Global Journals LaTeX JournalKaleidoscope™
Artificial Intelligence formulated this projection for compatibility purposes from the original article published at Global Journals. However, this technology is currently in beta. *Therefore, kindly ignore odd layouts, missed formulae, text, tables, or figures.*

A Network Science-based Approach for an Optimal Microservice Governance

Gihan Saranga Siriwardhana[1], Nishitha De Silva[2], Liyanage Sanjaya Jayasinghe[3] and Lakshitha Vithanage[4]

[1] Sri Lanka Institute of Information Technology

Received: 1 October 2020 Accepted: 29 October 2020 Published: 9 November 2020

Abstract

With the introduction of microservice architecture for the development of software applications, a new breed of tools, platforms, and development technologies emerged that enabled developers and system administrators to monitor, orchestrate and deploy their containerized microservice applications more effectively and efficiently. Among these vast arrays of technologies, Kubernetes has become one such prominent technology widely popular due to its ability to deploy and orchestrate containerized microservices. Nevertheless, a common issue faced in such orchestration technologies is the employment of vast arrays of disjoint monitoring solutions that fail to portray a holistic perspective on the state of microservice deployments, which in turn, inhibit the creation of more optimized deployment policies. In response to this issue, this publication proposes the use of a network science-based approach to the creation of a microservice governance model that incorporates the use of dependency analysis, load prediction, centrality analysis, and resilience evaluation to effectively construct a more holistic perspective on a given microservice deployment. Furthermore, through analysis of the factors mentioned above, the research conducted, then proceeds to create an optimized deployment strategy for the deployment with the aid of a developed optimization algorithm. Analysis of results revealed the developed governance model aided through the utilization of the developed optimization algorithm proposed in this publication, proved to be quite effective in the generation of optimized microservice deployment policies.

Index terms— auto-scaling, chaos engineering, kubernetes, machine learning, microservices, NSGA-?, time series.

1 I. INTRODUCTION

he term ”microservices” was first introduced in 2011 [1] and was considered as a specialized implementation of Service-Oriented Architecture (SOA), coined to denote the common architectural approach of decomposing applications into smaller selfcontained, loosely coupled services. The microservice architectural style was later widely adopted in place of the traditional monolithic architecture by many leading companies such as Amazon, Netflix, LinkedIn, and SoundCloud due to the capability to develop loosely coupled services possessing the ability to be independently deployed, versioned, and scaled while ensuring in benefits such as faster delivery, more excellent performance, and greater autonomy [1].

The shift in architectural style from the traditional monolithic architecture to microservice architecture also brought forth the creation of a set of new methodologies and approaches that established the policies, standards, and best practices for the adoption of microservices, designed for the agile IT environment, known as

"Microservices Governance" [2]. This approach to governance was entirely dissimilar to the traditional governance policies followed in monolithic applications primarily since governance in microservices followed a decentralized approach, whereas governance in monoliths followed a centralized approach where decisions were made "top-down" [2]. Although the decentralized approach of governance of microservice provided advantages such as the freedom to develop applications utilizing diverse technology stacks, a downside of this approach was that more steps should be taken to ensure effective governance is maintained, since typical applications required interconnections between a vast number of microservices where business process workflows were continuously introduced. Consequently, organizations required the service of a variety of tools, ranging from monitoring and autoscaling to others such as configuration management, service discovery, and fault tolerance, that facilitated the multitude of tasks required to ensure effective microservice governance was in effect.

In addition to the tools mentioned above, new deployment strategies that facilitated the newly developing microservice infrastructure were introduced. Amongst them, containerization of microservices became one of the most effective ways to deploy microservice applications due to its ability to efficiently package microservices by encompassing all the required libraries and dependencies needed during runtime. This procedure separated the application from the underlying infrastructure and enabled developers to run the application in an isolated environment, ensuring performance and functionality. As a result, propelled by services such as Docker, containerization became the preferred approach for effectively deploying microservices, in contrast to the traditional virtualization-based approach previously adopted. However, in the case when the number of microservices of a particular application increased, it became increasingly difficult to coordinate, schedule, monitor, and maintain the required containerized microservices, especially in times where utmost application performance was required. In response to this issue, the Kubernetes framework was introduced in 2014 [3] to allow organizations to run distributed systems more resiliently by providing effective solutions for load balancing, storage, orchestration, automated rollouts, and self-healing mechanisms [4]. The unique characteristics offered by Kubernetes in this regard, thereby transformed it into one of the most prominent microservice-based technologies available for organizations to deploy their vast arrays of microservice applications in productiongrade environments.

