National College of Ireland

Capacity Aware Container Placement in Heterogeneous Clusters using Genetic Algorithm

MSc Research Project
Cloud Computing

Bhavna Thakur
Student ID: x18145914

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Bhavna Thakur
Student ID:	x18145914
Programme:	MSc in Cloud Computing
Year:	2019
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	12/12/2019
Project Title:	Capacity Aware Container Placement in Heterogeneous Clusters using Genetic Algorithm
Word Count:	6582
Page Count:	27

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

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

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:	Bhavna Thakur
Date:	2nd February 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 sufficient to keep a copy on computer.	☐

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

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

Capacity Aware Container Placement in Heterogeneous Clusters using Genetic Algorithm

Bhavna Thakur
x18145914

Abstract

High performance application execution in a containerized environment has been a popular field of research in Cloud Computing. With Docker containers being lightweight and providing complete isolation, executing tasks with heavy resource demands has become effortless. Due to the ubiquity of containers,highly heterogeneous and unpredictable workloads proceed that sometimes result in resource exhaustion leading to start-up latency which can become a hindrance in the execution process. As the resource requirements from these diverse workloads cannot be always matched accurately, there is a probability for resource contention, which limits the scaling levels for further task-based containers and generates a significant delay. Therefore, there persists a trade-off between resource utilization and latency. Despite prevalent research work being done in this area, there still remains some open issues in the field of optimized container allocation in the cluster. In this research, Genetic Algorithm is used to generate a capacity-aware container placement method to ensure there is a significant amount of resource available to service the deployed application tasks with different resource requirements by creating a fitness function based on the capacity threshold parameter. On implementation, it is observed that there is a 40% increase in the resource utilization level, with 5000 seconds of improvement in execution time. On comparing the individual utilization values with the average utilization of the cluster, a balanced cluster is seen to be achieved.

1 Introduction

1.1 Brief History

The concept of Virtualization became an enabler for Cloud Computing to emerge as a platform to deploy large scale applications in distributed mode. Reliability and fault tolerance are provided using virtual machine technology by allowing access to surplus resources for their execution with pay-as-you-go and high availability being the key performance indicators of Cloud Computing. They paved way for virtualized clusters using virtual machine instances to service intense user demands having dynamic as well as static workloads. But performance overhead increased for hardware level virtualization, due to their heavy operational requirements. The solution to these challenges was OS level virtualization which allowed multiple isolated VMs to exist in an OS kernel.

Most of the OS level virtualization systems are based on Linux, and Docker is one such software that runs on linux kernel functions allowing applications to run on lightweight

containers. It runs the containers in isolation where software processes can be executed consistently and ensures a common platform such that any software can be implemented regardless of where it is deployed. Standalone containerized units are furnished which can envelope the entire application along with its dependencies. With fast and flexible application deployment and transparency in resource sharing, containers have become popular to use for large scale application execution across multiple clusters of compute nodes.

1.2 Evolution of Container Orchestration and Placement Strategy

Various container orchestration platforms are made bespoke to manage and cater to the needs of the deployed tasks by the scheduler. With host selection for container schedulers a huge problem to match for node compatibility, the cluster manager manages its hosts by fetching the loading information of every host along with the containers deployed in that cluster and their execution demands to scale them when required. The docker administrator manages the group of containers as one application which needs synchronization in execution times. Container orchestration handles the entire lifecycle of containers starting from their deployment, availability, scaling, load balancing, resource allocation, container migration and monitoring. Popular orchestration platforms like Kubernetes, Swarm, Apache Mesos were created for the deployment and management of applications packaged in containers for efficient execution [[1],[2],[3]].

Containers have the ability to modify the resources being consumed at runtime, enabling them to adjust the resource usage based on task concurrency and execution needs.When a task is launched in the cluster, the scheduler decides the appropriate instance to place the task. It also decides the container that needs to be terminated from the cluster while scaling down. The importance of an efficient Placement strategy lies in the role it plays in defining the overall performance of applications where Quality of Service or QoS parameters like reliability, elasticity, availability and resource utilization needs to be considered.[4] provides a detailed evaluation of the challenges faced in container-based performances in their study. The authors specify that containers are yet to provide resources for distributed application or policy-specific services.

1.3 Motivation

There has been an extensive research and development done on task schedulers for containerized applications that majorly aim to find an optimal method of placing containers to minimize cost overhead occurring due to imbalanced utilization of resources and networking overhead generated from bad placement decisions. There persists various performance issues when users want a specific available capacity meant to service all the different work streams existing in the application. There is a need to minimize the cold starts for these work streams in response to a new job request queued. For example, a research group can run hundreds of potentially different sequencing algorithms which can be compute-intensive, enabling them or any user to execute custom workflows anytime. So, capacity availability and fast results are both important[5]. There is a need to resolve this issue by not only creating a capacity awareness in the cluster but also ensuring there is a resource utilization balance maintained in the cluster. This study evaluates how can the resources required for a task execution be allocated faster each time in order to not only avoid startup latency but to also ensure maximum utilization of resources ensuring

capacity availability at the same time to achieve resource usage cluster balance.

1.4 Research Question

Does capacity aware distribution of containers in heterogeneous clusters using Genetic Algorithm increase the resource utilization level and change the overall performance during containerized task execution?

This study aims in ensuring a capacity aware container allocation is done in a heterogeneous cluster of resources such that a significant amount of resources are available to be used for the next workload having unpredictable resource demands in order to minimize the startup latency caused during task execution. The study also aims to evaluate whether the resource is used in a balanced way such that there is a state of equal resource sharing maintained in the cluster.

In order to meet the research objective for improving the overall performance and startup time of application execution in a heterogeneous cluster, Genetic Algorithm is used, which is a meta-heuristic, evolutionary algorithm. Table 1 shows the gaps in existing algorithms used for container placement.

Existing Placement Strategies	***Problems***
Spread Scheduling	Does not utilize the space completely
Binpack	Some nodes maybe overloaded
Random	Random allocation which cannot be controlled

Table 1: shows the shortcomings of the existing algorithms used in Dockers as discussed by [6]

To ensure capacity awareness is maintained, the fitness evaluation of Genetic Algorithm will comprise the capacity aware value which will generate the best fit solutions. These results will then be evaluated in Kubernetes to observe the change in resource utilization level, execution time and cluster balance.

1.5 Report Structure

The rest of the report is structured as follows. Section 2 presents relevant work done in the area of optimal container allocation using various methodologies. Section 3 discusses the steps followed for preparing the environment and data along with the measurements performed. Section 4 gives an overview on the architecture implemented, and its detailed implementation in Section 5. Section 6 presents evaluation of the proposed strategy and discusses the effectiveness of the deployed framework on Kubernetes. Section 7 summarises the work done, identifies and discusses the areas for improvement and concludes the report.

2 Related Work

The key factors that affects the overall performance of the applications which execute in containers and the recent studies which aim to optimize these factors are summarized in Table 2. A literature review related to the proposed objective is discussed based on different methodologies used.

2.1 Heuristic Algorithm

Zhang et al [7] consider container-VM-PM combination for placing containers and VMs on PMs efficiently by creating a fitness function model for appropriate selection. Hu, Laat and Zhao [8] proposes a container deployment algorithm that considers dependency awareness along with load balancing for multi resource requirements of the cluster providing a capacity limit. Considering cost and network as a primary factor for efficient container deployment, Zhou, Li and Wu [9] and Rodrigues et al [10] present an optimal container placement strategy with inter-container communication as the key parameter being considered for online container allocation in a cluster. The research presents a detailed insight on the algorithmic approach with economical benefits and computational efficiency.

2.2 ACO and PSO based methodology

C. Kaewkasi and K. Chuenmuneewong[11] uses Ant Colony System algorithm to find that it helps in improving the overall performance of applications that have the same configurations as their host. M. Lin et al [12] resolves the network transmission overhead caused in microservices by proposing an algorithm based on ACO for Multi objective Container-based Microservice Scheduling which is compared with [13]'s solution for optimized container allocation. M. K. Hussein, M. H. Mousa and M. A. Alqarni [14] utilize both VMs and PMs simultaneously by creating an optimum container placement method using Best Fit (BF), Max Fit (MF) and ACO combined with Best Fit (ACO-BF) algorithms for evaluating the resources used and schedule accordingly.

L. Li, J. Chen and W. Yan [15] overcome the under utilization of resources in container scheduling by applying Particle Swarm Optimization algorithm for optimal placement of containers. The ACO approach of [11] is compared here but it is shown to take longer time for computation and reaching to the solution. M. Adhikari and S.N. Srirama [16] use Accelerated Particle Swarm Optimization algorithm for an energy-efficient container scheduling strategy with focus on processing IOT and non IOT based tasks. With CO2 and temperature emission are important performance metrics considered, the proposed Energy Efficient Container Scheduling (EECS) strategy is shown to perform better in an energy-efficient way than the existing strategies.

2.3 Genetic Algorithm based methodology

Chen et al [17] focuses on generating an energy efficient container placement method with the aim to minimize the power consumption in Container-as-a-Service(CaaS) model using an Improved Genetic Algorithm. Dziurzanski and Indrusiak[18] creates a container allocation algorithm which measures the execution time of the task deployed in the container. C. Guerrero, I. Lera and C. Juiz [13] use NSGA-II and Greedy First Fit approach

for deploying applications with containers that are microservices focused. Mao et al.[19] provide a Genetic Algorithm approach to solve deadline constrained scheduling problem in a hybrid Cloud platform for an application with multiple tasks by choosing the hidden genes from the population set.

Aligning with the methodology of this research, Tan et al [20] have also integrated with Genetic Algorithm for allocation of resources in dual placement of containers in VMs and PMs like [7] to increase the efficiency of energy utilized in container deployment and execution. Applying Genetic Algorithm in a new perspective, Boukadi et al [21] provide a strategy for deploying Business Processes to containers by considering the QoS parameters and focusing on creating a cost optimized allocation method. For this they use Linear programming approach and Genetic Algorithm and compare the proposed architecture with First-Fit strategy.

