

Weighted Load Balancing Mechanisms over Streaming Big Data for Online Machine Learning

Petros Petrou
UBITECH LTD
Thessalias 8 and Etolias 10,
Chalandri, 15231, Greece
ppetrou@ubitech.eu

Sophia Karagiorgiou
UBITECH LTD
Thessalias 8 and Etolias 10,
Chalandri, 15231, Greece
skaragiorgiou@ubitech.eu

Dimitrios Alexandrou
UBITECH LTD
Thessalias 8 and Etolias 10,
Chalandri, 15231, Greece
dalexandrou@ubitech.eu

ABSTRACT

A growing number of complex applications, such as cloud and / or mobile computing, video on-demand and streaming big data analytics are influenced by the growth of users, devices and connections. At the same time, the microservices architecture paradigm enables software developers divide their application into small, independent, and loosely coupled services that can be hosted on multiple machines, thus enabling horizontal scale up. In this paper, we study how a weighted load balancing approach can improve application performance and online machine learning over streaming big data, specifically in Kubernetes-based environments. We introduce an automated process which prioritizes events by efficiently managing the communication among the interacting services (i.e. pods) through adaptive traffic routing and dynamic rules enforcement allowing to control the flow of data and API calls among them. This process guarantees services stability and autoscaling at runtime. We demonstrate the proposed approach in a prototype microservices application consisting of containerized and deployed pods on Kubernetes, named as *Information Aware Networking Mechanisms*. These mechanisms have been integrated into a sophisticated framework which takes into account the number of requests per second or the volume of data per hour and supports weighted load balancing mechanisms to minimize inter-pods communication and prioritize important events to an online machine learning model which is crucial for the examined application. The Information Aware Networking Mechanisms have been openly made available for deployment and experimentation to the research community to build upon.

1 INTRODUCTION

Globally, devices, connections and applications are growing faster (10% CAGR) than both the population (1% CAGR) and the Internet users (6% CAGR) [4]. This increasing use of services functioning over online and offline processing modes in COVID-19 era and the rising demand for big data analysis require optimizations and efficient traffic management mechanisms.

The existing and emerging trend in Information and Communication Technology (ICT) is towards high performance and robust applications featuring higher network speeds at the infrastructure level in order to fulfill the daily communication and business needs. This trend is accelerating the increase in the average number of devices and connections per user. Each year, new devices with various functionalities, increased capabilities and intelligence are introduced and adopted in the market. In addition to this, a growing number of complex applications, such as cloud-mobile computing serving web-mobile applications, Ultra-High-Definition (UHD) video on-demand and big data analytic applications are influenced by the growth of users, devices and

connections. Therefore, it is important to multiplex the changing variance of user growth, devices, connections and data traffic in multi-device ownership. Multimodal web-mobile or UHD video applications, in particular, may have a multiplier effect on traffic which can only be controlled by high-capacity engineering mechanisms at both network and application layers. For instance, a video on-demand application in a household generates on an average more streaming content than a SME comprised of 2-3 employees. At the same time, the pandemic pushed 41% of connected consumers to stay home and shop online from their sofas, avoiding crowded stores and waiting in lines at malls [12].

Recently, network engineering mechanisms [15] have matured in cloud platforms [14] providing "network connectivity as a service". The latter facilitates a very rich ecosystem of networking solutions and services, such as Load-Balancer-as-a-Service (LBaaS), Firewall-as-a-Service (FWaaS) and VPN-as-a-Service (VPNaaS). The current shortcoming with respect to networking in such platforms includes standalone services and containers networking. The issue is that every new networking solution proposes and tries to reinvent how containers [5] will be interacting by adding more complexity or multiple encapsulation among services.

Cloud-native applications rely on characteristics such as horizontal / vertical scaling and elastic services offered by the underlying platforms. Interest in offloading application level networking functionality such as traffic and security management onto service meshes is increasing in the community and becoming a critical element of big data cloud infrastructures [1]. For instance, Istio [10] is an open source service mesh management platform that layers transparently onto existing distributed cloud applications. This service mesh management can be further enriched by Kiali [11] which provides observability through dashboards and enables to operate the mesh with robust configuration and validation capabilities.

