National College of Ireland

Container Scheduling Using TOPSIS Algorithm

MSc Research Project
Cloud Computing

Anurag Pravin Shriniwar
Student ID: x18198171

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahani

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Anurag Pravin Shriniwar
Student ID:	x18198171
Programme:	Cloud Computing
Year:	2020
Module:	MSc Research Project
Supervisor:	Mr. Vikas Sahani
Submission Due Date:	17/08/2020
Project Title:	Container Scheduling Using TOPSIS Algorithm
Word Count:	6035
Page Count:	20

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 speci ed in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	15th August 2020

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 su cient to keep a copy on computer.	☐

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

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

Container Scheduling Using TOPSIS Algorithm

Anurag Pravin Shriniwar
x18198171

Abstract

Containerisation of the application is on the rise and need for orchestration tools is increasing along with it. Docker is de facto standard for containerisation but orchestration tool is in developing stage with tools like Docker Swarm, SwarmKit and Kubernetes in the market. Orchestration has many roles and responsibilities but scheduling is one of the important tasks which shows the scope of improvement. Orchestration tools are known to have problems in scheduling in a heterogeneous environment. This paper implements a multi-criterion decision making algorithm TOPSIS for container scheduling with Kubernetes. The implementation shows the accuracy of node selection on par with Kubernetes scheduler along with the feasibility of allocating user-de ned importance to system parameters.

1 Introduction

Containerisation of application gives developers unique advantages than the traditional architecture where developers have the exibility of deployment, ease of setting up CI/CD pipelines and added security that containerisation provides. This feature makes this technology appealing. Use of container is common in the new development spectrum and di erent products like LXC, Solaris containers, OpenViz have provided ways to containerise the application. Although this area is packed with new products and features, Docker is at the centre of attention. Docker is popular due to its ease of use, availability of stable popular images on docker hub platform and support available for the platform by both company and online by open-source platforms. All these features make it easy to adopt at large scale as well as small scale enterprise.

Managing these containers on a large scale includes a wide variety of tasks. Orchestration tools are available for managing these container clusters. Kubernetes, Docker Swarm, SwarmKit are orchestration tools that are popular currently. Swarm and SwarmKit are introduced by docker itself and Kubernetes is developed by Google. All these tools are open source. These tools perform the following task. Orchestration of containers was at the peak according to the hype cycle of cloud computing by Gartner (2018)

1. Provisioning and deployment of pods and containers.
2. Migration of Containers to optimise node resources.
3. Monitoring and health check up of nodes as well as containers.
4. Scale up and down of cluster con guration.

Orchestration tools schedule the container by using scheduling strategies. These strategies are known to be non optimised and there is scope for improvement in the scheduling area of containers. These tools provide 3 scheduling strategies which are Spread, Binpack and random. These strategies do not emphasise on the optimisation of the overall resources. Spread strategy as the name suggests tries to spread the containers across the available nodes as much as possible whereas Binpack tries to pack the containers in the least possible number of nodes. As an adaptation of containers and orchestration tools is increasing, enterprises nd di culty in using this strategy at a large scale.

This project implements TOPSIS algorithm for scheduling strategy where it uses following resource information from each node

1. Number of CPU cores available.
2. Percentage of CPU utilised.
3. Available virtual Memory.
4. Disk usage of node.
5. CPU Frequency.

User-de ned weight is assigned to each resource type which will be used in the calculation of the best node. TOPSIS algorithm processes this information and gives Best node among the available nodes. This node is used to schedule the containers. Implementation of the algorithm is done using python language. Python library of Kubernetes is used to call Kubernetes function to schedule pending containers from python code.

Further paper is divided into Literature review, Methodologies, Design, Implementation, Evaluation and conclusion along with future work related to the project.

2 Related Work

Existing strategies provided by current orchestration tools like Kubernetes, Docker swarm and SwarmKit work well in speci c conditions and shows mistakes in scheduling or poor resource utilisation which leads to poor application performance due to lack of resources one some nodes. This section reviews the current scheduling strategies along with a review of work done by independent researchers in this eld.

2.1 Overview of Orchestration Tools

Docker Swarm provides 3 scheduling strategies for container scheduling Spread, Binpack and Random. Working of these scheduling strategies is simple as the name suggests where Spread strategy tries to spread the total containers in maximum possible nodes whereas Binpack strategy tries to consolidate the containers in the least number of nodes. As the name suggests random strategy follows random allocation. Heap sort algorithm is used for nding out the least or highest density of container nodes and according to selected strategy container is placed on the speci c node. Li et al. (2019)

Kubernetes, Docker Swarm and SwarmKit are 3 major container orchestration tools available now. All of these tools are written in the GO language. GO is developed and

maintained by Google. Although the language is similar to C it performs well for memory management. Docker Swarm and SwarmKit are developed by docker itself and show better integration and ease of use for docker containers. Kubernetes is released by Google. It can be used for the majority of Linux containers available in the industry including docker container. This project uses Kubernetes for orchestration due to its edge over other orchestration tools regarding the integration of custom scheduling algorithm. No additional support is provided by Docker Swarm or SwarmKit for custom scheduling algorithm. Kubernetes released a new feature of supporting a custom scheduling algorithm with the latest Kubernetes release. A python library allows us to call Kubernetes APIs and schedule containers with python code.[1][2]