2.4 Machine Learning approach

Cai et al [22] presents an automatic resource allocation method that aims to avoid resource contention in a container cluster by using a clustering method. S. Nanda and T. J. Hacker [23] use a deep learning approach for an adaptive resource aware container placement strategy. For energy-aware container scheduling, Rocha et al [24] presents Heats, a state-of-the-art tool that considers the dynamism of underlying hardware specifications of the container-based applications and their varied energy requirements in heterogeneous cluster environment. Nardelli et al [25] propose an Adaptive Container Deployment (ACD) model based on Integer Linear Programming problem which exploits the characteristics of fog computing and IoT environment and can scale resources present in different locations geographically.

2.5 Miscellaneous approaches

Li et al [26] and Chen, Zhou and Shi [27] propose a novel scheduling algorithm which uses Graph Coloring Problem based on Graph Theory and a stable matching problem to map the containers to the node according to the heterogeneous resource configurations with their future aim to implement cluster stability. Kaur et al [28] consider data locality and communication overheads with energy utilization as the primary metrics.Pongsakorn et al [29] proposes a container re-balancing technique by providing insights on LXC technology based containers and the isolation provided by them with re-balancing to adjust the resource accordingly and Lahmann et al [30] identifies the existing gaps in allocation of memory for containers and the utilization of memory showing a graphical representation of the unevenness in resource utilization in containerized environment. E. Casalicchio[31] presents a survey on container orchestration methodologies.

2.6 Other approaches:

Li et al [32] proposes an adaptive load balancing algorithm for Nginx containers which run on their own scheduling algorithm and Hu et al [33] present an Enhanced Container Scheduler for optimal container scheduling. Aligning with the objective of this study, Bacou et al [34] presents a solution in Kubernetes to present optimized performance. Hanafy et al [35], A.Chung, J.W. Park and G.R. Ganger[36] and Wan et al [37] present a cost-optimum solution for container deployment of applications using microservices.

Liu et al [6] propose a container scheduling algorithm known as MultiOpt that aims to improve the overall performance of resource scheduling in Docker Containers.Table 1 summarizes the problems with these algorithms as discussed by the authors. Availability is focused by Alahmad et al [38] in their proposed placement strategy. E. Casalicchio [39] provides a state-of-the-art methodology for autoscaling of CPU intensive workloads in containerized applications.L. Lv et al[40] addresses the container assignment problem by reducing the communication overhead and Y. Mao et al[41] give a Resource aware placement strategy for containers in a heterogeneous cluster.

Methodology	***Reference of Recent Work***	***KPI***	***Advantages***	***Gaps***
ACO	C. Kaewkasi and K. Chuenmuneewong	Resource Utilization, Interconnectivity of containers/nodes	Improves performance by optimizing resource utilization and network overhead.	Dynamic workloads and resource parameters are not considered.
PSO	Li, J. Chen and W. Yan	Resource Utilization	Dynamic tasks considered.	Only energy parameters like CO2 and Temperature considered.
Genetic Algorithm	Guerrero et al	Resource Utilization;Interconnectivity of containers/nodes;Cost;Latency;Energy	All key performance metrics considered.	**Balanced Cluster and Dynamic placing of containers not considered**.
Machine Learning	Cai et al	Latency;Cost;Energy	Minimizes resource wastage.	Fails to consider all the KPIs.
Heuristic Algorithm	Hu laat and zhao	Cost;Interconnectivity of containers/nodes	Considers both VM and PM efficiency.	Fewer details provided regarding capacity and resource parameters.
Self proposed Algorithms	Tihfon et al	Cost;Latency;Network;Resource Utilization	All key performance metrics considered.	No information provided on workload and containers used, **heterogeneity and balance of cluster not considered**.
Miscellaneous	Hanafy et al	Energy;Cost;Latency	Detailed information on containers and cluster.	Limited performance parameters considered, lack of cluster stability.

Table 2: shows the key performance indicators of applications execution in Cloud environment by considering container placement in VMs or task scheduling in Cloud platforms and summarizes the recent works based on their focus on each of these KPIs

3 Methodology

The methodology for meeting the aim of the research are as follows:

3.1 Steps followed

For carrying out the research, a detailed evaluation of existing approaches was made with their pros and cons. The initial approach chosen was Amazon ECS where, using Blox Methodology [42], the existing scheduler can be extended to create a customized scheduler where the aim was to implement the Genetic Algorithm logic that will take as input the computing statistics generated by the existing scheduler and propose a better fitting resource value for deploying of containers on the existing nodes. This approach was not proceeded further with as there was a limitation of AWS budget usage.

Another approach was to implement the same using Kubernetes as Kubernetes scheduler follows bin-pack algorithm which is the same algorithm as Amazon ECS's[43]. Figure 1 shows a high level comparison of Kubernetes and ECS. Minikube allows creation of

	Similarity	Dissimilarity
Kubernetes	1. Container placement algorithm is Bin pack. 2. Can work on a cluster of compute instances as well as AWS EC2. 3. Price model depends on the underlying compute instances and their usage. 4. Provides CLI and Dashboard	1. High Availability needs to be self-maintained. 2. Loosely coupled as it allows to use various combinations of containers, load balancers, network models and volumes. 3. No such limitation from Kubernetes. 4. Applications defined in YAML of pods and can be scaled manually or automatically.
Amazon ECS	1. Container placement algorithm is Bin pack, Spread, Random. 2. Can work on a cluster of compute instances (EC2). 3. Price model depends on the underlying compute instances and their usage. 4. Provides CLI and Dashboard	1. ECS maintains High Availability by distributing containers in different Availability Zones. 2. ECS is tightly coupled as it requires to integrate with Amazon web services only. 3. Cannot have multiple containers exposing same port on same node. 4. Applications are defined in task definitions and can be scaled manually.

Figure 1: Shows difference and similarity between Kubernetes and Amazon ECS [44]

cluster on Windows, Linux or MacOS using minikube[45]. Deploying kubernetes cluster with minikube enabled the creation of single node with multiple pods which was not helpful for evaluation purposes. So, the next approach was to use virtual machine provided by VMWare to implement a multi-node cluster of Kubernetes.

3.2 Preparation of Sample data

For preparing the sample data, the kube-scheduler was deployed that runs using bin pack algorithm along with an nginx application, an application for running Prometheus. A custom scheduler was deployed in the cluster with default resource values and later changed according to the results generated by Genetic Algorithm by adding the resource limit values for evaluating the new utilization rates of CPU and memory.

3.3 Measurements and Calculations performed

The cluster is allotted heterogeneous resource values where the master node is given higher resources than the worker node. The reason behind this was to notice the variation in the application's resource utilization with the allocated resource values. The master node is allocated 2 CPU processors and worker node is allocated 1 CPU processor. In Kubernetes, 1 CPU is equivalent to 1000m where 'm' is 'millicpu'. This conversion is performed by Kubernetes API as shown in Table 3. So, the master node was provided resource equivalent of 2000m CPU and worker node was provided with 1000m CPU resource equivalent. For memory, the measurement unit followed is Bytes. In this scenario, master node was allocated 4GB and worker node was allocated 2GB of memory.

Resource	***Unit***	***Kubernetes Unit***
CPU	1 CPU	1000m(millicpu)
Memory	1GB	976562.5 Ki(Kibibyte) or 953.6(Mebibyte)

Table 3: Unit conversion table followed by Kubernetes [46]

Kubernetes gives a utilization percentage value by considering the requested CPU and memory by the nodes(Node), and i being the number of nodes, over the total allocated resources. For calculating the utilization values, the formula used in this study:

$$[CPU_{\text{used}}(Node_{\text{i}}) + Memory_{\text{used}}(Node_{\text{i}})]/TotalClusterCapacity \tag{1}$$

A capacity threshold parameter is introduced here for limiting the total capacity by setting a threshold value which should not exceed the total capacity:

$$Capacity_{\text{thresh}} < [CPU(Node_{\text{j}}) + Memory(Node_{\text{j}})] | \forall Node_{\text{j}} \tag{2}$$

This threshold value for Capacity usage is calculated by reserving a percentage value of the cluster capacity and fitness test is performed based on this.

For calculating the average cluster utilization rate($\text{Average}_{\text{cluster}}$), total requested resources is considered over total allocated resources for all the deployed container types. The difference(Diff_{i}) between the values of individual container's resource utilization($\text{R}_{\text{C}_{\text{i}}}$) is represented by:

$$Diff_{\text{i}} = Average_{\text{cluster}} - R_{\text{C}_{\text{i}}} \tag{3}$$

Diff_{i} will give a set of values which if plotted in a graphical representation will show how close individual utilization values are from the mean. After Genetic Algorithm presents a best fit value based on the fitness function where capacity threshold parameter is checked, the new values are deployed to calculate the new individual utilization rates as well as the total cluster utilization for benchmarking purpose. Although Kubernetes by itself ensures that the total requested resources is not exceeding the capacity, the utilization rates are calculated by considering the capacity threshold parameter for analysis purposes.

3.4 Statistical Techniques - Best Linear Fit

For estimating whether the outcome of genetic algorithm will enable an establishment of a balanced cluster, a best fit line is plotted from the values generated by the Genetic Algorithm. In this case, two plots are made for CPU usage values and memory usage values for individual containers and the deviation of their individual resource utilization rates from the average cluster utilization. PyCharm is used to plot the given values in python using the matplotlib, statistics and numpy libraries. The aim is to find how the individual resource utilization values vary from achieving a balanced cluster by analyzing how deviated the values are from the mean.

4 Design Specification

Kubernetes is an open source platform developed by Google for container orchestration and management of containerized workloads. The pluggable nature of Kubernetes allows it to automate diverse workloads running with varied configurations and expedites the entire execution process of applications. Designed specially for running containerized applications in production environment, Kubernetes help in ensuring SLA is met with no downtime by maintaining high availability. Table 4 shows a detail description of features of Kubernetes and the gaps in it.

Features	***Gaps***
Personalized IP Address of Containers and Load Balancing.	Does not follow CI/CD for applications.
Pluggable storage system.	There is no provision for in-built application-specific services like SQL, Spark, caching etc.
Automated and controlled container addition, removal or resource deployment.	Does not impose specific logging and monitoring feature, but provides external methods for log tracking.
Bin packing of containers to nodes according to resources provided.	Does not follow a specific configuration language rather runs on declarative APIs.
Container failover, frequent health check and container readiness, popularly known as Kubernetes Self-healing.	There is no comprehensive database and messaging system provided.
Ease of secret configuration and password management.	Random allocation which cannot be controlled

Table 4: shows the what can be done and what cannot be done in Kubernetes.