To accommodate the fast expanding traffic and the demanding nature of streaming data analytics in multimodal web-mobile (i.e. connected consumers) or video on-demand applications, automated weighted load balancing mechanisms are required over the underlying resources to enhance the capacity of existing computer networks, cloud infrastructures or data centres serving computationally intensive tasks. To enable heterogeneous and converging big data clouds being dynamically adaptive to changes, research is needed to identify and quantify the performance metrics which impact these environments. At the same time, understanding containerized platforms and how their characteristics affect the application performance enables to predictably achieve cost-effective and resource-aware deployments.

This paper is concentrated on an automated process which enables to prioritize events by efficiently managing the communication among the interacting pods through adaptive routing and policies enforcement allowing to control the flow of traffic and API calls between microservices. This process guarantees at

runtime services stability, autoscaling and advanced performance. We demonstrate the proposed approach in a prototype microservices application consisting of containerized and deployed pods on a Kubernetes cluster, named as *Information Aware Networking Mechanisms* consisting of two logical units, i.e., a *Load Generator* and an *Autoscaler*. Istio has been deployed on the cluster to establish pod level communications, delegate traffic flows and filter requests. We dive into the characteristics and functionalities of the Information Aware Networking Mechanisms delivered as an open source deployment and experimental framework, through which cloud administrators and big data engineers may benefit from in implementing their own network engineering automations. The Information Aware Networking Mechanisms are used to conduct experiments related with the velocity of requests, the volume of requests and the response time. The benefit is that the Autoscaler results in efficiently handling more requests per second or more data volume per hour through runtime adaptations by scaling up more pods without affecting the response time of the application. Our contributions in this paper are as follows:

- A presentation of how network traffic is routed among Kubernetes pods with Istio service mesh and how network policies are enforced to realize traffic management and observability in order to achieve incremental *ML Model training and update* over big streaming data.
- An experimental exploration of how to correctly configure the Istio service mesh enabled with sidecar injection featuring pods telemetry monitoring and visualization through Kiali.
- A data experimentation using Apache JMeter [2] and the Prometheus [17] monitoring system in order to analyse the enforcement of network policies and prioritization schemes (i.e. response time, requests per second, data volume per hour) based on *Service-Level Objectives (SLOs)* defined as concrete metrics.
- A fully open source deployment and experimental framework released under the Apache License^{1,2,3,4}, enabling traffic prioritization through weighted load balancing, access control and rate limit across diverse protocols and runtimes which is designed for scalability and reproducibility.

2 RELATED WORK

Many circumstances necessarily force changes in modern applications, such as the fast increase in the use of cloud computing, the extensive use of multimedia streaming, the time consuming task of ML Model training and update over big data or the high-speed, high-throughput and low-latency of user application requirements.

Cloud environments are based on predefined service, network and security policies and are difficult to be configured dynamically. In principle, these are time-shared environments with high variance, and thus, performance testing requires repeated experiments, consideration of multiple objectives (i.e. cloud scalability, maximized usage of memory, reduced disk I/O, minimized data transfer over the network or parallel processing to fully leverage multi-processors) and statistical analysis [16]. The proposed approach takes advantage of inter-pods communication to dynamically configure network policies and data traffic to targeted pods constituting the end-to-end application during runtime

based on defined SLOs (i.e. response time, requests per second, data volume per hour).

Although there is a growing interest in and rapid adoption of containers, running them in production requires a steep learning curve, due to technological immaturity and lack of operational know-how [3]. Henning and Hasselbring [8] introduced a method for benchmarking the scalability of distributed stream processing engines. Within their method, they presented use cases where the stream processing microservices need to fulfill multiple constraints along with an in advance knowledge w.r.t. how the demand of resources is evolving with increasing workloads. Our approach is different because the defined SLOs are monitored throughout the application life cycle to drive runtime adaptations via pods control in the service mesh and prioritization schemes enforced by the network policies.

Eismann et. al. [6] presented the benefits and challenges of microservices from a performance tester's point of view. Through a series of experiments, they demonstrated how heterogeneous microservices affect the application performance. They also presented that it is not trivial to achieve reliable performance testing and that the process of rating cloud infrastructure offerings based on their achieved elastic scaling still remains uncharted. The latter served both as a challenge and motivation for us to elaborate over the novel concept of runtime adaptations, network policies and pods prioritization schemes.