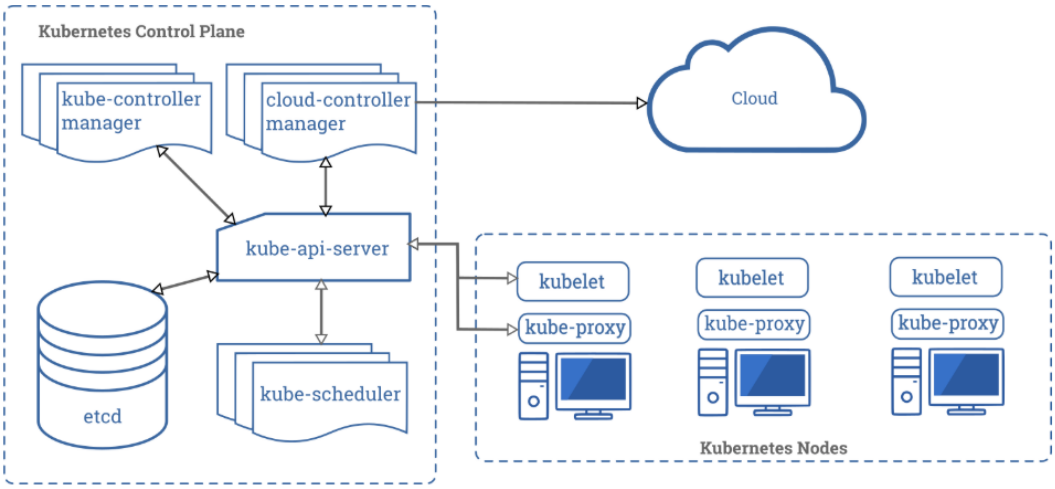

Figure 1: Process ow of implementation *Kubernetes Components* (n.d.)

Kubernetes architecture diagram is given with gure 1. Architecture is divided into two major parts Kubernetes control pane and Kubernetes node pane. Control pane is made up of API-Server from which all the Kubernetes APIs are exposed, ETCD where all the key-value data is stored, Kube-scheduler that schedules the new pending pods, Kube and cloud control manager.

Node component is made up of Kubelet that ensures pods are running with containers, Kube-proxy that maintains the pod network and some add on components like WebUI, DNS management and resource monitoring.

2.2 Review of scheduling strategies and algorithms

Existing strategies provides no method to emphasize and have a majority stake in decision of node selection based on a particular parameter. Researchers have published strategies to utilise the importance of one particular parameter and schedule a container aimed at optimising the usage of that particular resource. Cerin et al. (2017) has proposed the algorithm to reduce SLA violations. SLAs are important for service provider and violation of SLA leads to degradation of service and loss of revenue. The author proposes the scheduling based on the SLA class of the user. Although the author uses their

[1]https://github.com/docker/swarmkit
[2]https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

algorithm to select a container to schedule among the multiple requests at a time and uses Binpack strategy to schedule the container for economic purposes. Alahmad et al. (2018). have focused on optimisation and improvement of availability of the application via scheduling The algorithm calculates the availability of the system by using the mean time to fail and repair data and places the container according to the calculation.

Vaucher et al. (2018) has focused on security improvements while scheduling containers. This algorithm uses Intel's latest security feature SGX. SGX provides secure hardware environment for the execution of programs and the author suggests to use this feature and optimise the use of SGX while scheduling and thereby increasing the overall security of the environment. Wu and Chen (2017) has proposed an ABP scheduler which also focuses on saving SLA violations. The author suggests an algorithm to save network tra c cost by using common Docker hub base images of containers as a bu er layer. This layer enables faster scheduling time thereby avoiding SLA violation. Testing shows a drastic reduction in allocation time with a reduction from 10 seconds to 2 seconds.

Bhimani et al. (2018) has published a report for IO-intensive applications. The author has studied the performance of di erent IO-intensive applications under various conditions. Guidelines were designed for a homogeneous and heterogeneous mixture of application for operational optimisation. The author has developed a new docker scheduling algorithm by using the previously developed guidelines. The author aims to reduce total execution time and increase total resource utilisation by deciding the optimal number of containers in a batch where several containers are executing concurrently. The algorithm tries to balance the throughput of all running applications and testing is done with di erent applications requiring di erent IO. When compared without the algorithm execution time of applications shows improvements with the new algorithm implementation.

Research is in place for better utilisation of overall resources after scheduling. heuristic and meta-heuristic algorithms are implemented for resource utilisation improvement. Kaewkasi and Chuenmuneewong (2017) has proposed ant colony optimisation. Ant-Colony algorithm is a meta-heuristic algorithm and the author has proposed to implement it for scheduling strategy. results show promising results of 15% improvement in utilisation compared to general Greedy algorithm. Many implementations are published on similar grounds aimed at achieving better utilisation of resources is present. Menouer et al. (2018) has implemented the PROMETHEE and Kung algorithm and Menouer and Darmon (2019) has used TOPSIS algorithm in another publication. These mentioned algorithms are well-known algorithms in the family of Multi-Criterion decision making algorithm family.

Li and Fang (2017) states that a single algorithm that ts for every condition is not feasible. There are mainly three scenarios in the orchestration area.

1. Some applications on the node are non containerised.
2. Resource utilisation of some containers is not balanced and has unbalanced resources utilisation of one particular resource.
3. Requests are unbalanced between containers.

All these cases are not covered with one algorithm and the author suggests to use multi-algorithm collaboration scheduling strategy (MCSS) to tackle the situation. The

author aims to increase the overall performance of the scheduled containers. This algorithm uses probability and modi ed weighted scheduling algorithm to satisfy basic needs such as cluster load balance. Trend Based resource balancing allocation algorithm for multiple resource requirement conditions and Delay scheduling algorithm for solving concurrency problems. Testing shows an increase in performance and improved cluster load balancing.

Choi et al. (2016) has published a paper on similar grounds. The author states that container scheduling is treated as Linux kernel process and they are allocated resources on the same basis but is a fundamental di erence between Linux kernel process and container. Containers can be of Di erent SLAs, IO sensitive and some of them are real-time services. The author has proposed a General Purpose Scheduling Framework (GSPF) for container scheduling. This framework provides options for customised scheduling strategies and options to optimise resource management. This framework is situated between container and kernel to optimise resources and communicate with docker container as well. This framework contains three separate modules which are monitor, scheduler and resource allocator. GSPF also provides di erent APIs for each of its modules to get status of container, resource stats and assign resources to speci c containers which can be used with a custom scheduler. The implementation shows 13% improvement in throughput of applications.