The introduction of Kubernetes ushered in a new era of microservice governance through the introduction of container orchestration. Nevertheless, as evident throughout this publication, despite its immense use in orchestrating microservice applications, Kubernetes is still not able to provide a perfect governance solution to most modern microservice applications, as there are still prevalent issues that need to be addressed in Kubernetes particularly concerning the policies followed in the deployment of interdependent microservices.

A primary reason for the existence of inefficient optimization policies in Kubernetes based microservice deployments is the lack of the tools and services to obtain a holistic view of Kubernetes deployments and thereby optimize cluster performance. The current tools and services offered by Kubernetes often have to be preconfigured to the existing pre-conceived knowledge of the developers in contrast to the actual real-time utilization. Although implementing such solutions may be of use in the short term, it maybe it may be difficult to further improve upon the performance of the microservice cluster in the long term due to the lack of a holistic perspective on the interaction of the interdependent microservices in realtime use. Hence, it should be realized that if a particular microservice deployment is to be optimized for performance, a clear understanding regarding the relationships among the interdependent microservices during runtime is required. However, if a microservice deployment is to be truly optimized for optimal performance, it may also be necessary to take into account factors such as the resilience among the interdependent microservices, the effect of autoscaling policies, in addition to a clear understanding on the interactions of interdependent microservices. Regardless, even though there are several monitoring solutions available for such purposes, such as Prometheus, Istio, and Chaos Toolkit, their disjoint nature prevents them from allowing users to obtain a holistic perspective on the state of their deployed microservices. Furthermore, in cases such as fault management, error handling, and performance monitoring, due to the disjoint nature of these monitoring solutions, users are often unable to gain insight into possible solutions as to why a particular problem or bottleneck has occurred even though they are often made aware of the presence of a particular problem by these monitoring solutions.

In addition to the above-mentioned issues, these monitoring solutions are also often and plagued with other challenges such as the difficultly in successfully configuring and integrating these monitoring tools with the existing tools used by organizations [5]. The issues mentioned above may also further complicate the already complicated management and configuration process prevalent in Kubernetes and, in turn, may confuse inexperienced developers and system administrators, ultimately leading towards misallocation of cluster resources and degradation of cluster performance.

In response to the issues stated above, this publication proposes a novel approach to the creation of a unified governance model that can be used by developers and system administrators to effectively oversee the performance of their microservice deployments factoring in dependency analysis, load prediction, centrality analysis, and residency evaluation in order to determine the optimal placement of microservices and thereby create an optimized deployment plan for a given microservice deployment. Thus, through the application of the proposed governance model, users would be able to obtain a more holistic view of their deployment, resulting in a greater understanding of the runtime behavior of the deployed microservices, thereby enabling greater optimization possibilities. Through application of the approach proposed in this publication, the authors wish

to provide key insight to the contribution of a new set of microservice deployment optimization methodologies, which factor in the impact of key factors such as dependency among deployed microservices, autoscaling policies as well as resilience measures in microservice deployments.

The governance model proposed in this publication is comprised of four main components, each aimed at capturing a particular dimension of the microservice deployment with the ultimate goal of achieving a more holistic view of a given microservice deployment. Accordingly, the key components of the proposed model are as follows.

1. A generated microservice co-dependency map which is aimed at obtaining a clear perspective about the dependencies between each microservice and the importance of the deployment plan.

2 A load prediction and centrality analysis component

for the prediction of the level of interdependency among co-dependent microservices, the resource utilization of pods in the cluster as well as performing the task of the calculation of centrality measures of microservices in the co-dependency network. 3. A resilience evaluation component to evaluate the resilience of microservices in the cluster.

4. An optimal placement algorithm to determine the optimum placement of microservices in the Kubernetes cluster based on the above-stated measures.