Herbst [9] proposed a descriptive load profile modeling framework together with automated model extraction from recorded traces to enable reproducible workload generation with realistic load intensity variations for improved applications elasticity. The proposed mechanisms in this paper exploit load profiling to dynamically trigger adaptations and ML Model update at runtime.

The essential problem of dealing with big data is, in fact, a resource issue. Because the larger the volume of the data, the more the resources are required, in terms of memory, processors, and disks. The goal of performance optimization is to either reduce resource usage or make it more efficient to fully utilize the available resources, in order to take less time to read, write, or process the data.

Grohmann et. al. [7] use machine learning techniques to automatically recommend the best suitable approach for services demand estimation. Their approach works in an online fashion and incorporates new measurement data and changing characteristics on-the-fly. Lin et. al. [13] propose and answer two research questions regarding the prediction and optimization of performance and cost of serverless applications. They propose a new construct to formally define a serverless application workflow, and then implement analytical models to predict the average end-to-end response time and the cost of the workflow.

Compared to the above mentioned approaches, the proposed framework differs by introducing *Information Aware Networking Mechanisms* contributing in autoscaling and resulting in incremental ML Model training and update of a real-world connected consumer application at runtime. In an environment under high load, service demands normally cannot be directly measured, and therefore a number of estimation approaches exist based on high-level performance metrics. We show that service demands by means of velocity, volume and real-time constraints can be efficiently handled by weighted load balancing approaches which give us encouraging indications for further research on the application of service demand routing. The metrics that we take into consideration refer to the number of requests per second, the data volume per hour and the desired response time. The experiments over the proposed Information Aware Networking Mechanisms

¹<http://bigdatastack-tasks.ds.unipi.gr/ppetrouubi/istiouaml>

²<http://bigdatastack-tasks.ds.unipi.gr/ppetrouubi/istioproxy>

³<http://bigdatastack-tasks.ds.unipi.gr/ppetrouubi/istiopod>

⁴<http://bigdatastack-tasks.ds.unipi.gr/ppetrouubi/istiopodconsumer>

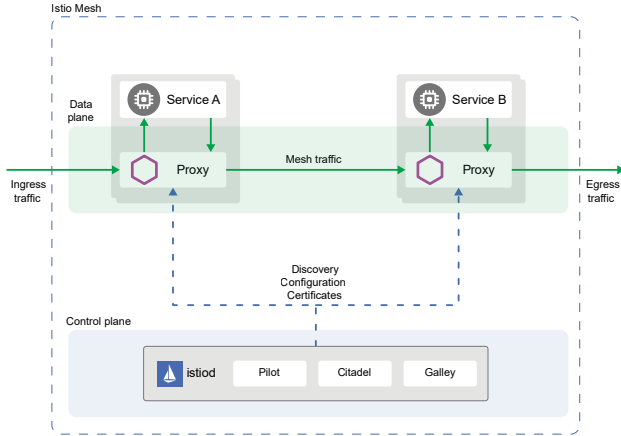
show that while Istio does cost in terms of adding some more resources, it is able to support a better overall performance and scalability for highly loaded environments in terms of successfully served requests, without requests rejection especially for data intensive operations in real-time.

3 INFORMATION AWARE NETWORKING MECHANISMS

3.1 Preliminaries

To facilitate the reader follow some basic concepts, we briefly present the core features of Istio. The Istio service mesh is divided in two logical units: a control plane and a data plane. The control plane manages and configures proxies to route traffic. It also configures components to enforce policies and collect telemetry data. In the data plane, Istio support is added to a service by deploying an intelligent *sidecar proxy*, called *Envoy*. This sidecar proxy routes requests to and from other proxies, mediates and controls all network communication between microservices by forming the mesh network. Figure 1 shows the different components that make up each plane, as well as the high-level architecture of Istio.

Figure 1: Istio Architecture [10].