Another strategy followed is to assign a node for the container by using Spread and Binpack strategy and then migrate the containers according to prominent resource utilisation or node resources availability. Li et al. (2019) suggest to identify prominent resource utilisation and group them by resource usage called an RPBG method. containers are migrated to another node when the maximum limit of resource utilisation is breached. Mao et al. (2017) has also proposed a method which is along the same line of monitoring and migration technique. The author has called his algorithm as DRAPS where resource bottlenecks are identi ed and migrated based on resource usage of the container. Implementation showed about 42% increase in overall utilisation of the system with this method.

2.3 Algorithms Under MCDM Category

Velasquez and Hester (2013) have presented a paper with detailed analysis, usage, literature review and advantage over each other of major Multi-Criterion Decision Making algorithms. The author states that the development of MCDM algorithms is going on for several decades and variations of each algorithm are published for usage in speci c real-world scenarios. Although algorithms are very complex and provide a meaningful and logical framework for planning and decision making some algorithms are well suited to some particular situations. The author states 11 major di erent MCDC algorithms like ELECTRE, TOPSIS, PROMETHEE, Fuzzy set theory and case-based reasoning and others. Many of these algorithms have di erent variations like ELECTRE has ELECTRE I, II, TRI, IV and IS.

The author states that TOPSIS has an advantage where steps remain xed irrespective of alternatives and several attributes where some algorithms vary based on input. On the other hand algorithms like ELECTRE and Multi-Attribute utility theory takes uncertainty and vagueness into account. Fuzzy set theory is utilised best under improper

and insu cient input and goal programming can give us in nite alternatives. TOPSIS can be used e ectively in areas like supply chain management, resource management, engineering and manufacturing systems.

2.4 Review of TOPSIS Algorithm

TOPSIS is short for Technique for Order of Preference by Similarity to Ideal Solution. Introduced in 1981, and the nal re ned version is published by Hwang et al. (1993). This algorithm uses the theory of Euclidean distance for the positive and negative ideal solution. Panda and Jagadev (2018) have surveyed the detailed analysis, application and usage of this method. The author states that this method was used to calculate performance scores and for stock market analysis and prediction in the Istanbul Stock Market. Supply chain management area uses TOPSIS for maintaining a balance between quality, quantity and cost control and choosing the best partner for the association. Some classi cation and clustering problems are also solved using this method. A Variation of this method, called D-TOPSIS, is used for resource selection in human resource department.

Although TOPSIS has been used extensively in di erent areas other than engineering, this algorithm is also suitable for application in engineering technologies. This paper uses TOPSIS for node selection criterion where alternatives are available nodes in the cluster, and criteria are system parameters of nodes.

2.5 Comparison of Reviewed Techniques

Overall comparison of scheduling strategies is done with below table. It can be observed that custom scheduling is implemented with various objectives such security, availability and optimisation of overall resources.

No	Scheduling Strategy	Area of Impact	Author
1	Algorithm Based on SLA class of User	Reduce SLA violations	Cerin et al. (2017)
2	Availability values of infrastructure	Increase the Availability of Application	Alahmad et al. (2018)
3	SGX aware scheduling	Increase security of Application	Vaucher et al. (2018)
4	Ant-Colony optimisation strategy	Increase overall resource utilisation	Kaewkasi and Chuenmuneewong (2017)
5	General Purpose Scheduling Framework (GSPF)	Optimise resources	Choi et al. (2016)
6	Observe and migrate method	Improve overall utilisation	Mao et al. (2017)
7	PROMETHEE and KUNG algorithm	Improve overall utilisation	Menouer et al. (2018)

3 Methodology

TOPSIS is made up of 6 xed steps irrespective of number of alternatives and criteria. Algorithms needs 3 Di erent inputs i.e. Input decision matrix, Weight matrix, and Cost-

Bene t matrix. Input Matrix is Normalised and then weights are applied to normalised matrix. Positive and Negative Ideal Solution are found out to calculated Positive and Negative Euclidean distance. This distance is used to calculate closeness parameter based on which alternates are ranked.

Detailed steps and mathematical formulae are given below.

Step 1 - Normalise the Input Matrix

Input Matrix is created where A is alternates which are available nodes in the cluster setup created and C is criterion for selection where criterion is system parameter value such as CPU, RAM and storage utilisation.

Input Matrix:

$$\begin{matrix} C1 & C2 & \dots & Cn \\ \begin{pmatrix} X_{11} & X_{12} & \dots & X_{1q} \\ \vdots & \vdots & \vdots & \vdots \\ X_{p1} & X_{p2} & \dots & X_{pq} \end{pmatrix} & & & \begin{matrix} A_1 \\ \vdots \\ A_n \end{matrix} \end{matrix} \tag{1}$$

Normalisation of Matrix:

Normalisation is done to remove the units of each values. each criterion value is in di erent units where RAM is measured in MB or GB, Storage is measured in GB or TB while CPU is just number. normalisation makes the matrix unit less so that further processing can be done on the matrix.

$$[G_{ij}]_{pq} = \frac{X_{pq}}{(\sum_{i=1}^{p} X_{pq}^2)^{1/2}} \tag{2}$$

Stemp 2 - Weighted Normalised Matrix:

Weight matrix is an input matrix where values are weight of each criterion to be considered in the calculation.

$$W_j = \begin{bmatrix} W_1 & W_2 & \dots & W_q \end{bmatrix} \tag{3}$$

This input matrix is multiplied with normalised matrix calculated above to get weighted normalised matrix.

$$[Z_{ij}]_{pq} = G_{ij} \quad W_j \quad \text{where} \quad i = 1...p, j = 1...q \tag{4}$$

where G_{ij} is normalised matrix calculated in previous step 2 and w_j is weight of criterion.