4.1 Existing Architecture - Kubernetes

Kubernetes enables cluster deployment which is a collection of different machines. These machines form nodes in Kubernetes that allow running of applications in containers.A

cluster generally has a master and a worker node. The application that has to be hosted in the containers are held in the pods which are hosted by the worker nodes. The master node is responsible for managing the cluster nodes and pods deployed in it. Figure 2 shows the architecture overview of Kubernetes with its components.

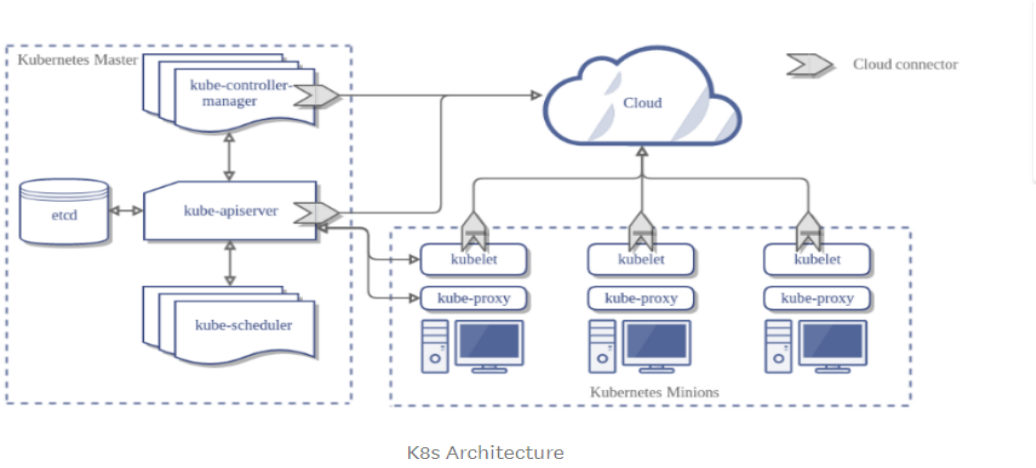


Figure 2: Overview of Kubernetes Architecture [47]

The components of Kubernetes are divided into sets of master components that helps in controlling the cluster and detecting any cluster event triggers to respond to them. They are comprised of **kube-apiserver**, **etcd**, **kube-controller and cloud-controller manager** and **kube-scheduler**. The node components help in providing an environment for running and managing the deployed pods. Node Componenets comprises **Kubelet** and **Kube-proxy** [47],[48].

4.1.1 Kube-apiserver and etcd

The Kube-apiserver exposes Kubernetes API that validates and configures data for pods,service, controllers and hosts. It forms the point of interaction for other components in the share state of cluster. The kube-apiserver is also responsible for scaling of instances horizontally.

etcd provides a storage for critical data in a distributed mode. The simplicity it provides by using key value pairs for data storage is what makes etcd easy to integrate with and popular to use.

4.1.2 kube-controller and cloud-controller manager

The kube-controller manager and cloud-controller manager are responsible for running controllers that perform the task of monitoring the cluster state and ensuring the desire state is reached by each controller. Kube-controller runs all controllers in the master and cloud-controller runs controllers that are integrating with the cloud providers. This enables the running of cloud vendor specific functions exclusive of Kubernetes functions.

4.1.3 kube-scheduler

The kube-scheduler is responsible for assigning nodes to pods by watching the state of newly created or unscheduled pods. The scheduler ensures that the maximum capacity of nodes is not exceeded by the resource requests and refrains from placing a pod to a node

if the check for capacity is not met to ensure there is no resource shortage for future tasks. Kubernetes scheduler follows bin-pack algorithm internally for placing pods in nodes.

4.1.4 Kubelet

Kubelets are agents that provides a health check of all Kubernetes containers running in the pods deployed. They are present in every node.

4.1.5 Kube-proxy

Kube-proxy is an integral part of Kubernetes architecture as it provides a way to implement the application running to be exposed as a network service. It forms a proxy network as it runs on every node in the cluster and maintains the network routing rules to allow inter cluster networking.

4.2 Implemented Architecture

Aligned with the above architecture, Kubernetes allows the use of multiple schedulers as the application requirements maybe diverse and the existing algorithm of kubernetes might not be applicable for all scenarios. Taking advantage of this feature, a new scheduler is deployed on top of the Kube-scheduler and assigned to the pods so that the new scheduler will be considered for container allocation instead of default scheduler. Figure 3 shows the implemented architecture. The scheduler to be deployed runs in a container image in the cluster. The values are given as input to genetic algorithm that gives best fit values back to the scheduler[49]. If the scheduler needs to be used for pods scheduling, the deployment yaml file for pods needs to be updated with the name of the custom scheduler [50]. Figure 4 shows the yaml file for pods with custom scheduler name assigned to the cluster.

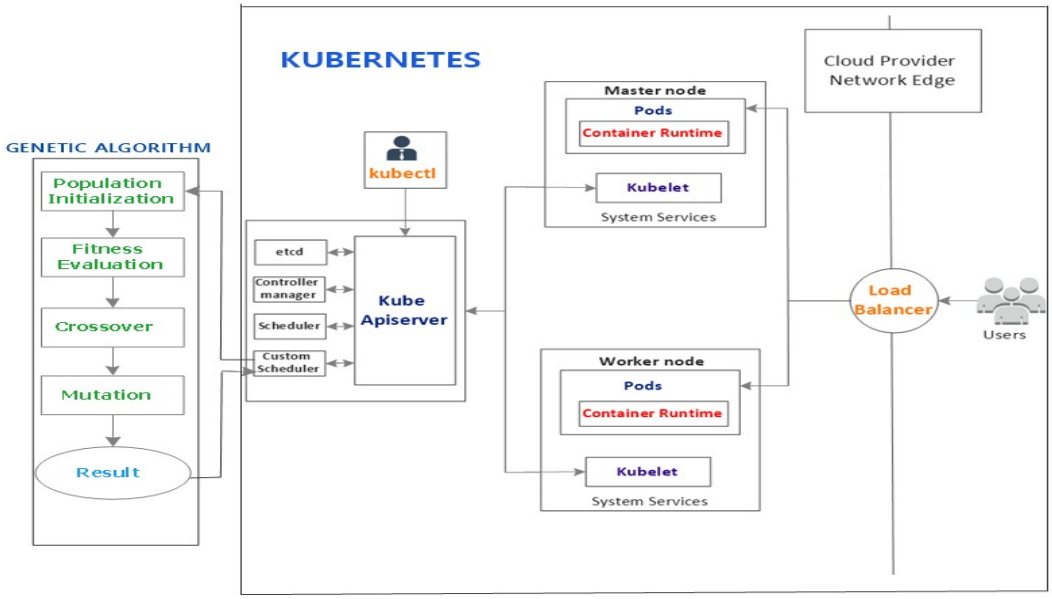


Figure 3: Kubernetes architecture integrated with Genetic Algorithm

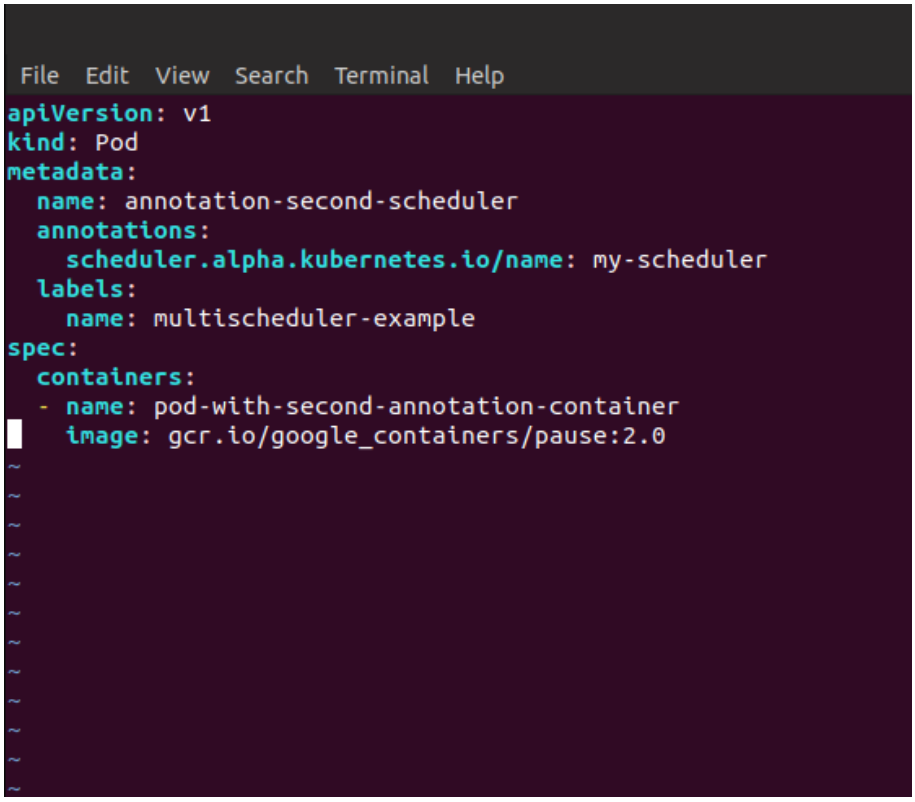


Figure 4: Pod Yaml file with a custom scheduler assigned

5 Implementation

5.1 Materials and Equipment

The implementation was carried out in Windows 10 Home version with configuration as Intel(R) Core(TM) i5-8250U CPU @ 1.60Hz 1.80Hz with 8GB RAM and 64-bit Operating System. VMWare Workstation Pro 12.5.2 version was used with Ubuntu 18.04 version installed where two VMs were created as master node and worker node [51]. To create a heterogeneous cluster in Kubernetes, both the nodes were allotted different resource configurations. Docker version of 19.03.5 was installed to run on Kubernetes. For running Genetic Algorithm, Python 3.7 version was used in PyCharm.