3.2 Logical Units

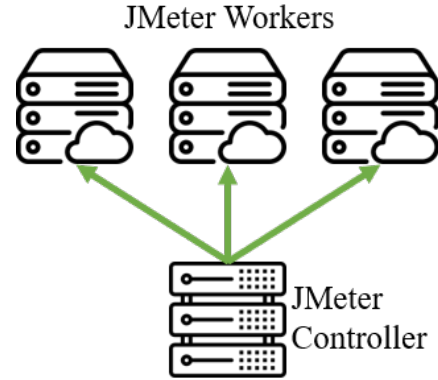
The core logical units constituting the Information Aware Networking Mechanism are:

- The *Load Generator* which issues requests and stresses the environment under experimentation; and
- The *Autoscaler*, which configures, controls and enforces the weighted load balancing policies to the pods serving the streaming big data application.

3.2.1 Load Generator. Data is harvested by two main sources. The Load Generator using JMeter generates variant workloads and simulates how the environment under test behaves from the perspective of an external observer. In addition to this, the Prometheus monitoring system deployed in the Kubernetes cluster reports on the internally observable behavior. The Load Generator has been deployed in a distributed mode as it is depicted in Figure 2. In this figure, JMeter initiates one controller node which launches tests on multiple worker nodes.

The configuration of the experiments was made by setting properties in a .jmx file through JMeter, which generates external traffic mimicking the behaviour of multiple HTTP clients. In this file, we configure the number of concurrent users accessing the application, the velocity / number of requests per second and the volume of requests measured in GB/hour. A sample .jmx file

Figure 2: JMeter in Distributed Mode [2].



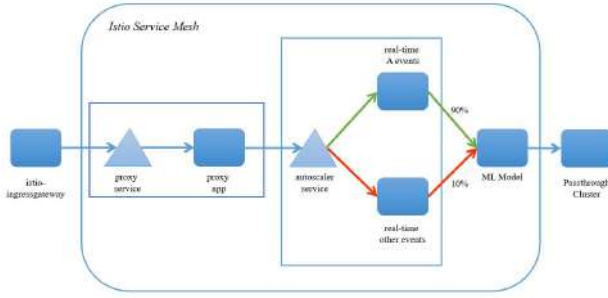
containing: a) important events (i.e. REC_RMV) which are getting higher priority by the Autoscaler because they are considered more crucial for the ML Model training and update; and b) events (i.e. VIZ_PROD) which are equally treated as they are arriving, is presented as follows. More technical details about the users (i.e. JMeter Workers) are presented in Section 4.

```
<events>
  <elementProp name="productId" elementType="HTTP">
    <strProp name="Arg.name">productId</stringProp>
    <strProp name="Arg.value">1</stringProp>
  </elementProp>
  <elementProp name="customerId" elementType="HTTP">
    <strProp name="Arg.name">customerId</stringProp>
    <strProp name="Arg.value">14</stringProp>
  </elementProp>
  <elementProp name="eventType" elementType="HTTP">
    <strProp name="Arg.name">eventType</stringProp>
    <strProp name="Arg.value">REC_RMV</stringProp>
  </elementProp>
</events>
<events>
  <elementProp name="productId" elementType="HTTP">
    <strProp name="Arg.name">productId</stringProp>
    <strProp name="Arg.value">2</stringProp>
  </elementProp>
  <elementProp name="customerId" elementType="HTTP">
    <strProp name="Arg.name">customerId</stringProp>
    <strProp name="Arg.value">12</stringProp>
  </elementProp>
  <elementProp name="eventType" elementType="HTTP">
    <strProp name="Arg.name">eventType</stringProp>
    <strProp name="Arg.value">VIZ_PROD</stringProp>
  </elementProp>
</events>
```

3.2.2 Autoscaler. With the convergence of all data operations and services in the same network mesh, the Autoscaler manages traffic by taking into consideration the network utilization, the pods requirements and the communication latency without compromising the efficiency of the serving application. Using policy statements, the end users can specify which kinds of pods they desire to give weighted load priority, at what times and on what part of their communication protocol (i.e. TCP, HTTP, etc.). By deploying and configuring the Istio service mesh [10] and the Autoscaler at the experimental testbed all the data metrics are collated by Prometheus Mixer [18] and stored in the Prometheus [17] monitoring system. Kiali [11] uses the data stored in Prometheus to show the service mesh topology, metrics, traffic information and more. The beneficiaries of the Autoscaler

can be cloud administrators and / or big data engineers with the ability to either configure their big data flows or the application requirements with specific networking primitives which fulfill the desired SLOs (i.e. bursty or voluminous pods demand and requests able to be efficiently served). The objective may refer to the efficient management of various kinds of traffic (i.e. streams, batches and micro batches) by getting the availability isolation and bandwidth priority that are needed to serve the end-to-end application. A common configuration as part of the architecture design which concretizes the data flow events and the respective logical interaction of the pods is depicted as follows.