Step 3 - Positive Ideal Solution and Negative Ideal Solution

Positive ideal and Negative ideal is calculated based on bene t and cost factor given as input. If the Criterion is bene t factor then Max value of particular criterion is PIS and when criterion is cost then min value of particular criterion column is PIS and vice vars for NIS.

$$PositiveIdealSolution(PIS) : \begin{cases} max(Z_{ij}jC+) \\ min(Z_{ij}jC\ \ ) \end{cases} \tag{5}$$

where C+ is Bene t and C- is cost

$$NegativeIdealSolution(NIS) : \begin{cases} min(Z_{ij}jC+) \\ max(Z_{ij}jC\ \ ) \end{cases} \tag{6}$$

where C+ is Bene t and C- is cost

Step 5 - Euclidean Distance from PIS and NIS

Positive and Negative Euclidean distance for each alternate is calculated based on PIS and NIS value calculated in previous step.

$$P_i = (\sum Z_{ij}^2 \quad Z_j)^{1/2} \quad \text{where} \quad Z_j = PIS \tag{7}$$

$$P_i = (\sum Z_{ij}^2 \quad Z_j)^{1/2} \quad \text{where} \quad Z_j = NIS \tag{8}$$

Step 6 - Closeness Parameter

Closeness parameter is ratio of Negative euclidean distance to total euclidean distance

$$P = P_i \ /(P_i \ + P_i) \tag{9}$$

Alternates are ranked based on this closeness parameter where highest rank is given to the maximum value of closeness parameter and vice vars.

3.1 Pseudo Code

4 Design Specification

Cloud implementation of the project is done using KOPS *KOPS* (n.d.). Kops helps in creating, upgrading, destroying and managing the kubernetes cluster on various cloud providers such as AWS, Google cloud and more. Kops is o cially supported on AWS. Architecture diagram of this implementation is given with g 1.

Following key features of Kops makes it production grade kubernetes cluster:

1. KOPS create and destroys VPC from scratch
2. It supports multiple master nodes

Algorithm 1 Docker Scheduling

Input Input request for scheduling

Output Node is selected by algorithm and container is scheduled.

0: Getting the system information of each node.
0: **for** $iteration$ = $1, 2, \ldots, P$ **do**
0: **for** $iteration$ = $1, 2, \ldots, Q$ **do**
0: Get the values of Q^{th} criterion for P^{th} hosts.
0: Prepare P x Q matrix.
0: **end for**
0: **end for**
0: Preparing the weight matrix by getting the values from user.
0: **for** $iteration$ = $1, 2, \ldots, Q$ **do**
0: Get weight of Q^{th} criterion.
0: **end for**
0: Pass the Input and Decision matrix to TOPSIS algorithm python le.
0: Assign output ranking of function call to variable.
0: **if** User de ned conditions **then**
0: Filter out host that does not satisfy conditions.
0: **end if**
0: Get highest ranking out of remaining host
0: Schedule a container on highest ranked host.
=0

3. Support for horizontal scaling
4. Integration with CloudFormation and Terraform
5. Support for multiple instance groups, public and private topology.

Kops automatically creates many AWS components is taken care by KOPS like Networking, VPC creation, creation of instance groups but some AWS pre-requisites are done manually. IAM policy, User creation, SSH con guration, S3 bucket, DNS con guration are the tasks that are done manually. Kops is also suggested in the book Cloud Native DevOps with Kubernetes by Arundel and Domingus (2019) Following AWS services are used for kubernetes implementation:

1. Simple Storage S3: This storage is used to store the state of the cluster. S3 bucket with unique name is created in EU-WEST-1A region. Bucket name created for this con guration is kops-state-anurag.
2. Route 53: Kops require DNS name under which all the con guration is created. cluster is created and deleted under the speci ed DNS. Route 53 is used for hosting the DNS name. Name servers provided by Route 53 are integrated with DNS name provider.
3. IAM policy: Kops require full access to EC2 machines, S3 bucket, Route 53 and VPC. IAM policy is set for the user accordingly.

Algorithm 2 TOPSIS Algorithm

Input Inputs passed by algorithm 1
Output $A_1 > A_2 > \ldots > A_n$.

0: Assign input from Algorithm 1 to local matrix Xpq.
0: Calculation of Normalised matrix.
0: **for** $iteration$ = $1, 2, \ldots, P$ **do**
0: **for** $iteration$ = $1, 2, \ldots, Q$ **do** Normalise the Matrix.
0: **end for**
0: **end for**
0: Multiplying normalised matrix with respective weights.
0: **for** $iteration$ = $1, 2, \ldots, P$ **do**
0: **for** $iteration$ = $1, 2, \ldots, Q$ **do**
0: G_{pq} * W_q
0: **end for**
0: **end for**
0: We get normalised matrix Z_{pq}.
0: Positive Ideal Solution calculation.
0: **for** $iteration$ = $1, 2, \ldots, P$ **do**
0: **for** $iteration$ = $1, 2, \ldots, Q$ **do**
0: Max of Z_k.
0: **end for**
0: **end for**
0: Negative Ideal Solution calculation.
0: **for** $iteration$ = $1, 2, \ldots, P$ **do**
0: **for** $iteration$ = $1, 2, \ldots, Q$ **do**
0: Min of Z_k.
0: **end for**
0: **end for**
0: Calculation of Euclidean best value
0: **for** $iteration$ = $1, 2, \ldots, i$ **do**
0: **for** $iteration$ = $1, 2, \ldots, j$ **do**
0: $(\sum Z_{ij}^2 \quad Z_j^+)^{1/2}$
0: **end for**
0: **end for**
0: Calculation of Euclidean worst value
0: **for** $iteration$ = $1, 2, \ldots, i$ **do**
0: **for** $iteration$ = $1, 2, \ldots, j$ **do**
0: $(\sum Z_{ij}^2 \quad Z_j)^{1/2}$
0: **end for**
0: **end for**
=0