5.2 Applications deployed

Nginx is the best example of container deployment as it is the most popular web server used in containers [52]. An application can be executed by creating a Kubernetes Deployment file in YAML format where the details of an application are provided like Selector to define the connection between pods and deployment, replicas to state the number of copies of application to create, container name and image, etc, for example, a YAML file describing a Deployment that runs the nginx:1.7.9 Docker image or a Redis Image. Apart from nginx application, two simple applications are deployed(Figure 5). One is for allotting best price to the running nodes in the cluster by considering the resource used by the scheduled pods. Another application is deployed for testing Prometheus,which is an open source application that is used for monitoring the cluster metrics. It provides graphical representation of the resource utilization values according to the time the cluster was up

and running [53].

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	nginx-deployment-76bf4969df-6khrd	1/1	Running	0	3h37m
default	nginx-deployment-76bf4969df-mzfsz	1/1	Running	0	3h37m
default	nginx-deployment-76bf4969df-rt59b	1/1	Running	0	3h37m
default	nginx-deployment-8ddd7c545-sdvgp	0/1	Pending	0	104m
default	prometheus-8596b54698-2mlnm	1/1	Running	0	9h
default	scheduler-67dbd6755f-9klgb	2/2	Running	0	104m

Figure 5: shows running pods for deployed applications in cluster

5.3 Kubernetes Cluster Creation and Networking

5.3.1 Multinode Cluster creation

Kubectl is used to run commands for Kubernetes cluster in CLI. In this case, a two node cluster is created where one is the master node and another is the worker node (Figure 7). This is created by using "kubeadm join" command in Kubernetes that helps in the initialization and simultaneously adding the worker node in the cluster [54]. To enable a secure bidirectional communication of nodes in the cluster, a discovery token is used while initializing the other worker nodes. This is a root certificate authority which is given by the Kubernetes Control Plane in the form of a shared token used for connection verification as shown in figure 6.

```
bhavnat@slave-node:~$ sudo kubeadm join 192.168.159.132:6443 --token 43cp2i.dadwbog9pykv22ul \
>     --discovery-token-ca-cert-hash sha256:df16d6ce3a73c29df736256c72ef00183f1c330e7c05592608afa9149c12e821
[sudo] password for bhavnat:
[preflight] Running pre-flight checks
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.16" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Figure 6: shows worker node is joined to the master node running in a different system

```
bhavna@master-node:~$ kubectl get nodes
NAME          STATUS   ROLES    AGE   VERSION
master-node   Ready    master   52m   v1.16.2
slave-node    Ready    <none>   12m   v1.16.2
```

Figure 7: shows running nodes from the master node

5.3.2 Kubernetes networking

As Kubernetes has to ensure that the applications deployed are not allocated the same IP address as all the machines in the cluster are in a shared state, it assigns a unique IP to all the pods to facilitate networking in the cluster. Flannel is one such enabler for deploying a networking model in a multi-node cluster environment in Kubernetes and used in the experiment as it enables a layer 3 IPV4 network between the nodes and assigns a subnet to each node from a large pool of pre-configured address space. It interacts with the Kubernetes API or etcd for networking tasks like storing the configuration details or distributing the subnets, etc. The subnets are leased to the nodes so that they can be re-used for a different cluster. Instead of controlling the inter-container networking, Flannel manages the traffic flow in between the hosts [55].

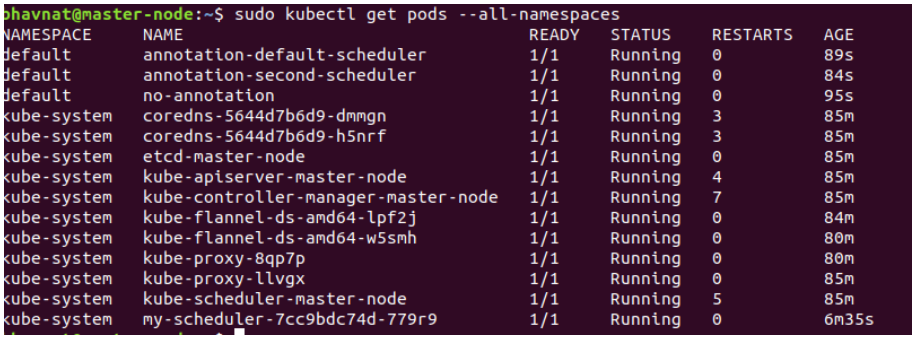


Figure 8: shows flannel and coredns specific pods running in the cluster

5.4 Genetic Algorithm

The scheduler yaml takes into consideration the new generated CPU and Memory values given by Genetic Algorithm which is executed in Python and considers the capacity threshold as a fitness check value while calculating the best fit resource metrics. Figure 10 shows the step-by-step execution process of Genetic Algorithm.

The selection process is done randomly with mutation probability defined as 50% of the first generation and then slowly decreases with each proceeding generation to reach a constant. A mutation index value is chosen by checking the mutation probability value against a randomly generated number. If it is greater than the number, then crossover is performed otherwise it is not proceeded further (figure 9). For the crossover procedure, a uniform crossover is used with probability value of 0.5 to split the existing population to create new off-springs.