Figure 3: Conceptual Architecture of Streaming Flows.



The deployed microservices / pods and their interaction are described in YAML files. In order to enable Istio service mesh for pods at the experimental testbed, we add "sidecar.istio.io/inject: true" in the YAML file. The *Proxy Service* based on Istio policies acts as a gateway which receives external traffic. Then, the *Autoscaler Service* through the Istio service mesh sidecar splits the requests / traffic based on the application requirements by filtering accordingly the type of events. In Figure 3, the important events (i.e. real-time A events) get more priority (i.e. 90%) and are assigned a greater bandwidth within the cluster compared to the events (i.e. real-time other events) related with more trivial user interactions, e.g. to visualize a product or equivalent. As the end-to-end application under experimentation serves a connected consumer cluster of pods, the real-time events of type A relate with transactions directly affecting the ML Model. Specifically, the real-time events of type A contribute in the online training of the ML Model which implements the Alternating Least Squares matrix factorization algorithm in the Apache Spark MLlib. This ML Model supports a recommender pod which posts personalized suggestions to the end users of the connected consumer application.

To address the current challenges, the SLOs and the respective network policies have been implemented and are supported by the Autoscaler. At the same time, to generalize the appropriate attributes in order to weight the traffic towards concrete microservices / pods, we give priority to events of interest according to their type. This operation implements the policy enforcement endpoint inside the pod as sidecar container in the same network namespace. This approach is highly flexible and facilitates to apply policies in the support of operational goals, such as service routing, prioritization schemes over data flows, retries and circuit-breaking.

The Autoscaler operates at the application layer by taking into consideration metrics from the Kubernetes cluster and specifically from the Prometheus monitoring system. The latter gives the advantage of being universal. Our focus is to address the challenges arising from the diverse data processing modes (i.e., stream, micro-batch, batch) to efficiently enforce policies to the

underlying pods (i.e. distributed object storage, ML Model training and update as well as other serving pods) without the restriction to only use HTTP. The workloads in the experimental testbed communicate without IP encapsulation or network address translation for improved bare metal performance, which enables easy troubleshooting and better interoperability.

Through Kiali, we visualize mesh network health between the interacting pods where the Autoscaler routes the weighted loads (i.e. events A vs. other events) to the ML Model. For instance, through the Kiali dashboard we can see in Figure 4 the request duration of the Proxy Service stressing the Autoscaler by means of average, median (i.e p50) and maximum (i.e. p95) quantiles.

Figure 4: Visualizing Networking Mechanisms via Kiali.



4 EXPERIMENTS

In this section, we describe our approach to experiment over the behaviour of the Information Aware Networking Mechanisms, consisting of two logical units which generate and prioritize big data flows based on their characteristics and requirements. Benchmarking with variant volumes and velocity of data is fundamental to assess the feasibility of the examined application. Our goal is to validate the efficiency of the proposed mechanisms based on the data volume of requests per hour, the velocity of requests per second and the response time. To this end, we have defined experiments with the following goals:

- Data generation with variant volumes and velocity;
- Experimental scalability and reproducibility;
- Data control and exploration analysis before and after the Autoscaler execution resulting in more efficient ML Model updates.