The remainder of this publication is organized as follows. Section ? discusses the background and the related work literature referenced in the development of this optimization model. Section ? discusses the methodology followed in the development of the proposed model along with an overview of its key components. Section ? discusses the results obtained through the application of the developed model and, finally, the conclusion of this publication, along with directions for future work, is outlined in Section ?.

3 II. BACKGROUND AND LITERATURE

The apparent need for improved microservice governance modeling strategies, along with some of the prevalent issues in current microservice governance methodologies, have been highlighted in several publications throughout the years. The authors of [6] highlight the need for new modeling strategies that capture the recent advances in deployment technology such as Kubernetes. The publication [7] states the inability of monitoring frameworks to measure microservice performance level metrics would lead to the creation of several new research topics, which include the development of holistic techniques for collecting and integrating monitoring data from microservices and datacenter resources. In contrast, publications such as [1] highlight the use of past actions and events to better inform resource management decisions in microservice environments along with the challenges such as the overloading of monitoring events faced in resource monitoring and management processes.

In addition, several publications have also proposed performance modeling strategies for Kubernetes deployments. In this regard, [8] proposes an architectural approach that federates Kubernetes clusters using a TOSCA-based cloud orchestration tool. In contrast, research publications such as [9] proposed a tool named Terminus to solve the problem of finding the best-suited resources for the microservice to be deployed so that the whole application achieves the best performance while minimizing the resource consumption. Other researches include the reference net-based model for pod & container lifecycle in Kubernetes proposed by the authors of [10] and the generative platform for benchmarking performance and resilience engineering approaches in microservice architectures as proposed in [11].

The approaches suggested in the publications stated above are all approaches that aim at performance optimization of Kubernetes deployments. However, a key aspect to note in this regard is the fact that the methodologies stated in the publications mentioned above, fail to capture critical dimensions such as the dependent relationships between microservices, the effect of autoscaling policies, as well as resilience measures in the determination of the optimal placement of a particular microservice with regard to its global significance. Therefore, to our knowledge, there is no current solution proposed, that takes into consideration an integrated modeling strategy, factoring key elements essential to the optimization of microservice deployments such as codependencies present as well resilience and centrality measures among microservices when developing a holistic governance policy for Kubernetes based microservice deployments, as proposed in this research.

4 III. METHODOLOGY

The proposed governance model consists of four principal components each interlinked as depicted in Fig. 1 below. The following sub-sections provide an in-depth analysis regarding the methodology followed in the development of the proposed governance model along with an overview of the respective functionalities of its components. The microservice co-dependency network consists of three sub-components which can be listed as follows.

1. An Istio service mesh platform that incorporates Kiali and Prometheus monitoring solutions. 2. A backend NodeJS "K8Advisor" server for integration with metric APIs provided by monitoring solutions. 3. A database solution for the storage of gathered metric data.

The Istio service mesh provides the core metric servers such as Kiali and Prometheus, configured to retrieve data from the app, pod, and node levels in the cluster. In this regard, the microservice dependency map utilizes

quantified measurements derived from request and response times obtained primarily from the Kiali metric server to facilitate the development of the microservice co-dependency map.

The K8Advisor server aggregates all APIs exposed from the Istio service mesh and exposes a single endpoint such that required metrics could be queried more effectively. The server is configured to query metrics and trigger required processes as per a configured scheduler. The metrics collected in this regard, are then stored in the No-SQL database along with additional information such as timestamps to facilitate the creation of time series datasets utilized in the training of machine learning models. The K8Advisor server is also capable of generating CSV (Comma Separated Values) files on demand by reading the No-SQL database. The server will also expose an endpoint that can be accessed via an HTTP request in order to trigger required functions on demand. All the data stored in the database is maintained within the same Kubernetes cluster without exposing it to the public in order to maintain the privacy of user data. Lastly, in addition to the above, the K8Advisor server is also responsible for the creation of a node latency map through the evaluation of latency measures between the nodes in the cluster. Here, the Round-Trip Time (RTT) of network calls between nodes in the cluster is evaluated and, through the use of a developed shell script, the average latency measures between cluster nodes are obtained and forwarded to the optimization algorithm.

5 b) Load prediction and Centrality Analysis

The key objective of the load prediction component and centrality analysis component is the utilization of historical data and centrality measures to aid in the optimization of microservice deployments and the creation of holistic autoscaling policies. In this regard, the component performs the following key tasks.