```
if mutate_prob > np.random.rand():
    mutate_index = np.random.randint(len(self.numbers) - 1)
    self.numbers[mutate_index] = np.random.randint(101)
```

Figure 9: shows the mutation probability is checked to perform crossover

The fitness function in Genetic Algorithm checks the capacity threshold value, which reserves 10% of total allocated resources, and it increases the resource requested value by 0.5 times. It keeps evolving a new population by simultaneously performing a fitness check and presenting the best value in the current population and making it as the new criteria for the next generation population. The objective is to see whether genetically evolved capacity aware population affects the utilization level and performance of the container running cluster. This process continues depending on the number of iterations given.

6 Evaluation

On deploying the applications, the CPU request initially was 850m in master node and 150m in worker node which gives 42% utilization and 20% utilization of CPU resource respectively. The memory requested by master node is 190Mi which is 4% utilization in comparison to worker node Memory request value of 50Mi leading to 2% of memory utilization only as shown in Figure 11. Taking these resource request values as input by

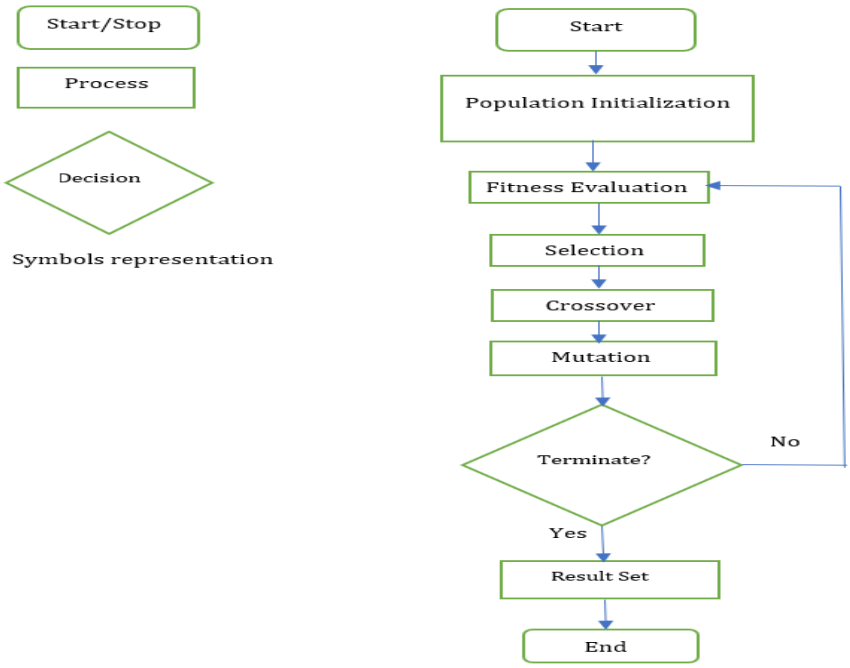
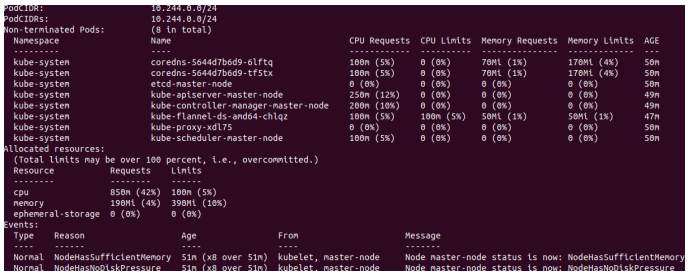
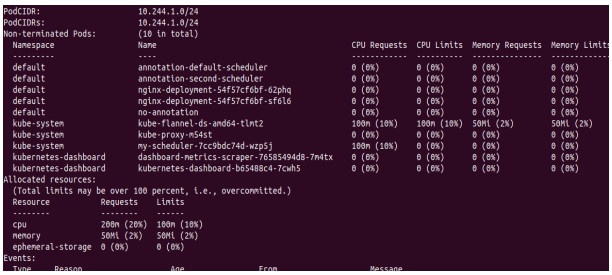


Figure 10: An overview of Genetic Algorithm



(a) master node CPU and memory utilization



(b) worker node CPU and memory utilization

Figure 11: Shows the resource utilization values for master and worker nodes as given by Kubernetes

Genetic algorithm, a fitness function is calculated where the capacity threshold value is reserved at 10% of total allocated resources. This value has been chosen by deciding the maximum value to be reserved and can vary for experimental purposes.

6.1 Resource Utilization

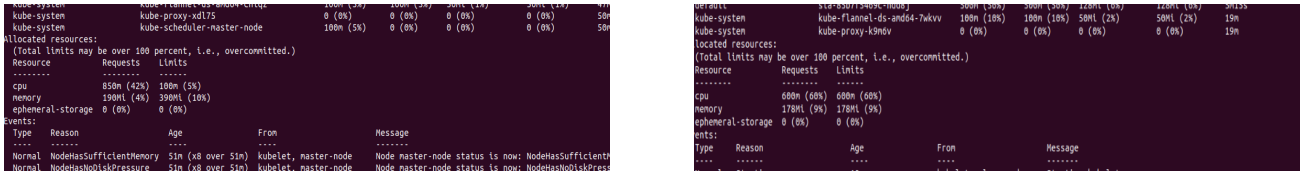
After deploying the CPU and Memory best values given by Genetic Algorithm (Figure 17) in Kubernetes, there are three cases studied:

6.1.1 Case Study 1: Using Genetic Algorithm values

On using the Genetically evolved values for the application resource request as in figure 12(a), the new utilization result is observed as shown in Figure 13 where the master node increases the utilization level on worker node and increases the limit keeping its own utilization level unchanged.

(a) (b)

Figure 12: Shows deploying application with new resource values



(a) master node CPU and memory utilization (b) worker node new CPU and memory utilization

Figure 13: Shows the updated resource utilization values given by Kubernetes after using Genetically evolved values

6.1.2 Case Study 2: Using Genetic Algorithm resource values with higher limits

When the values given by genetic algorithm are specified in the yaml file in Figure 12(b) along with higher limits specified, the new resource utilization level observed is shown in figure 14 where the worker node's limit is increased more than 100% to increase the utilization of CPU and memory in order to service the requested demands. This is because, when the master node evaluates the resource requests given in the container yaml file of the deployed application, it checks the total available capacity and rearranges the utilization value of available worker nodes first to serve the new resource requested by the container [56]. The calculation for the available capacity is done by considering the sum of the allocatable resource in the entire cluster. From the utilization value generated, it can be assumed that master node ensures the worker node is maximum utilized till a peak value is reached and load is distributed to other nodes in the cluster.

6.1.3 Case Study 3: Resource request higher than individual node's capacity

When the request value is set higher than the entire cluster, the worker nodes fail to execute the application and show insufficient resource message as shown in figure 15.

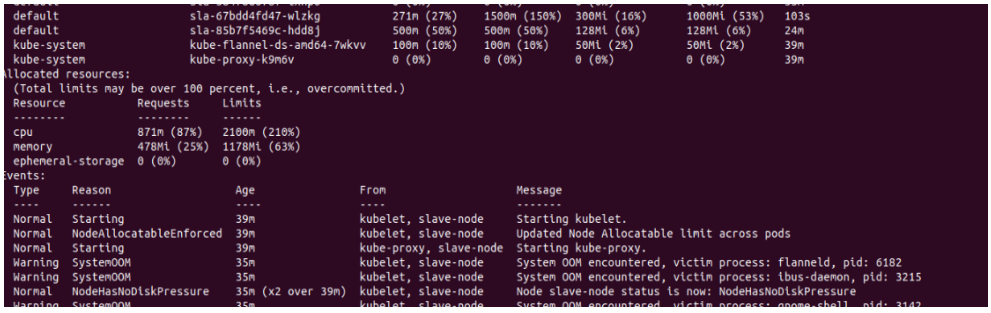


Figure 14: Shows the new utilization values after specifying higher limits with new resource values

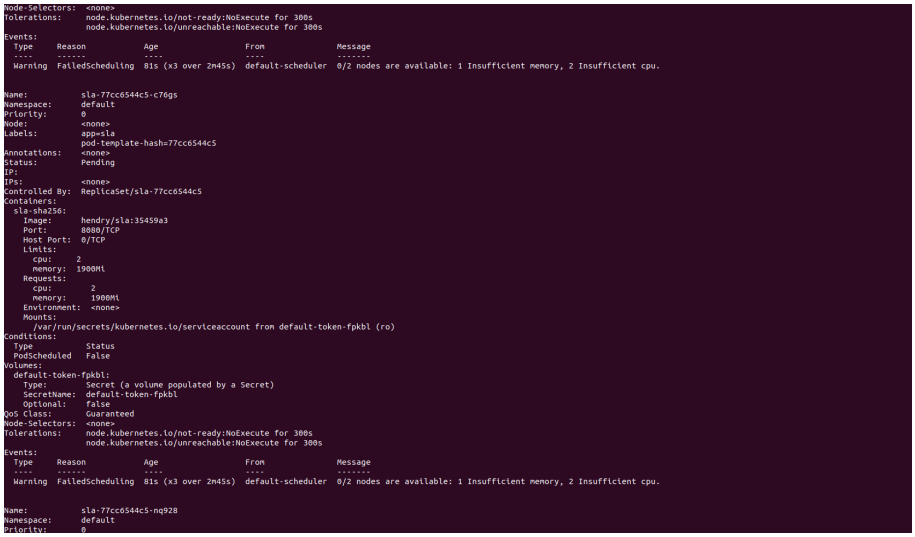
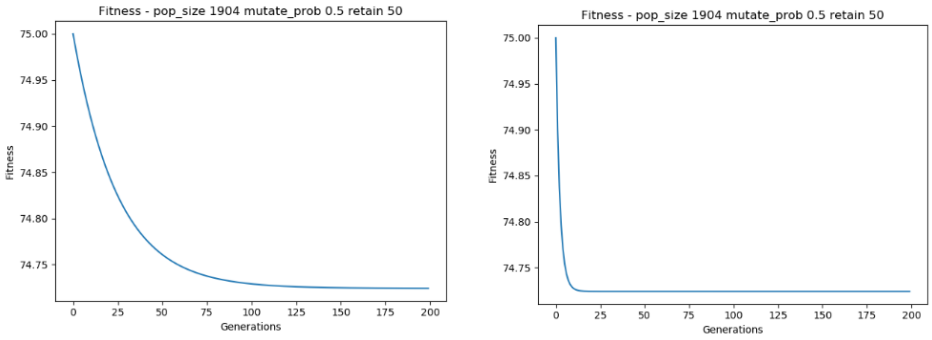


Figure 15: Shows insufficient resource when request crosses entire cluster limit



(a)2% capacity reservation (b) 20% capacity reservation

Figure 16: Shows the difference in the capacity reservation for memory and the time taking to provide a stable value

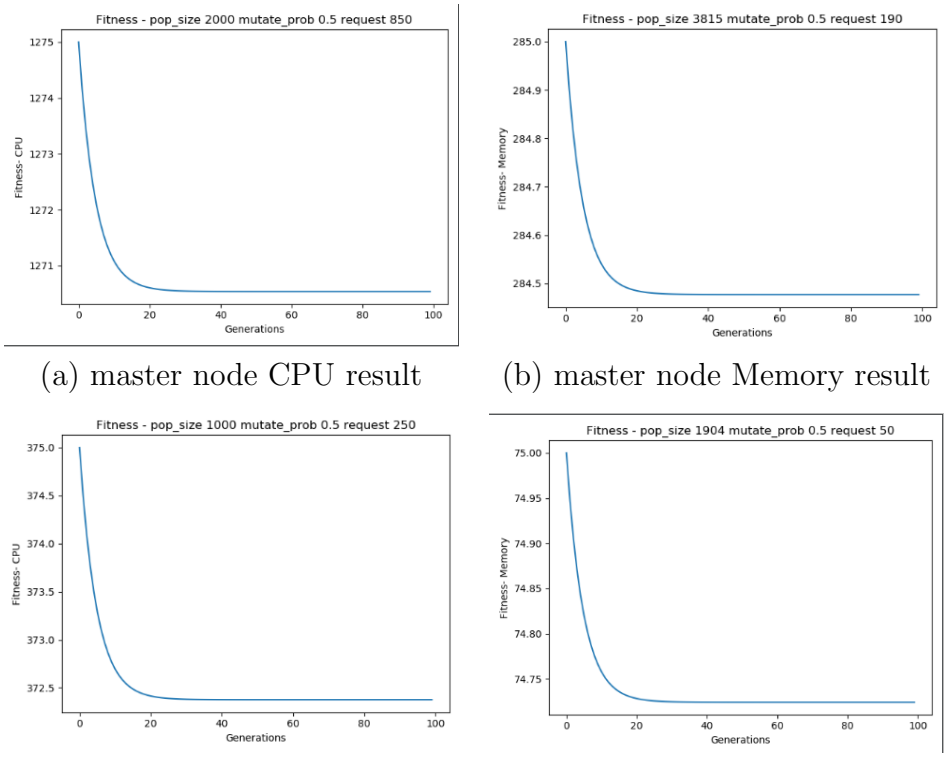
6.2 Genetic Algorithm

6.2.1 Case study 1: Varying the Capacity Threshold value

The fitness function in Genetic Algorithm is calculated with the capacity threshold value reserved at 10% of total allocated resources. On evaluation, it is seen that the lower the capacity threshold percentage value is considered, for example 2%, a stable value is reached later in the evolution generation, whereas the higher the capacity threshold percentage value,20% in this case, the earlier the stable value is reached. Figure 16 shows this variation of values through graphical representation presented by GA.

6.2.2 Case study 2: Varying the Number of Generations parameter

Figure 17 shows the results generated by running Genetic Algorithm for 100 generations and 200 generations with CPU and memory value of 1271m and 285Mi for master node and 373m and 75Mi for worker node. The reason behind executing it with different generations was to evaluate the point where it finds the best fit value of CPU and memory. While observing, it was noticed that by the time a new population is evolved for 80th



(a) master node CPU result (b) master node Memory result

(c) worker node CPU result (d) worker node Memory result

Figure 17: Genetic algorithm outcome for CPU and Memory best value for master and worker nodes(100 and 200 generations).

generation, a stable value has been developed.

6.2.3 Case study 3: Request value greater than the Total Capacity

In the Genetic Algorithm implementation, the "request" parameter takes in the current requested value of resource and "pop_size" parameter takes the total allocated capacity for a node. If the "request" parameter value is greater than the "pop_size" parameter, the Genetic Algorithm code performs a comparison where it handles this case and performs a fitness check by updating the total value to give fit results which are lower than the total allocated capacity. An output is shown, for example, where the requested value is 2048m but the total capacity is 2000m in Figure 18.

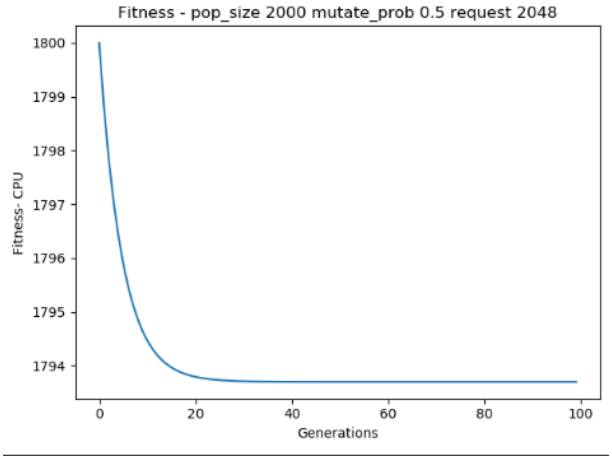
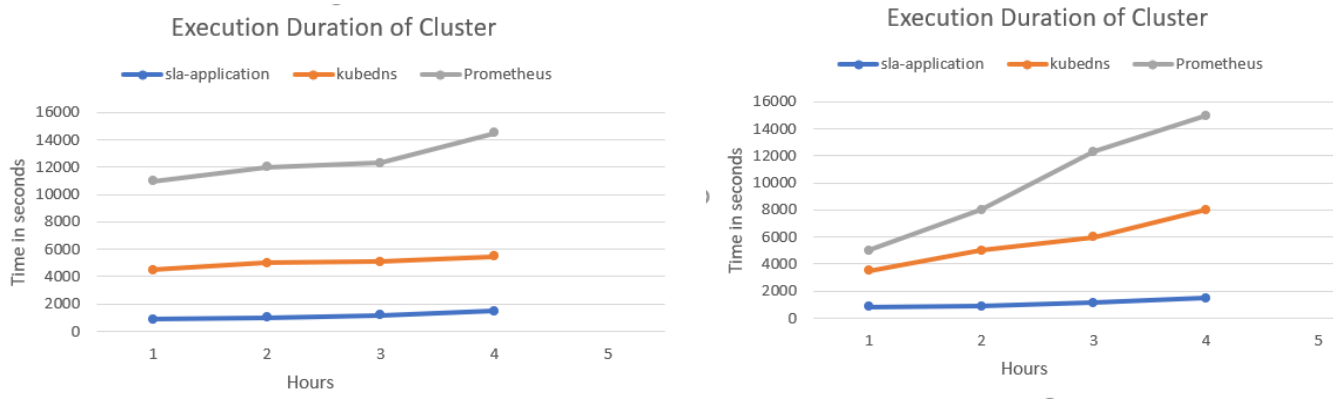


Figure 18: Shows the fit value given by Genetic Algorithm after specifying request value higher than total capacity



(a)Initial execution duration

(b) Final execution duration

Figure 19: Shows the duration of execution in seconds for every container

6.3 Performance Evaluation

Figure 19 shows the execution duration of all the containers running in kubernetes as given by Prometheus showing the total seconds of execution over specific time intervals. For the running application containers, there is a slight change in the execution time but Prometheus shows a significant change, depicting the total execution time is lowered by nearly 5000 seconds.

6.4 Cluster Balance

Genetic Algorithm is executed for several other values of allocatable resources generated by changing the capacity threshold value from 10% reservation to 50% reservation and using the master and worker node limit of usage values to analyze graphically whether a balanced cluster can be achieved or not. Figure 20 gives a graphical representation of the utilization levels plotted for the two nodes.

