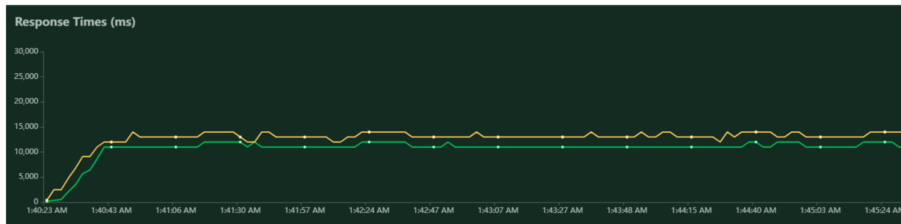

Figure 6: For Istio-based system

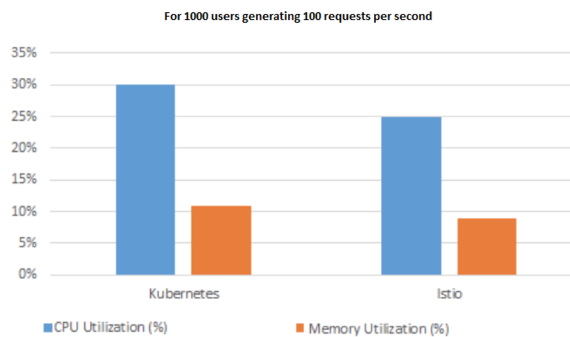

Figure 7: Comparison of Resource consumption

6.2 Experiment 2

For the second experiment, we have applied configuration stated from scenario 2 as displayed in Table 4.From Figure it can be noticed that, the response time for first few seconds for both the systems are zero, which might mean that when the web-application was unresponsive at start.After 1 minute of receiving load, the response time for proposed model (24464 ms) was less than that of the default kubernetes model (34925). The CPU usage and memory consumption for default system continued to remain more than that of the proposed system as shown in figure.

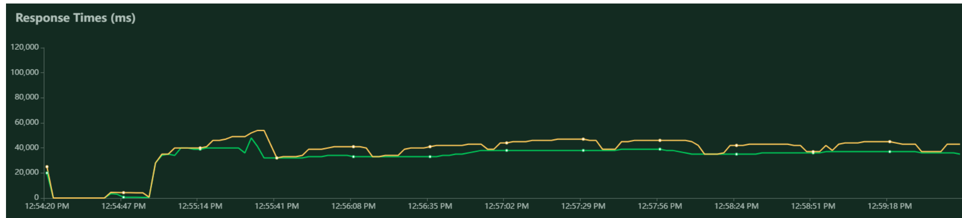

Figure 8: For Kubernetes-based system

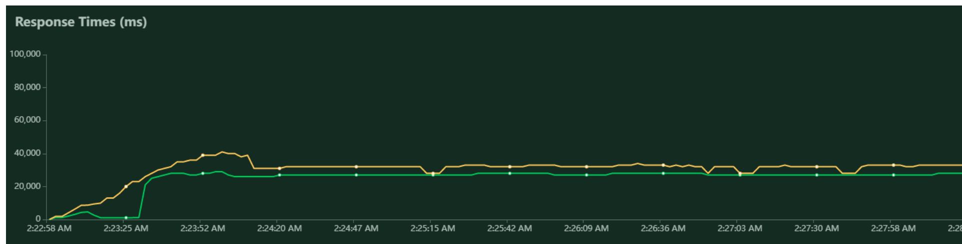

Figure 9: For Istio-based system

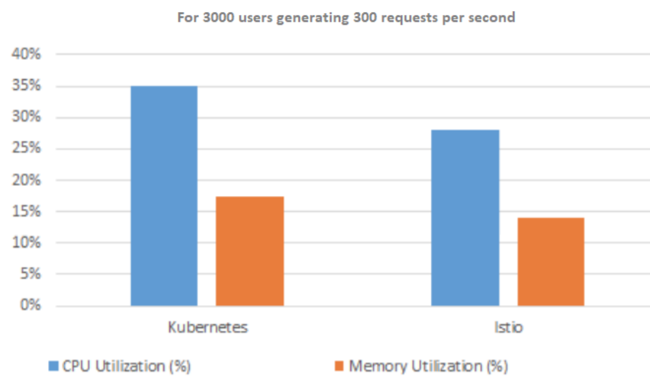

Figure 10: Comparison of Resource consumption

6.3 Experiment 3

For the third experiment, we selected values as given in scenario 3 of Table 4.It can be witnessed from the start, that the application hosted on default kubernetes system is struggling with varying response times when 5000 users with 500 requests per second are deployed, Whereas the Istio based proposed system remains stable. Also the CPU utilization for default kubernetes cluster shoots above 40 % and memory nearly 25 %, but the CPU usage and memory consumption for Istio based cluster continues to remain lower than that of the default model.