? Prediction of future resource utilization values (primarily CPU and memory) based on historical pod resource utilization data. ? Prediction of inter-microservice link weight (dependency measures), based on historical link weight data derived from the load-based metrics in the co-dependency network.

? Calculation of centrality measures of microservices in the co-dependency network. The resource utilization prediction process is performed through performing a time series-based prediction on pod utilization metrics, in which predicted CPU and memory utilization values for a particular period are forecasted. The prediction process for resource utilization is performed through the application of a Long Short-Term Memory (LSTM) network in which a particular number of time steps of utilization metrics are used to predict future utilization values. Once predictions are made, the predicted utilization values for a particular period (e.g. -24 hours in advance) are passed through to the optimization algorithm to infer optimal autoscaling decisions.

The process of inter-microservice link weight prediction is primarily a network-based time-series prediction process in which the inter microservice link weights derived through load-based metrics are forecasted such that the next predicted weights for the links in the co-dependency map are determined. The forecasted weights determined through the use of an LSTM prediction model could then be used to provide an accurate estimation of the load that is expected to be received by microservices in the cluster, enabling the identification of key potential microservices which may in turn, highly manipulate microservice placement decisions and the realization of optimal cluster performance The calculation of microservice centrality measures is also performed within the load prediction component. Here, the microservices in the codependency network are evaluated on several centrality measures to facilitate the identification of influential microservices in the cluster. These calculated centrality measures are then forwarded to the optimization algorithm as inputs, to infer autoscaling decision through determination of required service instance levels. In this regard, the proposed governance model is expected to make use of the key centrality measures such as degree, betweenness, closeness as well as eigenvector centrality measures to facilitate the identification process of influential microservices.

6 c) Resilience Evaluation

The resilience evaluation component is particularly based on chaos engineering principles and utilizes the dependency measures derived from the codependency network to effectively target the most prominent services in the cluster for the evaluation of resilience measures. This process is performed through the use of Chaos Toolkit and the resulting resilience measures thus obtained, are then utilized to derive a holistic perspective on the resilience and health of interdependent services in the cluster. The optimization algorithm utilized in the proposed governance model is predominantly based on the NSGA-? (Non-Dominant Sorting Genetic Algorithm) algorithm. The algorithm generates a multitude of optimized solutions that enables the user to infer optimization decisions predicated on three key optimization categories, which are as follows.

7 d) Optimization Algorithm

? Solutions optimized for best performance and availability, thereby maintaining a balance between reduced latency and number of instances. ? Solutions optimized for optimal performance based on the reduction of latency. ? Solutions optimized for highest availability based on the maximization of the number of instances.

These optimized solutions are generated following four main input parameters utilized by the optimization algorithm as depicted in Fig. 2 above and can be listed as follows. 1. Predicted microservice dependency measures from the load prediction and centrality analysis component. 2. Node latency map generated from the Node Server. 3. Required number of microservice instances derived from centrality measures and predicted

resource utilization metrics from the load prediction and centrality analysis component. 4. General cluster infrastructure information gathered from monitoring solutions.

The sub-sections below provide an in-depth insight into the manner these input parameters are utilized in the developed algorithm as well as their impact on the creation of holistic optimization policies.

8 i. Predicted Microservice Dependency Measures

In microservice deployments, although factors such as latency cannot be completely eliminated, dependent microservices can be deployed in nearby nodes or the same node in order to reduce the overall latency of an application. Therefore, making use of this approach while intending to solve low availability and suboptimal performance issues, as well as to aid in the creation of autoscaling policies, the developed optimization algorithm makes use of the predicted loadbased link weights obtained from the load prediction component. This is done such that optimal placement and scaling decisions could be performed ahead of time, establishing a future deployment strategy such that users such as DevOps engineers would be able to make use of the gathered information to create an optimized microservice deployment plan. In addition, making use of the predicted dependency measures (load-based link weight), optimal placement decisions are determined through the application of (1) and (**??**), as defined below, which calculates the average latency among the microservice instances, based on the dependency measures and as the node latency map obtained from the Node sever. D j = ? L i i=m j=1 n**(1)**

Minimize TL= ? W j × D j j=m j=1

(2)

ii.