The plotted graph for CPU gives a slope of -0.000357 and an intercept of 0.721 where as for Memory plot, the slope generated is -0.00018 and intercept is 0.0895. The graphical

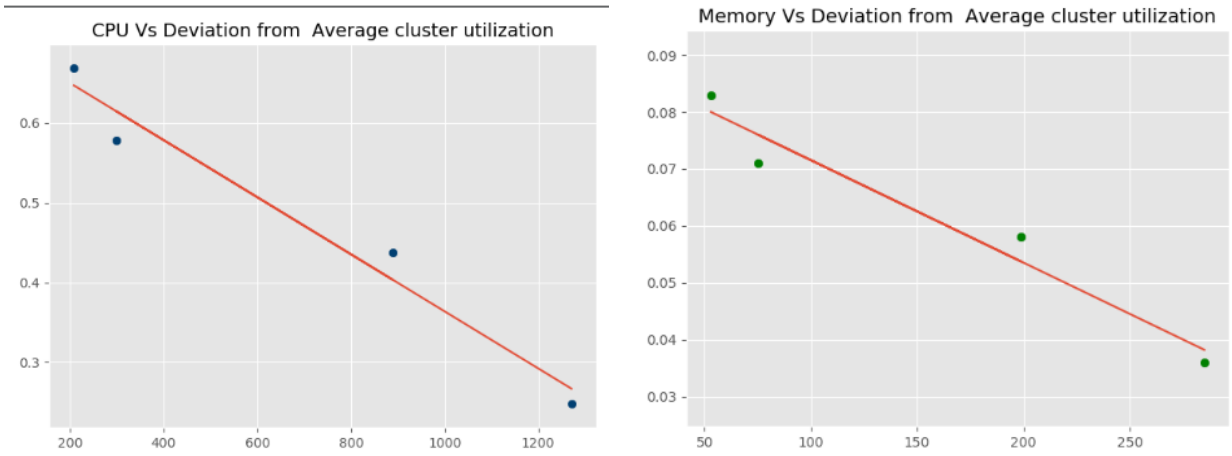


Figure 20: Shows the CPU and memory usage values and the deviation of individual utilization from the average cluster utilization

representation shows there is less deviation from the mean utilization of the entire cluster denoting a cluster which is nearly balanced but can be better.

6.5 Discussion

From the above evaluations performed considering different performance indicators of a container cluster, it can be seen that Genetic Algorithm reaches a stable resource value based on the given values by continuously checking the capacity aware fitness function. This value, when used to benchmark resource utilization and performance level, is seen to change the limits of the cluster, along with an increase in utilization level, to accommodate the new request (Figure 21), but if the application is less resource intensive, the worker and master node continues to execute as before. There is also a change in the total time duration of application execution but it is not significant and is an area which can be improved further. Although there is an increase observed in the utilization level values, these values depend on the cluster capacity and the demand generated by the application deployed. If the cluster was larger with enough capacity, the same resource values for the applications used might not have brought significant variation in resource utilization levels. For measuring whether the genetically evolved values can reach a balanced cluster, the deviation of individual resource utilization from the average value of the entire cluster is measured and it is seen to be nearly balanced but there is a scope for improved results to achieve a completely balanced cluster.

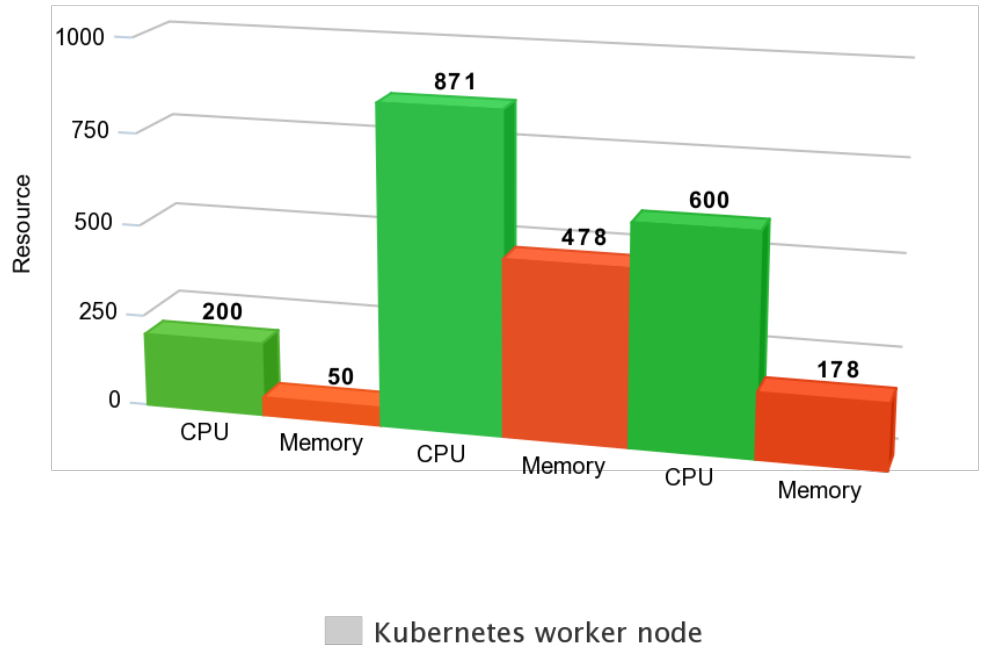


Figure 21: Shows change in Resource metrics for worker node

7 Conclusion and Future Work

The aim of this research was to enable a capacity aware container allocation in a way that the resource intensive applications deployed in the cluster can run without facing resource contention. This study deploys multiple applications in Kubernetes cluster to get the current resource requested values. The research presents the following:

- It proposes a new resource value given by Genetic Algorithm that continuously provides evolved outcomes.
- The values which pass the fitness check of capacity threshold value introduced in fitness function are proceeded for generating a new population of results.
- Using the fit generation of values, the containers are redeployed to measure any change in the utilization levels, only to find the cluster readjusting itself to accommodate applications with higher resource demands and gives a fairly better utilization results.
- There is not much variation in execution time which can be improved in the future work.
- A statistical measure is generated which shows how balanced is the cluster.

These results are subject to variation based on the system configuration used in case of VMs or cluster configuration if cloud is used. The utilization levels may not change if the application is not very demanding and enough resource exists.

Currently, genetic algorithm is used statically and there is much room for improvement in this area as a high resource-demanding application can be used in a cluster having more number of nodes. The future aim of this study is to use a larger cluster

of nodes for dynamic resource measures and simultaneous allocation of containers for various applications with an enhanced Genetic Algorithm.

References

[1] M. A. Rodriguez and R. Buyya, "Containers orchestration with cost-efficient auto-scaling in cloud computing environments," *arXiv preprint arXiv:1812.00300*, 2018.

[2] Docker, "What is a container?,*https://www.docker.com/resources/what-container*." [Online]. Available: https://www.docker.com/resources/what-container

[3] K. Hwang, *Cloud computing for machine learning and cognitive applications, Chapter 3.* MIT Press, 2017.

[4] N. G. Bachiega, P. S. Souza, S. M. Bruschi, and S. d. R. de Souza, "Container-based performance evaluation: A survey and challenges," in *2018 IEEE International Conference on Cloud Engineering (IC2E).* IEEE, 2018, pp. 398–403.

[5] "As discussed with Tom Maddox, head of territories, solution architect, amazon web services uk ltd. and Simon Thulbourn, specialist solution architect, aws emea,*Number of mail and on-call discussions: 3.*"

[6] B. Liu, P. Li, W. Lin, N. Shu, Y. Li, and V. Chang, "A new container scheduling algorithm based on multi-objective optimization," *Soft Computing*, vol. 22, no. 23, pp. 7741–7752, 2018.

[7] R. Zhang, A.-m. Zhong, B. Dong, F. Tian, and R. Li, "Container-vm-pm architecture: A novel architecture for docker container placement," in *International Conference on Cloud Computing.* Springer, 2018, pp. 128–140.

[8] Y. Hu, C. De Laat, Z. Zhao *et al.*, "Multi-objective container deployment on heterogeneous clusters," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 592–599.

[9] R. Zhou, Z. Li, and C. Wu, "An efficient online placement scheme for cloud container clusters," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1046–1058, 2019.

[10] L. R. Rodrigues, M. Pasin, O. C. Alves Jr, C. C. Miers, M. A. Pillon, P. Felber, and G. P. Koslovski, "Network-aware container scheduling in multi-tenant data center," *arXiv preprint arXiv:1909.07673*, 2019.

[11] C. Kaewkasi and K. Chuenmuneewong, "Improvement of container scheduling for docker using ant colony optimization," in *2017 9th international conference on knowledge and smart technology (KST).* IEEE, 2017, pp. 254–259.

[12] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud," *IEEE Access*, 2019.

[13] C. Guerrero, I. Lera, and C. Juiz, "Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications," *The Journal of Supercomputing*, vol. 74, no. 7, pp. 2956–2983, 2018.

[14] M. K. Hussein, M. H. Mousa, and M. A. Alqarni, "A placement architecture for a container as a service (caas) in a cloud environment," *Journal of Cloud Computing*, vol. 8, no. 1, p. 7, 2019.

[15] L. Li, J. Chen, and W. Yan, "A particle swarm optimization-based container scheduling algorithm of docker platform," in *Proceedings of the 4th International Conference on Communication and Information Processing*, ser. ICCIP '18. New York, NY, USA: ACM, 2018, pp. 12–17. [Online]. Available: http://doi.acm.org/10.1145/3290420.3290432

[16] M. Adhikari and S. N. Srirama, "Multi-objective accelerated particle swarm optimization with a container-based scheduling for internet-of-things in cloud environment," *Journal of Network and Computer Applications*, vol. 137, pp. 35–61, 2019.

[17] R. Zhang, Y. Chen, B. Dong, F. Tian, and Q. Zheng, "A genetic algorithm-based energy-efficient container placement strategy in caas," *IEEE Access*, 2019.

[18] P. Dziurzanski and L. S. Indrusiak, "Value-based allocation of docker containers," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2018, pp. 358–362.

[19] J. Mao, L. Sun, Y. Zhang, and J. Sun, "An improved genetic algorithm for solving bag-of-tasks scheduling problems with deadline constraints on hybrid clouds," in *2018 IEEE International Conference on Progress in Informatics and Computing (PIC)*. IEEE, 2018, pp. 305–310.