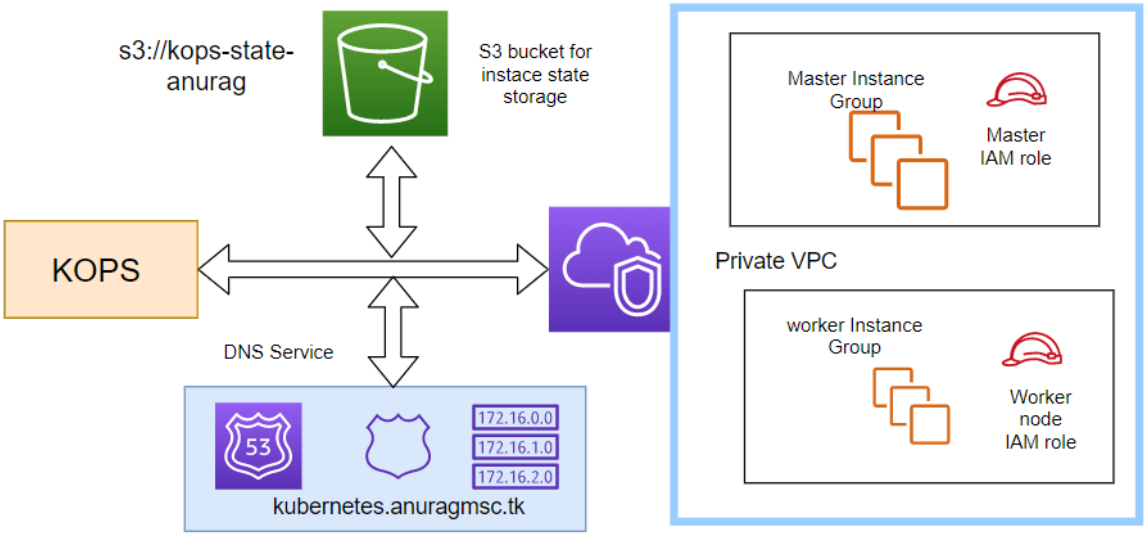

Figure 2: Kops architecture created on AWS

4. EC2 Machines : Kops creates EC2 machine for hosting Kubernetes master and worker node.

5. VPC: VPC creation is not a manual task, It is created through KOPS internal scripts. User is free to make changes in VPC con guration according to requirement. This implementation uses default VPC created by KOPS.

Algorithm is integrated with the kubernetes after implementation of cloud infrastructure. Architecture of algorithm implementation is divided into 3 main parts Input matrix creation, TOPSIS algorithm and container scheduling. Architecture diagram of this integration is shown with gure 2.

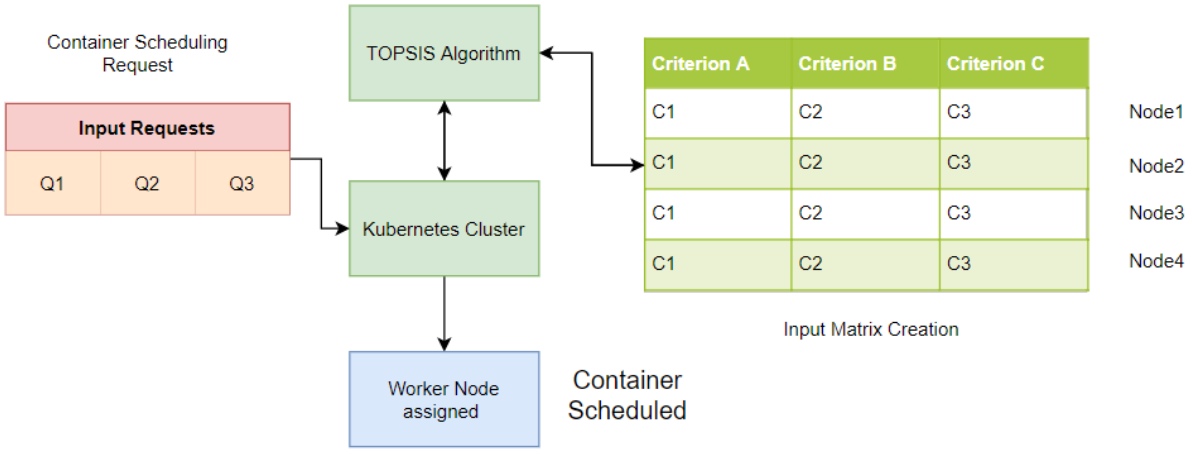

Figure 3: Architecture Diagram of Algorithm Integration

Normal kube-scheduler works in following way and process ow diagram shown in in gure 4:

1. etcd is used to save state of newly created pod but node is not assigned to pod.
2. Kube-Scheduler checks every pods in loop and nds out that pod is not assigned any node.
3. Kube-Scheduler nd best node for the pod.
4. Kube-Scheduler call API server and binds a node to a pod.
5. Kubelet schedules the particular pod on the speci ed node.

Implemented strategy changes and adds a step in between these steps:

1. Python code is written to loop through the pods and nd out unbounded pods is written.
2. TOPSIS algorithm to nd out best node.
3. A call to API server to bind this node to pods is made.