9 Node Latency Measures

The main objective of the optimization algorithm is the maximization of performance through the minimization of latency among microservices. Therefore, the developed optimization algorithm also utilizes a developed node latency map obtained from the Node Server, to evaluate the fitness of generated solutions.

iii. Required Microservice Instances In the process of fitness calculation, the first step is the calculation of the required number of instances per microservices. Here, the calculation of the required number of microservices instances is performed by utilizing the predicted resource utilization values derived from the load prediction component, applied on the Horizontal Pod Autoscaling algorithm. Also, the centrality measures derived from the co-dependency network will be utilized to infer the optimum microservice instance levels, particularly in cases where historical information of the cluster is unknown. The required microservice instance levels are also utilized in availability fitness calculation measures, aided through the use of a generalized logistic function [12] to avoid giving high scoring fitness values from resources that require low resource consumption and are of low instance levels, thereby establishing a fairer scoring method. In this regard, the fitness is calculated as defined through (3) given below. Fitness Cucalation Table **??**:M N Maximize TA= ? R i ×generalizedLogisticFunction? S i R i ? i=n i=1 (3)

The fitness function also makes use of a scoring system based on the distribution of the number of instances deployed on cluster node resources known as the scale value. In this regard, a higher number of instances distributed among cluster nodes throughout the deployment are given a higher score than localized instances deployed within a single node. This task is performed to avoid convergence of dependent services into one node and affecting availability. These scale values are then utilized to infer performance and availability decisions.

10 iv. General Cluster Information

The optimization algorithm also makes use of the general cluster infrastructure information such as the resource power consumption of nodes and node labels names. The information gathered in this regard is primarily utilized in the definition of constraints utilized by the optimization algorithm.

11 IV. RESULTS AND DISCUSSION

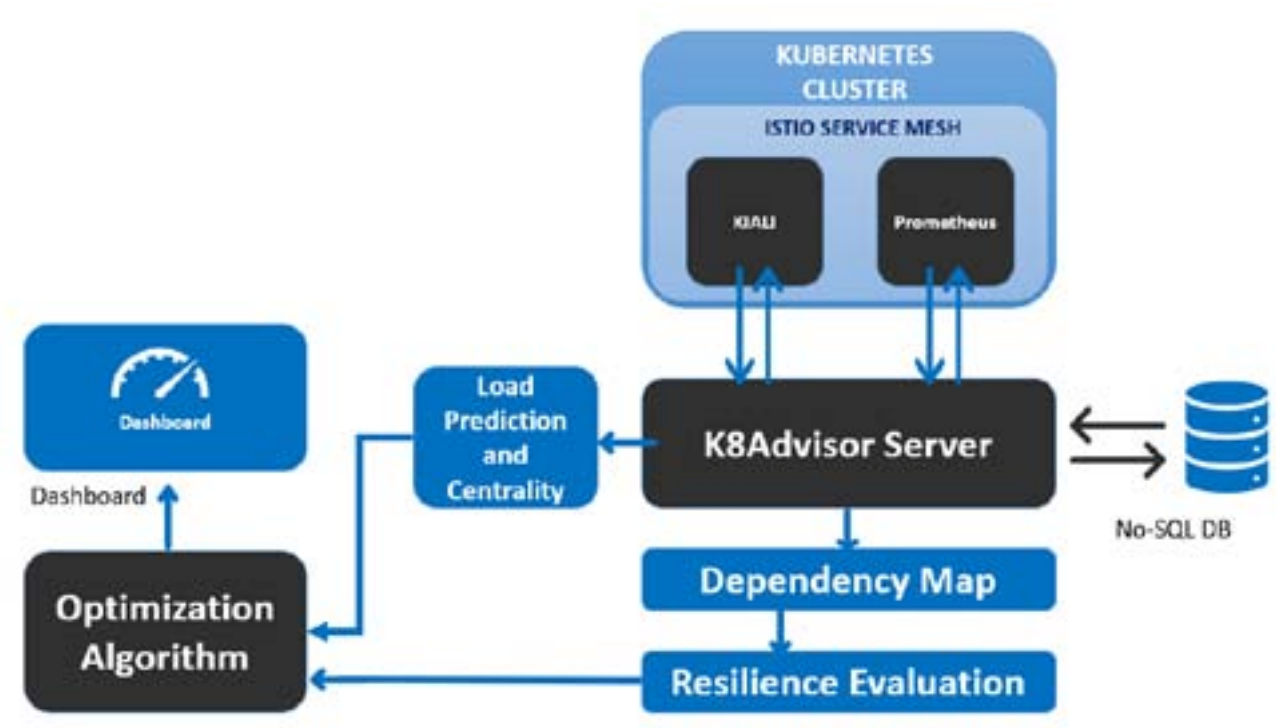
The developed optimization model was evaluated on a sample microservice cluster dataset containing 3 nodes and 6 microservices. For evaluation purposes, the JSON (JavaScript Object Notation) representation of this cluster dataset, along with the additional information required by the optimization algorithm which includes the node latency map, predicted inter-microservice dependency measures as well as the required number of microservice instances, is provided to the developed optimization algorithm in order to compute the optimized solutions. Fig. 3 below depicts the structure of the sample input JSON provided to the optimization algorithm.