[20] B. Tan, H. Ma, and Y. Mei, "Novel genetic algorithm with dual chromosome representation for resource allocation in container-based clouds," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 452–456.

[21] K. Boukadi, R. Grati, M. Rekik, and H. Ben-Abdallah, "Business process outsourcing to cloud containers: How to find the optimal deployment?" *Future Generation Computer Systems*, vol. 97, pp. 397–408, 2019.

[22] L. Cai, Y. Qi, W. Wei, and J. Li, "Improving resource usages of containers through auto-tuning container resource parameters," *IEEE Access*, vol. 7, pp. 108 530–108 541, 2019.

[23] S. Nanda and T. J. Hacker, "Racc: Resource-aware container consolidation using a deep learning approach," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, ser. MLCS'18. New York, NY, USA: ACM, 2018, pp. 2:1–2:5. [Online]. Available: http://doi.acm.org/10.1145/3217871.3217876

[24] I. Rocha, C. Göttel, P. Felber, M. Pasin, R. Rouvoy, and V. Schiavoni, "Heats: Heterogeneity-and energy-aware task-based scheduling," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2019, pp. 400–405.

[25] M. Nardelli, V. Cardellini, and E. Casalicchio, "Multi-level elastic deployment of containerized applications in geo-distributed environments," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2018, pp. 1–8.

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

[27] F. Chen, X. Zhou, and C. Shi, "The container deployment strategy based on stable matching," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2019, pp. 215–221.

[28] K. Kaur, N. Kumar, S. Garg, and J. J. Rodrigues, "Enloc: data locality-aware energy-efficient scheduling scheme for cloud data centers," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[29] U. Pongsakorn, Y. Watashiba, K. Ichikawa, S. Date, H. Iida *et al.*, "Container rebalancing: Towards proactive linux containers placement optimization in a data center," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1. IEEE, 2017, pp. 788–795.

[30] G. Lahmann, T. McCann, and W. Lloyd, "Container memory allocation discrepancies: An investigation on memory utilization gaps for container-based application deployments," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 404–405.

[31] E. Casalicchio, "Container orchestration: A survey," in *Systems Modeling: Methodologies and Tools*. Springer, 2019, pp. 221–235.

[32] R. Li, Y. Li, and W. Li, "An integrated load-balancing scheduling algorithm for nginx-based web application clusters," in *Journal of Physics: Conference Series*, vol. 1060, no. 1. IOP Publishing, 2018, p. 012078.

[33] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "Concurrent container scheduling on heterogeneous clusters with multi-resource constraints," *Future Generation Computer Systems*, 2019.

[34] M. Bacou, A. Tchana, and D. Hagimont, "Your containers should be wysiwyg," in *2019 IEEE International Conference on Services Computing (SCC)*. IEEE, 2019, pp. 56–64.

[35] W. A. Hanafy, A. E. Mohamed, and S. A. Salem, "A new infrastructure elasticity control algorithm for containerized cloud," *IEEE Access*, vol. 7, pp. 39 731–39 741, 2019.

[36] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '18. New York, NY, USA: ACM, 2018, pp. 121–134. [Online]. Available: http://doi.acm.org/10.1145/3267809.3267819

[37] X. Wan, X. Guan, T. Wang, G. Bai, and B.-Y. Choi, "Application deployment using microservice and docker containers: Framework and optimization," *Journal of Network and Computer Applications*, vol. 119, pp. 97–109, 2018.

[38] Y. Alahmad, T. Daradkeh, and A. Agarwal, "Availability-aware container scheduler for application services in cloud," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2018, pp. 1–6.

[39] E. Casalicchio, "A study on performance measures for auto-scaling cpu-intensive containerized applications," *Cluster Computing*, pp. 1–12, 2019.

[40] L. Lv, Y. Zhang, Y. Li, K. Xu, D. Wang, W. Wang, M. Li, X. Cao, and Q. Liang, "Communication-aware container placement and reassignment in large-scale internet data centers," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 540–555, 2019.

[41] Y. Mao, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, "Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster," in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2017, pp. 1–8.

[42] Blox, "Blox,open source tools for building custom schedulers on amazon ECS *https://github.com/blox/blox*." [Online]. Available: https://github.com/blox/blox

[43] Kubernetes, "Production-grade container orchestration,*https://kubernetes.io/*." [Online]. Available: https://kubernetes.io/

[44] C. Wright, "Kubernetes vs amazon ecs,*https://platform9.com/blog/kubernetes-vs-ecs/*." [Online]. Available: https://kubernetes.io/docs/tasks/tools/install-minikube/

[45] Kubernetes, "Install minikube - kubernetes,*https://kubernetes.io/docs/tasks/tools/install-minikube/*." [Online]. Available: https://kubernetes.io/docs/tasks/tools/install-minikube/

[46] Kubernetes.io, "Managing compute resources for containers, *https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/*." [Online]. Available: https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

[47] Kubernetes, "Kubernetes components,*https://kubernetes.io/docs/concepts/overview/components/*. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/

[48] Kubernetes.io, "Cluster architecture *https://kubernetes.io/docs/concepts/architecture/nodes/*." [Online]. Available: https://kubernetes.io/docs/concepts/architecture/nodes/

[49] A. Gawanmeh, S. Parvin, and A. Alwadi, "A genetic algorithmic method for scheduling optimization in cloud computing services," *Arabian Journal for Science and Engineering*, vol. 43, no. 12, pp. 6709–6718, 2018.

[50] Kubernetes, "Configure multiple schedulers, *https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/*." [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/

[51] Ubuntu, "Install kubernetes on ubuntu *https://ubuntu.com/kubernetes/install*." [Online]. Available: https://ubuntu.com/kubernetes/install

[52] Nginx, "Nginx application platform *https://www.nginx.com/products/*." [Online]. Available: https://www.nginx.com/products/

[53] Prometheus, "From metrics to insight, *https://prometheus.io/*." [Online]. Available: https://prometheus.io/

[54] Kubernetes, "kubeadm join *https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-join/*." [Online]. Available: https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-join/

[55] CoreOS, "Flannel, *https://github.com/coreos/flannelflannel*." [Online]. Available: https://github.com/coreos/flannel#flannel

[56] Kubernetes, "Kubernetes tasks- assign cpu resources to containers and pods *https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/*." [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/

National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Bhavna Thakur
Student ID: x18145914

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Bhavna Thakur
Student ID:	x18145914
Programme:	MSc in Cloud Computing
Year:	2019
Module:	MSc Research Project
Supervisor:	Manuel Tova-Izquierdo
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	1715
Page Count:	13

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

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

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:	Bhavna Thakur
Date:	2nd February 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 sufficient to keep a copy on computer.	☐

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

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