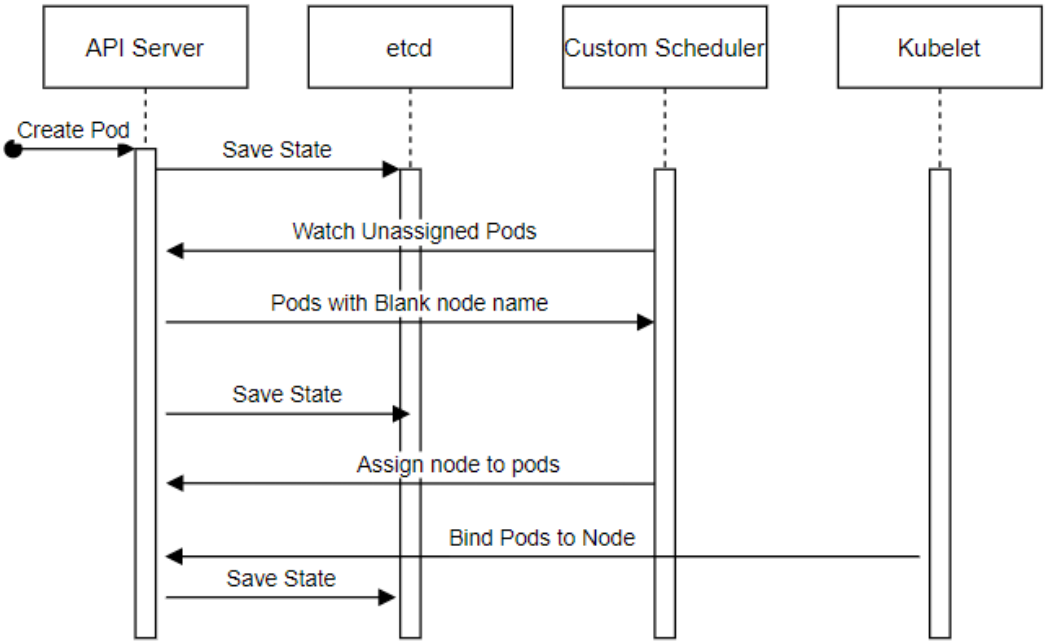

Figure 4: Process ow of custom scheduler

5 Implementation

Implementation of the algorithm is done using python. Di erent light weight python libraries are used for fast paced implementation. Kubernetes provides Kubernetes library for python which is used for container scheduling. Following is list of libraries used in the implementation.

1. Kubernetes: This is o cial kubernetes library for python for carrying out kubernetes commands with the help of python code.

2. PSUTIL: This library is used to collect system information from the available nodes. psutil provides CPU, RAM, swap space, storage read and write latency, network packets sent and received from the host. Preeth E N et al. (2015) used this utility in evaluation of docker.

3. Paramiko: This utility is used for multiple SSH connection at the same time. It is also used to execute script located at the remote machine, collect the information and bring back the output to central location.

4. Numpy : Numpy is used for matrix calculation in algorithm implementation.

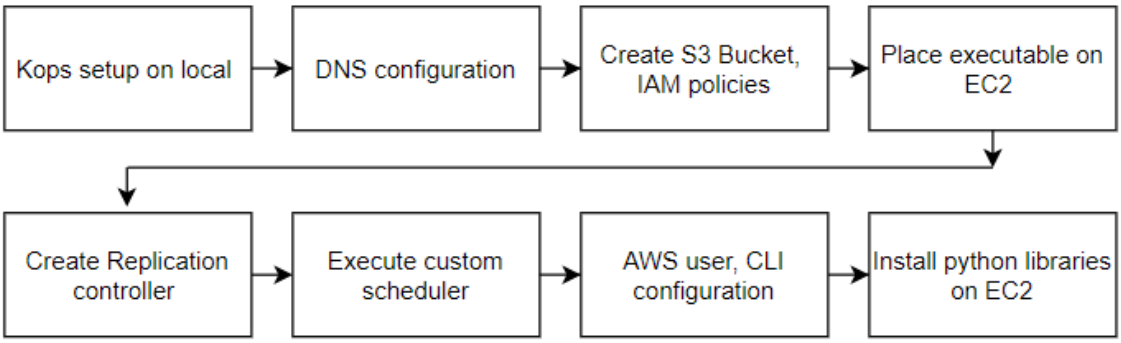

Figure 5: Process ow of implementation

Implementation is divided into two parts where one script is placed on every worker node on the cluster (system_parameters.py) and another script (custom_scheduler.py) is used to call this script from the local machine, perform TOPSIS algorithm, provide the best alternative and schedule the pending containers on the node.

As this implementation is with AWS, AWS CLI is con gured with the new user created for this project. User is given Full administrative access with Programmatic access. This access is required as python commands are used for AWS con gurations.

KOPS is downloaded on local Linux machine and cluster is created with the desired number of Master and Worker nodes. SSH is con gured with public and private key with all the EC2 machines. This SSH private key is used in the python code for remote execution of the script on the EC2 machine. Process ow diagram of the implementation process is shown with gure 5

Pods are con gured and created to adopt the custom Kubernetes scheduler. As stated in previous sections, Kubernetes provides a feature to accommodate custom scheduler. Pods cluster is created with a replication controller. All the details of the replication controller are written in YAML le. An extra tag of **schedulerName:** is added in the YAML le with scheduler name for this feature. This same scheduler name is given in python code written.

When pods are created with this replication controller YAML le, they stay pending until custom scheduler python le is executed and once the le is executed, pending pods are scheduled on a best available node.

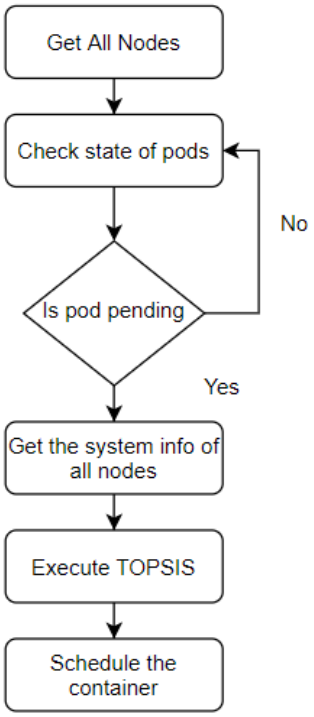

Figure 6: Flow chart diagram of custom scheduler

The custom scheduler checks the available nodes in the cluster which includes both master and client. It checks the state of all pods and gets the pending pods in the next step. If there are pending pods then it gets the system information of all the available client nodes and creates a matrix. Apply the TOPSIS algorithm and get the best node. This node is used to schedule all the pending state pods. Flow chart of the custom scheduler is given with gure 6.