Once the optimization algorithm is executed, a set of optimized solutions are obtained. In this regard, two optimized solutions are obtained once the algorithm is executed; one solution represents the cluster orientation with the highest cluster performance as depicted in Fig. **??**, whereas the second solution obtained depicts the solution that represents the cluster orientation with the highest cluster availability as depicted in Fig. **??**. For added clarification, the tabular format of the representation is given alongside the resulting solutions. Note the fact that in the tabular format depicted in Fig. **??** and Fig. **??**, each cell in the table represents the optimal number of instances of a given microservice that should be present in order to achieve the required optimization goal (highest performance or highest availability).

With regard to the resulting solution obtained that represents the cluster orientation with the highest performance, the fact that the optimization algorithm has successfully managed to determine the cluster orientation with the highest performance is evident primarily due to the fact that the highest dependent services as provided in the input JSON have been determined to be placed on the same node by the optimization algorithm. This fact is determined through comparing the keys of the key-value pair sets in the "pod_dependency_map" feature of the input JSON which represents inter-dependent sets of microservices with the tabular representation of the resulting optimal performance solution, that also depicts the inter-dependent microservices as described in the input JSON (such as M0 and M2) placed on the same node. (For example: -"[0, 2]: 1000" in the input JSON represents microservice M0 and microservice M2 are interdependent microservices with a dependency level of 1000) Similarly, through comparing the "microservices_instances_requirement" feature of input JSON which represents the required number of instances required for each of the six microservices respectively, with the resulting instance levels obtained from resulting highest availability solution, it is evident that the optimization algorithm has also ensured highest availability of microservices through the allocation of a higher number of microservice instances than the required instances. (For example-Microservice M0 requires the presence 4 instances and the optimization algorithm has allocated 8 instances of M0 as determined through its optimization process)