4.1 Configuration and Deployment

To evaluate the performance of the Information Aware Networking Mechanisms, we used a cluster of one master with 3 workers. Each node has a 4 core QEMU Virtual CPU at 2.4GHz, and 16 GB of RAM. The Load Generator is being executed through 3 distributed pods lying in each worker and several network, node-level and pod-level metrics such as latency, throughput, CPU & memory usage, disk I/O, etc. within the cluster are measured using the Prometheus monitoring system. The configuration and the deployment of the respective pods acting as enablers of the Information Aware Networking Mechanisms are described through the following YAML file.

```

kind: VirtualService
apiVersion: networking.istio.io/v1alpha3
metadata:
  name: autoscaler
  namespace: istioapp
spec:
  hosts:
    - autoscaler.istioapp.svc.cluster.local
  http:
    - match:

```

```

- uri:
  exact: /poll
route:
- destination:
  host: autoscaler.istioapp.svc.cluster.local
  subset: v1
  weight: 90
- destination:
  host: autoscaler.istioapp.svc.cluster.local
  subset: v2
  weight: 10
- fault:
  abort:
    httpStatus: 500
    percentage:
      value: 0
  match:
    - uri:
      exact: /feedbacks/event_A
  route:
    - destination:
      host: autoscaler.istioapp.svc.cluster.local
      subset: v1
      weight: 100
- fault:
  abort:
    httpStatus: 500
    percentage:
      value: 0
  match:
    - uri:
      exact: /feedbacks/event_B
    - uri:
      exact: /feedbacks/event_C
    - uri:
      exact: /feedbacks/event_D
  route:
    - destination:
      host: autoscaler.istioapp.svc.cluster.local
      subset: v2
      weight: 100

```

4.2 Performance Evaluation of Autoscaler

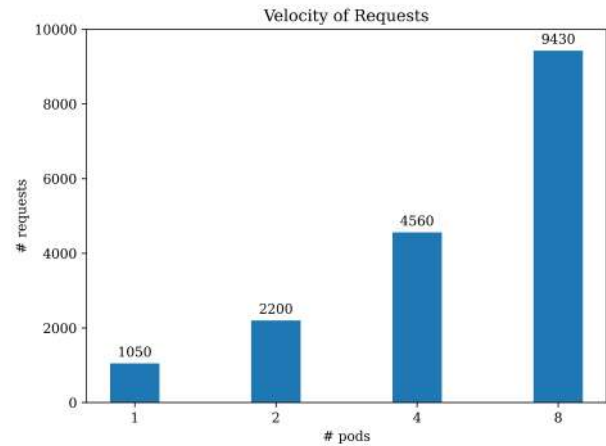
The Autoscaler plays the role of efficiently filtering, splitting and load balancing in a weighted manner the big data flows targeting at the ML Model. The Kiali system interacts with the Prometheus monitoring system to collect metrics w.r.t. the data volume of requests per hour, the velocity of requests per second and the response time. The experimental settings of the Autoscaler are broken into 3 phases, as follows:

- Configuration of variable data volumes and velocity of requests, which includes the initialization of the respective pods at the JMeter and Istio contexts. The Proxy Service receives data from the Load Generator based on defined metrics w.r.t. the velocity (i.e. the number of requests per second, 1K, 2.2K, 4.5K and 9.4K) and the data volume (i.e. 1.44GB/hour, 2.81GB/hour, 5.62GB/hour and 10.44GB/hour), while the Autoscaler splits the data flows in other serving pods. Then, the ML Model performs an HTTP GET request based on the defined prioritization scheme (e.g. real-time A events = 90% and real-time other events = 10%). The prioritization scheme can be adjusted according to the application constraints (i.e. SLOs), while in the examined case it achieves to automatically scale up the serving pods contributing in the ML Model update.

- The decomposition and routing of the respective requests is being held through the Istio service mesh. Kiali communicates with Prometheus and gets metrics (i.e. % weighted requests) about how the requests received by the Proxy Service and prioritized by the Autoscaler are routed to the ML Model pod. According to the metrics received by each ML Model pod, the Autoscaler dynamically scales up more (ML Model) pods before a pod's queue saturation reaches 95%.
- The visualization of the experimental results including the velocity of requests in total number per second, the volume of requests in GB/hour and the response time in seconds. These results validate that the weighted load balancing mechanisms have been correctly activated and that resources are efficiently scaled up to serve the application without rejecting new data / requests, as needed.