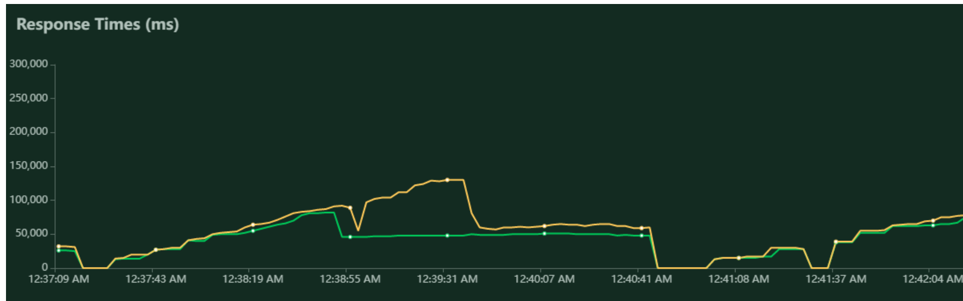

Figure 11: For Kubernetes-based system

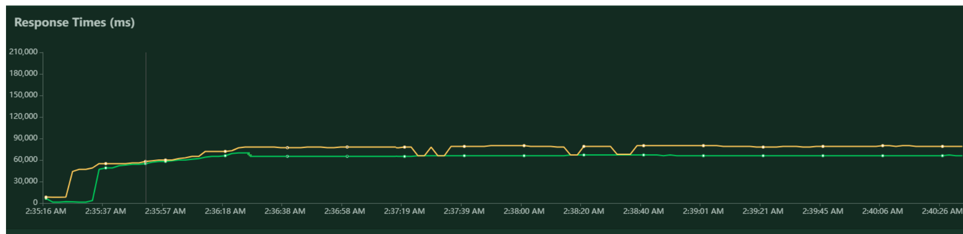

Figure 12: For Istio-based system

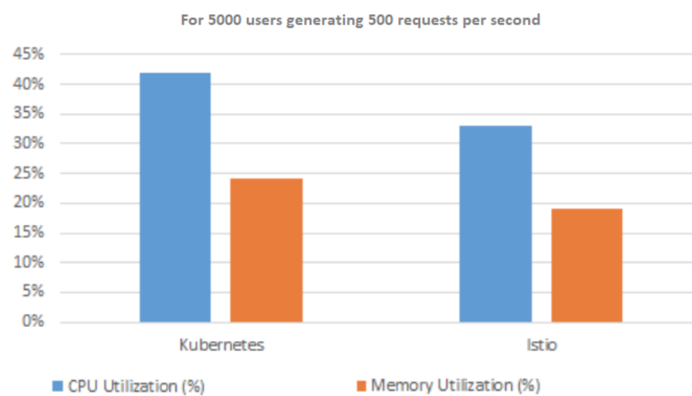

Figure 13: Comparison of Resource consumption

6.4 Discussion

From the above-conducted experiments on the proposed istio-based system and the default kubernetes-based system, it can be said that the response time is nearly the same for both the models, when there are a smaller number of users and fewer incoming requests per second on the application (scenario 1). The proposed design consumes fewer resources as compared to default design in this scenario. But when the number of users and requests gradually increase (scenario 2 3), the default Kubernetes system begins to struggle, thus delivering higher response time, failed requests, and consuming more CPU and Memory. Whereas the proposed istio-based system outperforms by maintaining stability and consistency in response rate and consumes fewer resources compared to the default one.
The existing web applications based on micro-services architecture can introduce the proposed istio-based design as it is capable of efficiently balancing the bulk web requests among the micro-services and improves the response time of which is critical for any application. The proposed design also consumes fewer resources when compared to other models, thus saving any additional cost of buying or provisioning extra resources.

7 Conclusion and Future Work

This Research proposes a service-mesh Istio-based system, which efficiently handles the varying requests on a web application using micro-services architecture and hosted on a Kubernetes cluster by ,improving the response time and consuming fewer resources as compared to traditional kubernetes system as shown in the evaluation part.This study is also successful in applying dynamic load balancing policies directly onto the individual micro-services of an application with the help of Istio control plane by the method of injecting sidecar proxy onto every micro- service of the application. Furthermore the security issue among individual micro-services deployed on the same pod of kubernetes is solved by the means of implementing service authorization through mTLS by encrypting traffic.Detailed metric tracking of incoming requests and resources can be done with the help of kiali and grafana.
Container complexity in some situations adds a compute overhead and increases the latency when running the proposed istio-based system. Further research to reduce the latency and complexity can be done.End-to-end tracing of every service outside the scope of proposed istio-based system cannot be measured.Therefore by introducing tools such as dynatrace or Jaeger can enabled end-to-end tracing and are beneficial when troubleshooting issues.

References

Abdollahi Vayghan, L., Saied, M. A., Toeroe, M. and Khendek, F. (2018). Deploying microservice based applications with kubernetes: Experiments and lessons learned, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 970–973.