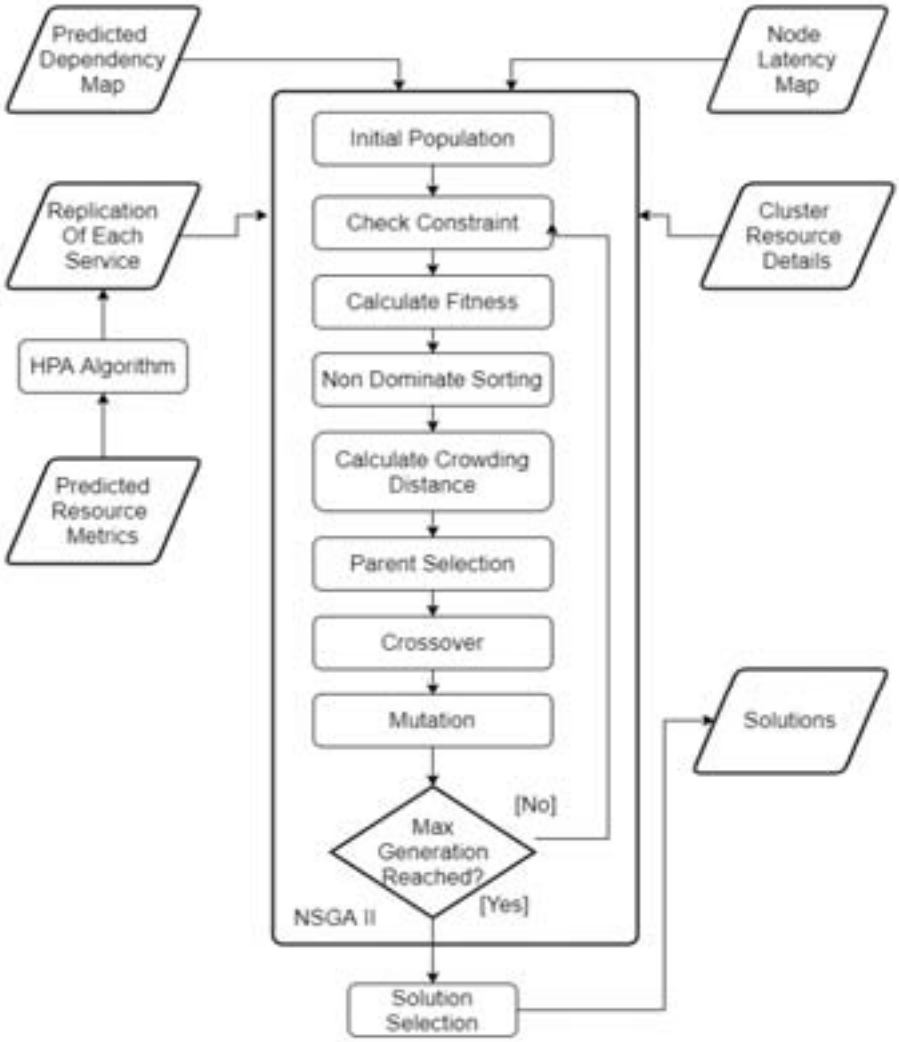
V. CONCLUSION This publication suggests the application of a network-science based microservice governance model in an attempt to aid in the creation of optimized microservice deployment policies currently hindered due to the employment of disjoint monitoring solutions prevalent in microservice-based governance methodologies. In this regard, the proposed model seeks the creation of a holistic perspective of microservice deployments, through the incorporation of dependency analysis, load prediction measures, centrality measures as well as resilience measures. Furthermore, through the incorporation of the above measures, the research conducted utilizes the application of an optimization algorithm to determine an optimal deployment strategy for a given microservice deployment.

The publication also discusses the core architecture along with the methodologies followed in the development of the proposed governance model as well as the results obtained through the application of the proposed governance model. Analysis of the results suggests the developed governance model proved to be effective in determining the optimized cluster representations pertaining to the highest performance and availability. Future work will include considering the inner workings of applications deployed in a Kubernetes cluster so as to further increase the accuracy of existing prediction models and resilience analysis components.



1

Figure 1: Fig 1 :



2

Figure 2: Fig 2 :A

{"nodes": {"0": {"ip": "", "cpu": 4000, "memory": 8000, "storage": 500000}, "1": {"ip": "", "cpu": 2000, "memory": 4000, "storage": 500000}, "2": {"ip": "", "cpu": 4000, "memory": 8000, "storage": 500000}}, "pods": {"0": {"name": "", "cpu": 300, "memory": 200, "storage": 300}, "1": {"name": "", "cpu": 300, "memory": 200, "storage": 300}, "2": {"name": "", "cpu": 200, "memory": 200, "storage": 300}, "3": {"name": "", "cpu": 200, "memory": 200, "storage": 300}, "4": {"name": "", "cpu": 200, "memory": 200, "storage": 300}, "5": {"name": "", "cpu": 200, "memory": 200, "storage": 300}}, "nod_latencies": {"[0, 1]": 20, "[0, 2]": 30, "[1, 2]": 20, "[0, 0]": 1, "[1, 1]": 1, "[2, 2]": 1}, "pod_dependency_map": {"[0, 2]": 1000, "[1, 3]": 500, "[4, 5]": 200}, "pod_importance": "", "microservice_instances_requirement": [4, 4, 2, 2, 2, 2]}