The experimental results in Figure 5 demonstrate that by increasing the velocity of the streaming data flows (i.e. 1K, 2.2K, 4.5K and 9.4K requests per second), the Autoscaler needs to merely scale up to 8 pods in order to efficiently serve and prioritize more requests to the ML Model without rejecting newcoming data. The Autoscaler itself is served by the cloud infrastructure without adding considerable overhead in the underlying computing resources, while it handles all the type of events and determines to add more resources when the velocity or the volume of important events increases. Then, the Autoscaler ensures that the real-time prioritized events of type A will get the required network bandwidth and computing resources. The real-time A events are considered important because they contribute in updating the ML Model which calculates the recommendations for the personalized suggestions delivered to the end users of the connected consumer application.

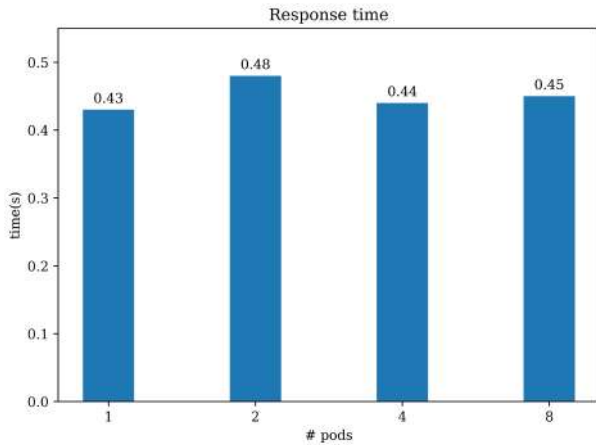
Figure 5: Velocity of Requests.



The experimental results in Figure 6 depict that the end-to-end response time between the Autoscaler and the ML Model remains constant although the number of requests has been increased by a factor of X1000 or more in each iteration without considerable resources cost or waste, i.e. by only scaling up few pods. The additional resources allocated to the Autoscaler permit to efficiently prioritize more requests to the ML Model in a parallel manner without discharging events, while maintaining the response time low and constant by satisfying the real-time SLOs and constraints of the streaming application.

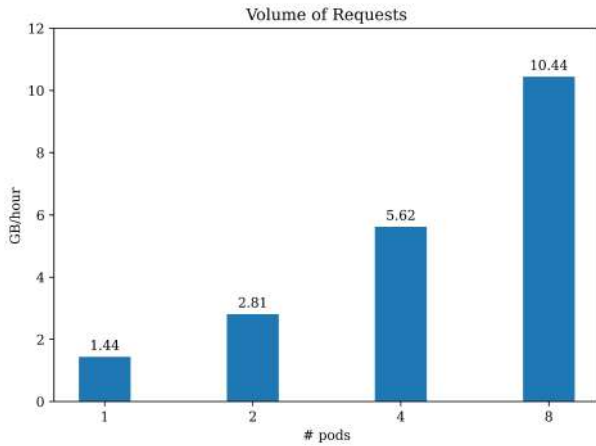
The experimental results in Figure 7 present that the Autoscaler can efficiently manage huge volumes of data (i.e. up

Figure 6: Response Time.



to tens of GB per hour) by scaling up more pods allowing more traffic to be served on-demand by the ML Model.

Figure 7: Volume of Requests.



5 CONCLUSIONS AND FUTURE WORK

The purpose of this work is to present a systematic analysis in the direction of Information Aware Networking Mechanisms which enable efficient traffic prioritization through weighted load balancing, access control and rate limit across diverse protocols and runtimes. The Information Aware Networking Mechanisms are offered as a fully open source deployment and experimental framework released under the Apache License which is designed for big data clouds in the support of massive data generation, scalability, reproducibility and data exploration. In order to analyse the enforcement of network policies and prioritization schemes (i.e. response time, velocity and volume of requests) based on Service-Level Objectives (SLOs), we used Apache JMeter and the Prometheus monitoring system. We also presented an experimental deployment of the Istio service mesh enabled with sidecar injection featuring pods traffic prioritization, telemetry monitoring and visualizations through Kiali. The Autoscaler takes into consideration several network, node-level and pod-level metrics such as latency, throughput, CPU & memory usage, disk I/O, etc. within the cluster in order to draw decisions, enforce policies and scale up more pods to facilitate the online training and update of the ML Model, and therefore the streaming connected consumer application.