6 Evaluation

Evaluation of the project is done on local implementation on the laptop. Kubeadm, Kubelet and Kubectl are used for Kubernetes implementation on VirtualBox machine. 1 master node and 3 client nodes are set up for testing of this project. All these VM's are interconnected with each other with the same IP range so that master can control client nodes with SSH commands. Creation of a heterogeneous environment is not possible on a cloud platform where all the worker nodes are created with the same con guration. Hence evaluation of the project is done on the local cluster setup. cluster is setup using the following components.

1. **Kubeadm :** This ensures the bootstrapping of all the components in kubernetes cluster.

2. **Kubelet :** It manages the components like starting and stopping of all pods and kubelet runs on all the nodes of the cluster.

3. **kubectl :** Command line utility to manage kubernetes.

6.1 Scenario 1

Testing of the project in this scenario is done where the con guration of all worker nodes is kept the same. The weight associated with characteristics is distributed equally and stress testing tool is used to generate arti cial load on the worker nodes. Virtual memory is utilised by using stress testing tool STRESS-NG. 70% of the memory is utilised by the tool, and then scheduling is checked with Kubernetes and custom scheduler. Stress-NG was used by Ismail and Riasetiawan (2016) for testing CPU and Memory performance in Xen data center. Following table 6.1 shows % of the memory utilised by the testing tool and node selection of the custom algorithm and Kubernetes scheduler.

Test Case	Kclient 1	Kclient 2	Kclient 3	Kubernetes Scheduler	Custom Scheduler
1	0%	0%	70%	Node 1	Node 1
2	70%	0%	70%	Node 2	Node 2
3	70%	50%	70%	Node 2	Node 2
4	70%	70%	0%	Node 3	Node 3

Custom Kubernetes scheduler and existing Kubernetes scheduler shows the same node selection when all the node con guration is the same. Selection of the node of both the methods remains the same even after some nodes are put under load using stress-ng. The node which is not fully utilised with the resources is selected for scheduling with both custom and Kubernetes algorithm. This can be observed in case 3 where node 2 is less utilised as compared to node 1 and 3 and Node 2 is selected for the scheduling. When only one of the node is put under load test then best node out of other 2 is selected for scheduling. Node selection for scheduling is the same for both the methods in this case also.

6.2 Scenario 2

This evaluation is done with varying weight of characteristics. Random values of CPU and RAM are assigned for client nodes in the con guration. Di erent test cases are executed with the same con guration of the machines but di erent weight value. Impact of the weight assigned to characteristics is evaluated in this testing. Con guration of the cluster is given as follows Master Node: 2 CPU, 3GB Ram and 40GB storage , Client Node 1: 2 CPU, 1.5 GB RAM, 35GB storage, Client Node 2: 1 CPU, 1.5 GB RAM, 30GB storage and Client Node 3: 1 CPU, 3 GB RAM, 30GB storage

All the system information is passed to TOPSIS algorithm and algorithm provides the best node from the provided input. Stepwise calculations of TOPSIS algorithm for one such case is given below.

When a pod is created with the help of replication controller and using YAML le where scheduler name is given as custom scheduler, it remains in the pending state till

```
anurag@kmaster:~/Desktop/Scheduler$ cat testoutput.txt
Input Data is :
[[20 12 35 28 18]
 [10 16 30 24 17]
 [10 25 30 25 18]]
Weight Of the Matrix is :
[20, 20, 20, 20, 20]
Normalised Array :
[[0.22983951 0.13790371 0.40221915 0.32177532 0.20685556]
 [0.11491976 0.18387161 0.34475927 0.27580742 0.19536359]
 [0.11491976 0.28729939 0.34475927 0.28729939 0.20685556]]
weighted Normalised Matrix :
[[4.59679026 2.75807416 8.04438296 6.43550637 4.13711124]
 [2.29839513 3.67743221 6.89518539 5.51614832 3.90727172]
 [2.29839513 5.74598783 6.89518539 5.74598783 4.13711124]]
Ideal Best Array Is :
[4.59679026 5.74598783 8.04438296 6.43550637 4.13711124]
Ideal worst is :
[2.29839513 2.75807416 6.89518539 5.51614832 3.90727172]
Best Euclidiean Distance :
[4.3426897242612705, 5.0261183467476425, 8.074352122151666]
worst Euclidiean Distance :
[7.681592764930557, 5.897578679960546, 6.105435491988965]
12.0242824892
10.9236970267
14.1797876141
Performance Score :
[0.6388400116044554, 0.5398885254269778, 0.43057312691343763]
Best alternate is : 1
```

Figure 7: Step wise calculations of TOPSIS algorithm for test case 1

custom scheduler script is not executed. Figure 8 shows the creation of pod with YAML le nginxrc.yaml and pod in the pending state. Age of the pod is shown as 23s.

```
anurag@kmaster:~/Desktop/Scheduler$ kubectl create -f nginxrc.yaml
replicationcontroller/nginx created
anurag@kmaster:~/Desktop/Scheduler$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP       NODE     NOMINATED NO
DE   READINESS GATES
nginx-w24z2   0/1     Pending   0          23s   <none>   <none>   <none>
     <none>
anurag@kmaster:~/Desktop/Scheduler$ python custom\ scheduler.py
Input IP is 192.168.56.102
kclient
Scheduling nginx-w24z2
```

Figure 8: Pods in Pending state after creation with Custom Scheduler

The custom scheduler receives the output of TOPSIS algorithm and it schedules the pending pods on the respective node. Figure 9 shows the container scheduled on node kclient. Internal IP is assigned to a pod and random name is generated for a pod. state of the pod can be seen as Running in the gure.

```
anurag@kmaster:~/Desktop/Scheduler$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP               NODE      NO
MINATED NODE   READINESS GATES
nginx-w24z2   1/1     Running   0          86s   192.168.224.116   kclient   <n
one>           <none>
```

Figure 9: Pods scheduled on alternative

Con guration of all the machines is kept same for the testing purpose and weight assigned for each characteristics is changed for each test case. Following table 6.2 depicts the weights for each test cases.