Abdollahi Vayghan, L., Saied, M. A., Toeroe, M. and Khendek, F. (2019). Microservice based architecture: Towards high-availability for stateful applications with kubernetes,

2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), pp. 176–185.

Abidi, S., Essafi, M., Guegan, C. G., Fakhri, M., Witti, H. and Ghezala, H. H. B. (2019). A web service security governance approach based on dedicated micro-services, *Procedia Computer Science* **159**: 372–386. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 23rd International Conference KES2019.
URL: *https://www.sciencedirect.com/science/article/pii/S1877050919313742*

Cha, D. and Kim, Y. (2021). Service mesh based distributed tracing system, *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1464–1466.

Dua, A., Randive, S., Agarwal, A. and Kumar, N. (2020). Efficient load balancing to serve heterogeneous requests in clustered systems using kubernetes, *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–2.

Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L. and Villari, M. (2016). Open issues in scheduling microservices in the cloud, *IEEE Cloud Computing* **3**(5): 81–88.

Gannon, D., Barga, R. and Sundaresan, N. (2017). Cloud-native applications, *IEEE Cloud Computing* **4**(5): 16–21.

Istio (2021). Sample bookinfo application.
URL: *https://github.com/istio/istio/tree/master/samples/bookinfo*

Joseph, C. T. and Chandrasekaran, K. (2020). Intma: Dynamic interaction-aware resource allocation for containerized microservices in cloud environments, *Journal of Systems Architecture* **111**: 101785.
URL: *https://www.sciencedirect.com/science/article/pii/S1383762120300758,*

Kang, M., Shin, J.-S. and Kim, J. (2019). Protected coordination of service mesh for container-based 3-tier service traffic, *2019 International Conference on Information Networking (ICOIN)*, pp. 427–429.

Khaleq, A. and Ra, I. (2021). Intelligent autoscaling of microservices in the cloud for real-time applications, *IEEE Access* **PP**: 1–1.

Lim, H., Kim, Y. and Sun, K. (2021). Service management in virtual machine and container mixed environment using service mesh, *2021 International Conference on Information Networking (ICOIN)*, pp. 528–530.

Liu, Q., Haihong, E. and Song, M. (2020). The design of multi-metric load balancer for kubernetes, *2020 International Conference on Inventive Computation Technologies (ICICT)*, pp. 1114–1117.

McDaniel, S., Herbein, S. and Taufer, M. (2015). A two-tiered approach to i/o quality of service in docker containers, *2015 IEEE International Conference on Cluster Computing*, pp. 490–491.

Nguyen, N. D. and Kim, T. (2021). Balanced leader distribution algorithm in kubernetes clusters, *Sensors* **21**(3).
URL: *https://www.mdpi.com/1424-8220/21/3/869*

Nguyen, N. and Kim, T. (2020). Toward highly scalable load balancing in kubernetes clusters, *IEEE Communications Magazine* **58**(7): 78–83.

Niu, Y., Liu, F. and Li, Z. (2018). Load balancing across microservices, *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 198–206.

Rajavaram, H., Rajula, V. and Thangaraju, B. (2019). Automation of microservices application deployment made easy by rundeck and kubernetes, *2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pp. 1–3.

Rusek, M. and Landmesser, J. (2018). Time complexity of an distributed algorithm for load balancing of microservice-oriented applications in the cloud, *ITM Web of Conferences* **21**: 00018.

Takahashi, K., Aida, K., Tanjo, T. and Sun, J. (2018). A portable load balancer for kubernetes cluster, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, Association for Computing Machinery, New York, NY, USA, p. 222–231.
URL: *https://doi.org/10.1145/3149457.3149473*

Wojciechowski, , Opasiak, K., Latusek, J., Wereski, M., Morales, V., Kim, T. and Hong, M. (2021). Netmarks: Network metrics-aware kubernetes scheduler powered by service mesh, *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–9.

Yi, C., Zhang, X. and Cao, W. (2018). Dynamic weight based load balancing for microservice cluster, *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*, CSAE '18, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3207677.3277955*

Yu, R., Kilari, V. T., Xue, G. and Yang, D. (2019). Load balancing for interdependent iot microservices, *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 298–306.

Zhang, F., Tang, X., Li, X., Khan, S. U. and Li, Z. (2019). Quantifying cloud elasticity with container-based autoscaling, *Future Generation Computer Systems* **98**: 672–681.
URL: *https://www.sciencedirect.com/science/article/pii/S0167739X18307842*

Zhang, J., Ren, R., Huang, C., Fei, X., Qun, W. and Cai, H. (2018). Service dependency based dynamic load balancing algorithm for container clusters, *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*, pp. 70–77.

Zhu, H., Wang, H. and Bayley, I. (2018). Formal analysis of load balancing in microservices with scenario calculus, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 908–911.