In the near future, we plan to extend the Information Aware Networking Mechanisms in order to optimize the time the route policies need to take effect. As the modification of a policy takes some time (i.e. seconds) in order to be propagated to all the sidecars, we plan to intelligently enhance the networking mechanisms. Especially in large deployments, the respective policies need to be enforced well in advance an automatic process, like the Autoscaler, takes any decision or actuates the addition of more resources.

ACKNOWLEDGMENTS

The research leading to these results has received funding by the European Commission project H2020 BigDataStack “Holistic stack for big data applications and operations” (<https://bigdatastack.eu/>) under grant agreement No. 779747.

REFERENCES

- [1] Anne Thomas Andrew Lerner. 2018. *Innovation insight for service mesh. Market report*. Technical Report.
- [2] Apache JMeter 2020. *The Apache JMeter application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance*. Retrieved December 20, 2020 from <https://jmeter.apache.org/>
- [3] Arun Chandrasekaran. 2020. *Best Practices for Running Containers and Kubernetes in Production*. Retrieved December 20, 2020 from <https://www.gartner.com/en/documents/3902966/best-practices-for-running-containers-and-kubernetes-in-production>
- [4] CISCO 2020. *Cisco Annual Internet Report (2018–2023) White Paper*. Retrieved December 10, 2020 from <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>
- [5] Docker Container 2020. *What is a Container? A standardized unit of software*. Retrieved December 10, 2020 from <https://www.docker.com/resources/what-container>
- [6] Simon Eismann, Cor-Paul Bezemer, W. Shang, Dusan Okanovic, and A. V. Hoorn. 2020. *Microservices: A Performance Tester’s Dream or Nightmare? Proceedings of the ACM/SPEC International Conference on Performance Engineering (2020)*.
- [7] Johannes Grohmann, N. Herbst, Simon Spinner, and Samuel Kounev. 2018. *Using Machine Learning for Recommending Service Demand Estimation Approaches - Position Paper*. In *CLOSER*.
- [8] S. Henning and W. Hasselbring. 2020. *Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines*. *ArXiv abs/2009.00304* (2020).
- [9] N. Herbst. 2018. *Methods and Benchmarks for Auto-Scaling Mechanisms in Elastic Cloud Environments*.
- [10] Istio 2020. *Connect, secure, control, and observe services*. Retrieved December 10, 2020 from <https://istio.io/>
- [11] Kiali 2020. *Service mesh management for Istio*. Retrieved December 10, 2020 from <https://kiali.io/>
- [12] Lauren Thomas 2020. *Black Friday 2020 online shopping surges 22% to record \$9 billion, Adobe says*. Retrieved December 10, 2020 from <https://cutt.ly/ph0IMfI>
- [13] C. Lin and H. Khazaei. 2021. *Modeling and Optimization of Performance and Cost of Serverless Applications*. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2021), 615–632. <https://doi.org/10.1109/TPDS.2020.3028841>
- [14] OpenStack 2020. *Cloud Infrastructure for Virtual Machines, Bare Metal, and Containers*. Retrieved December 10, 2020 from <https://www.openstack.org/>
- [15] OpenStack Neutron 2020. *Neutron’s documentation*. Retrieved December 10, 2020 from <https://docs.openstack.org/neutron/pike>
- [16] Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim Von Kistowski, Ahmed Ali-Eldin, Cristina Abad, José Nelson Amaral, Petr Tma, and Alexandru Iosup. 2019. *Methodological principles for reproducible performance evaluation in cloud computing*. *IEEE Transactions on Software Engineering* (2019).
- [17] Prometheus 2020. *An open-source monitoring system with a dimensional data model, flexible query language, efficient time series database and modern alerting approach*. Retrieved December 20, 2020 from <https://prometheus.io/>
- [18] Prometheus Mixer 2020. *Prometheus mixer is a query and remote read API proxy that reads raw samples from multiple Prometheus remote read endpoints, mixes them into time-series normalized to a single sample per interval and runs a query on the resulting data*. Retrieved December 20, 2020 from <https://github.com/mz-techops/prometheus-mixer>