Figure 3: A

chromosome	
6	
0	
0	
0	
0	
5	
4	
0	
0	
0	
0	
2	
0	
4	
0	
0	
6	
0	
fitness_availability	1.2092686077
fitness_performance	1
fitness_scale	0.3611111111
generation	191
index	267
rank	1

Micro-services	Nodes		
	Node 0	*Node 1*	*Node 2*
M0	6	0	0
M1	0	0	5
M2	4	0	0
M3	0	0	2
M4	0	4	0
M5	0	6	0

45

Figure 4: Fig. 4 :Fig. 5 :

chromosome	
1	
2	
5	
4	
1	
2	
3	
0	
0	
0	
0	
5	
6	
1	
1	
6	
0	
1	
fitness_availability	1.5562381799
fitness_performance	0.0514054793
fitness_scale	0.7222222222
generation	188
index	174
rank	1

Micro-services	Nodes		
	Node 0	*Node 1*	*Node 2*
M0	1	2	5
M1	4	1	2
M2	3	0	0
M3	0	0	5
M4	6	1	1
M5	6	0	1

Figure 5: A

1

Number of dependencies in pod-level
Number of dependency links in app-level
W Dependency request weight in app-level
L The latency of dependency in pod-level
D Dependency average latency in app-level
TL Total latency

Figure 6: Table 1 :

["kubernetes. -Wikipedia (2020)] , "kubernetes. -Wikipedia . https://en.Wiki-pedia.org/wiki/Kubernetes. accessed Jun. 16, 2020.

[ACM/SPEC International Conference on Performance Engineering ()] doi: 10.11 45/3053600.3053627. *ACM/SPEC International Conference on Performance Engineering*, 2017. p. .

[Generalised logistic function -Wikipedia (2020)] *Generalised logistic function -Wikipedia*, https://en.wikipedia.org/wiki/Generalised_logistic_function Jul. 14, 2020.

[Kubernetes: The Challenge of Deploying Maintaining (2020)] *Kubernetes: The Challenge of Deploying & Maintaining*, https://techolution.com/kubernetes-challenges/ accessed Jun. 16, 2020.

[Microservices Governance: A Detailed Guide (2020)] *Microservices Governance: A Detailed Guide*, https://www.leanix.net/en/blog/microservices-gover-nance accessed Jun. 16, 2020.

[Jamshidi et al. ()] 'Microservices: The journey so far and challenges ahead'. P Jamshidi , C Pahl , N C Mendonca , J Lewis , S Tilkov . 10.1109/MS.2018.2141039. *IEEE Software* 2018. 35 (3) p. .

[Düllmann and Van Hoorn] *Model-driven generation of microservice architectures for benchmarking performance & resilience engineering approaches*, T F Düllmann , A Van Hoorn . p. 2017.

[Medel et al. ()] 'Modelling performance & resource management in Kubernetes'. V Medel , O Rana , J Á Bañares , U Arronategui . doi: 10.11 45/2996890.3007869. *Proceedings -9th IEEE/ACM International Conference on Utility and Cloud Computing*, (-9th IEEE/ACM International Conference on Utility and Cloud ComputingUCC) 2016. 2016. p. .

[Fazio et al. ()] 'Open Issues in Scheduling Microservices in the Cloud'. M Fazio , A Celesti , R Ranjan , C Liu , L Chen , M Villari . 10.1109/MCC.2016.112. *IEEE Cloud Computing* 2016. 3 (5) p. .

[Heinrich ()] 'Performance engineering for microservices: Research challenges & directions'. R Heinrich . 10.1145/305. *ACM/SPEC International Conference on Performance Engineering*, 2017. p. .

[Jindal et al. ()] 'Performance modeling for cloud microservice applications'. A Jindal , V Podolskiy , M Gerndt . 10.1145/3297663.3310309. *ICPE 2019 -Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019. p. .

[Kim et al. ()] 'TOSCA-based and federation-aware cloud orchestration for Kubernetes container platform'. D Kim , H Muhammad , E Kim , S Helal , C Lee . 10.3390/app9010191. *Applied Sciences (Switzerland)* 2019. 9 (1) .

[What is Kubernetes? | Kubernetes (2020)] *What is Kubernetes? | Kubernetes*, https://kubernetes.io/docs/concepts/overview/what-is-kuber-netes/ accessed Jun. 16, 2020.