Test Case	CPU	RAM	Total Storage	Available Storage	CPU Freq
1	20	20	20	20	20
2	30	20	15	15	15
3	35	40	10	10	5

Results of above 3 cases is as follows.

Test Case	Custom Scheduler	Kubernetes Scheduler
1	Node 1	Node 1
2	Node 2	
3	Node 3	

Case 1: Custom scheduler and inbuilt kubernetes scheduler selects the same node for scheduling which is Node 1 when equal distribution of the weights is provided to algorithm.

Case 2 & 3: Algorithm selects di erent node for scheduling when unequal distribution of the weight is given as input. Node 3 with highest RAM is selected when 40% weight is given for input and node 2 is selected when some percentage of weights assigned to storage is increased to 15% from 10%.

7 Conclusion and Future Work

Docker shows seamless integration with orchestration tool Kubernetes and Kubernetes has gained widespread popularity. Although Kubernetes is complete package compared to other orchestration tools, there is a scope of improvement in the scheduling area. One for all scheduling strategy is known to have caused problems in scheduling large scale systems and does not take any inputs from the user as far as importance assigned to speci c characteristics is considered.

The implemented strategy shows same scheduling decisions as compared to Kubernetes scheduler when all characteristics are given equal importance but this strategy also provides a way to change the importance of certain characteristics in the decision making of node selection. If the unequal distribution of weight is assigned to a system parameter then implemented algorithm shows deviation from Kubernetes scheduler and this deviation is according to user preference.

Future work of the project includes implementation on a large scale system with 2000 to 3000 pods on large scale cluster. 1) architectural changes to current implementation so that system parameter will be collected by API calls rather than SSH connection which takes higher time. 2) Implementation of multi-container scheduling considering the resource requirement of each container in the queue. 3) Usage of di erent networking parameters in the input matrix generation stage.

References

Alahmad, Y., Daradkeh, T. and Agarwal, A. (2018). Availability-aware container scheduler for application services in cloud, *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, Orlando, FL, USA, pp. 1{6. Core Ranking:B.

Arundel, J. and Domingus, J. (2019). *Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud*, O'Reilly Media.

Bhimani, J., Yang, Z., Mi, N., Yang, J., Xu, Q., Awasthi, M., Pandurangan, R. and Balakrishnan, V. (2018). Docker container scheduler for i/o intensive applications running on nvme ssds, *IEEE Transactions on Multi-Scale Computing Systems* 4(3): 313{326.

Choi, S., Myung, R., Choi, H., Chung, K., Gil, J. and Yu, H. (2016). Gpsf: general-purpose scheduling framework for container based on cloud environment, *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, pp. 769{772.

Cerin, C., Menouer, T., Saad, W. and Abdallah, W. B. (2017). A new docker swarm scheduling strategy, *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, Kanazawa, Japan, pp. 112{117.

Gartner (2018). Hype cycle for cloud computing, 2018.
URL: *https://www.gartner.com/en/documents/3884671/hype-cycle-for-cloud-computing-2018*

Hwang, C.-L., Lai, Y.-J. and Liu, T.-Y. (1993). A new approach for multiple objective decision making, *Computers & operations research* 20(8): 889{899.

Ismail, H. A. and Riasetiawan, M. (2016). Cpu and memory performance analysis on dynamic and dedicated resource allocation using xenserver in data center environment, *2016 2nd International Conference on Science and Technology-Computer (ICST)*, IEEE, pp. 17{22.

Kaewkasi, C. and Chuenmuneewong, K. (2017). Improvement of container scheduling for docker using ant colony optimization, *2017 9th International Conference on Knowledge and Smart Technology (KST)*, Chonburi, Thailand, pp. 254{259. Total Citations:31.

KOPS (n.d.).
URL: *https://github.com/kubernetes/kops*

Kubernetes Components (n.d.).
URL: *https://kubernetes.io/docs/concepts/overview/components/*

Li, H., Chen, N., Liang, B. and Liu, C. (2019). Rpbg: Intelligent orchestration strategy of heterogeneous docker cluster based on graph theory, *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, Porto, Portugal, pp. 488{493. Core Ranking:B.

Li, Q. and Fang, Y. (2017). Multi-algorithm collaboration scheduling strategy for docker container, *2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)*, IEEE, pp. 1367{1371.

Mao, Y., Oak, J., Pompili, A., Beer, D., Han, T. and Hu, P. (2017). Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster, *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, San Diego, CA, USA, pp. 1{8. Core Ranking:B.

Menouer, T., Cerin, C. and Leclercq, E. (2018). New multi-objectives scheduling strategies in docker swarmkit, *in* J. Vaidya and J. Li (eds), *Algorithms and Architectures for Parallel Processing*, Springer International Publishing, Cham, pp. 103{117. Core Ranking: B.

Menouer, T. and Darmon, P. (2019). New scheduling strategy based on multi-criteria decision algorithm, *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, IEEE, Pavia, Italy, pp. 101{107. Core Ranking:C.

Panda, M. and Jagadev, A. K. (2018). Topsis in multi-criteria decision making: A survey, *2018 2nd International Conference on Data Science and Business Analytics (ICDSBA)*, IEEE, pp. 51{54.

Preeth E N, Mulerickal, F. J. P., Paul, B. and Sastri, Y. (2015). Evaluation of docker containers based on hardware utilization, *2015 International Conference on Control Communication Computing India (ICCC)*, pp. 697{700.

Vaucher, S., Pires, R., Felber, P., Pasin, M., Schiavoni, V. and Fetzer, C. (2018). Sgx-aware container orchestration for heterogeneous clusters, *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, pp. 730{741. Core Ranking: A.

Velasquez, M. and Hester, P. T. (2013). An analysis of multi-criteria decision making methods, *International journal of operations research* **10**(2): 56{66.

Wu, Y. and Chen, H. (2017). Abp scheduler: Speeding up service spread in docker swarm, *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, IEEE, Guangzhou, China, pp. 691{698. Core Ranking: B.