Configuration Manual

Bhavna Thakur
x18145915

1 Introduction

1.1 Purpose of this document

A Configuration Manual is presented as a part of MSc Research Project requirements as given in the Project Handbook. The objective of this document is to give a detailed procedure for successful deployment of the Capacity Aware Container Placement in the cluster by providing the Software tools and settings.

1.2 Document Structure

Section	**Purpose**
General Information and Prerequisites	This section provides the Objective, Solution Summary, Architectural Requirements and Prerequisites.
Environment Development Requirements	This part provides a setp-by-step process the steps for setting the environment used for development along with solution update
Procedure for Solution Deployment	This section gives a step-by-step process for setting the environment of the proposed solution.
Deployment Validation	This section gives the steps required to check whether the deployment is successful or not.

2 General Information

2.1 Objective

The objective of the Capacity Aware Container Placement Strategy is to ensure there is some capacity available in the cluster to service dynamic resource requirements by applications and to measure if there is any improvement in the performance as well as a balanced cluster is reached by using genetically evolved best fit resource value that conforms with the fitness function where a capacity threshold value is used for performing a fitness check.

2.2 Solution Summary

The Capacity Aware Container Placement in Heterogeneous Cluster comprises a Kubernetes Cluster of Docker containers on which applications are deployed. Kubernetes runs

its own kube-scheduler for container allocation by checking the capacity limits of the nodes. On running the application containers, it gives values of resource utilization. These values are given to Genetic Algorithm, which is responsible for generating best fit values that pass the capacity threshold fitness check. Based on the output, the applications are redeployed in the cluster by specifying these values in resources template. The utilization levels are again measured to see if there is any change. Taking values of Genetic Algorithm, a check is performed to see if a balanced cluster can be reached. This is done by observing the deviation of the resource utilization from the mean resource utilization of the cluster.

2.3 Architectural Requirements

In this section the architecture required that forms the base is explained.

2.3.1 Virtual Machine or Cloud Platform

To host a multinode cluster, a Virtual Machine or a cloud platform is used where multiple instances form nodes that will host the containers or pods to execute the application. Here VMWare Workstation Pro 12.5.2 version is used with Ubuntu 18.04 version installed. While creating the Virtual Machine, here two VMs are created and allotted 2 CPU and 1 CPU each with 4GB and 2GB Memory.

2.3.2 Kubernetes

Kubernetes version 1.4.0 is installed on the Ubuntu system on both nodes. Kubernetes version can be checked by running the –version command in the terminal.

2.3.3 Docker

To run containers on the cluster, Docker Technology is used which enables running applications on containers by providing cgroups and namespaces by using OS level virtualization to enable running containers in isolation. Docker version 19.03.5 is installed on Ubuntu.

2.3.4 Prometheus

Prometheus is a monitoring tool deployed in the cluster that monitors and generates graphical representation of the running status of the applications along with performance evaluation.

2.3.5 Genetic Algorithm

Genetic Algorithm is executed on PyCharm 2019.2.5 Porfessional Edition is used with Python 3.7 version for implementing Genetic Algorithm.

2.4 Prerequisites

The user should know basic linux commands, Python programming and basic knowledge about Kubernetes. Along with this, the user should also have administrator access for performing the setup.

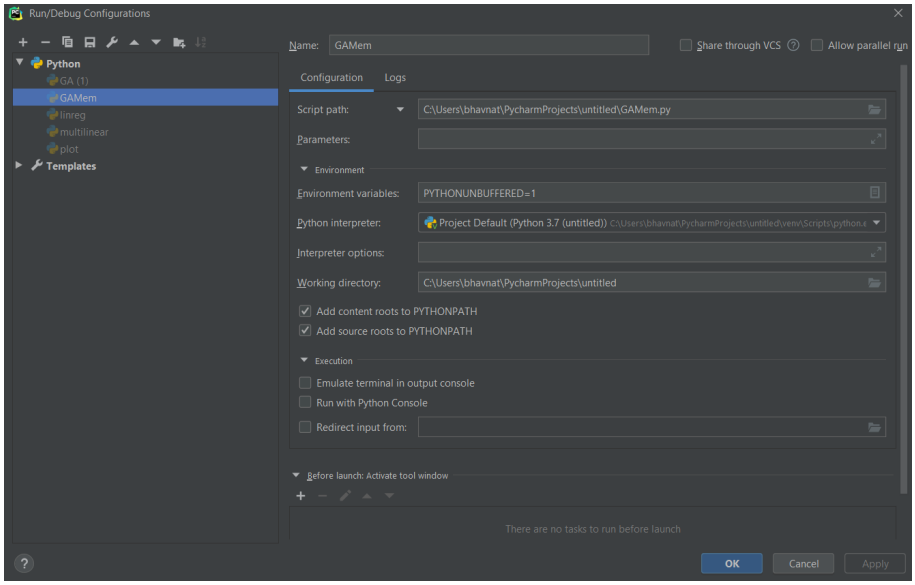


Figure 1: PyCharm Project Interpreter changes

3 Environment Development Requirements

3.1 Code Repository

Code is present in the zip file uploaded along with this manual.

3.2 Required Programming Languages

3.2.1 Ubuntu 18.04 or Linux Environment

Linux is used on Ubuntu 18.04 version for this project. Ubuntu can be installed and given as environment image during the Virtual Machine setup.

3.2.2 Python v3.7 and PyCharm

Install Python 3.7 and perform the setup of PyCharm for updating the python interpreter with the execution path of python where it is installed in the local system (Figure 1). Check whether the environment variables are updated automatically while setup. If not, then they need to be updated manually.

After installation of Python, the following packages need to be installed:

- numpy==1.17.4
- matplotlib==3.1.2
- pandas==0.25.3
- zipp==0.6.0
- py==1.8.0
- dataframe==0.2.1.3
- requests==2.22.0

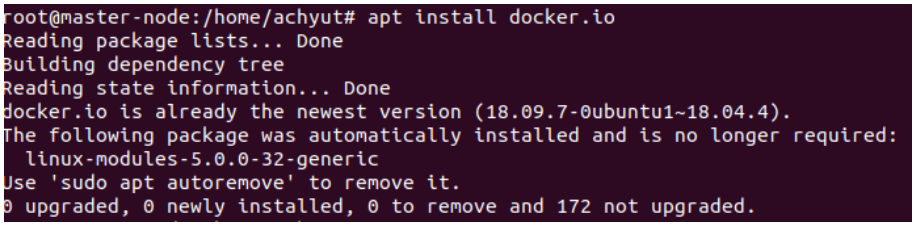


Figure 2: shows docker install command

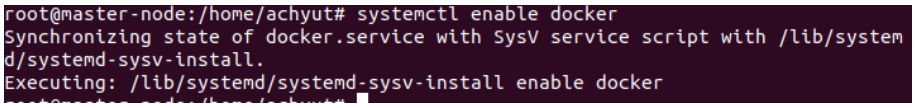


Figure 3: shows docker enable command

3.3 Docker Installation

- In the two VMs created, Docker environment needs to be installed on both. To install Docker, the command is: **sudo apt install docker.io** as in Figure 2.
- Once installed, check the version by running the command: **docker –version**.
- Enable docker on both the nodes, by running the command: **sudo systemctl enable docker** as shown in Figure 3

4 Kubernetes

4.1 Key Generation

First, a key needs to be generated for authentication and authorization purposes [1]. For generating a signed key of Kubernetes, the following command needs to be executed as shown in Figure 4:
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg — sudo apt-key add.

Install curl if needed by running **sudo apt install curl**

4.2 Xenial Kubernetes Repository

To add Xenial Repository, run the following command on both the nodes(Figure 5):
sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main".

4.3 Kubeadm installation

Next, Kubeadm needs to be installed which bootstraps and supports kubernetes cluster lifecycle and can run on different platforms. Run the command:

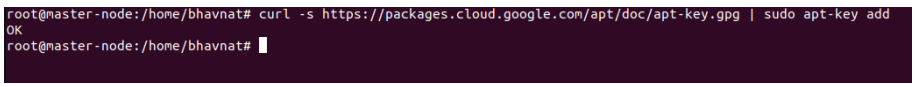


Figure 4: Kubernetes Key Generation

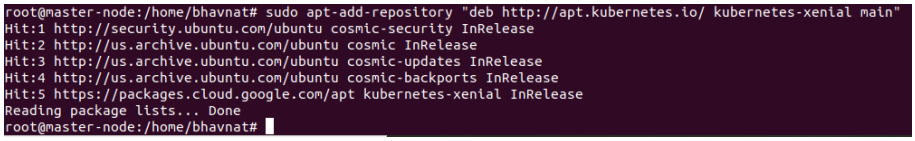


Figure 5: Adding Repository to Kubernetes

.

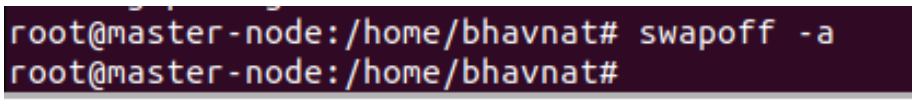


Figure 6: Swapoff

.

sudo apt install kubeadm.

Check the version of Kubeadm by running the command: **kubeadm version**.

4.4 Deploying Kubernetes

4.4.1 Swap Memory

Swap memory is the space of the physical memory that is used to extend the RAM of the system when the Physical Memory is full [2]. It is advisable to disable the swap area as it might decrease the performance of Kubernetes. To disable swap, type the command on both nodes in Figure 6:
sudo swapoff -a

4.4.2 Naming of hosts

To rename the host nodes to define which is master and which are worker nodes, type the command on the respective nodes:
sudo hostnamectl set-hostname master-node
sudo hostnamectl set-hostname worker-node

4.4.3 Initialize Kubernetes

For starting the Kubernetes cluster, the following command needs to run on master node(Figure 7):
sudo kubeadm init –pod-network-cidr=10.244.0.0/16. Install Kubeadm to run the above initialization command:
sudo apt install kubeadm
kubeadm version

For starting the cluster, run the following commands one by one on master node which is the output of initialize command as seen in Figure 8:
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

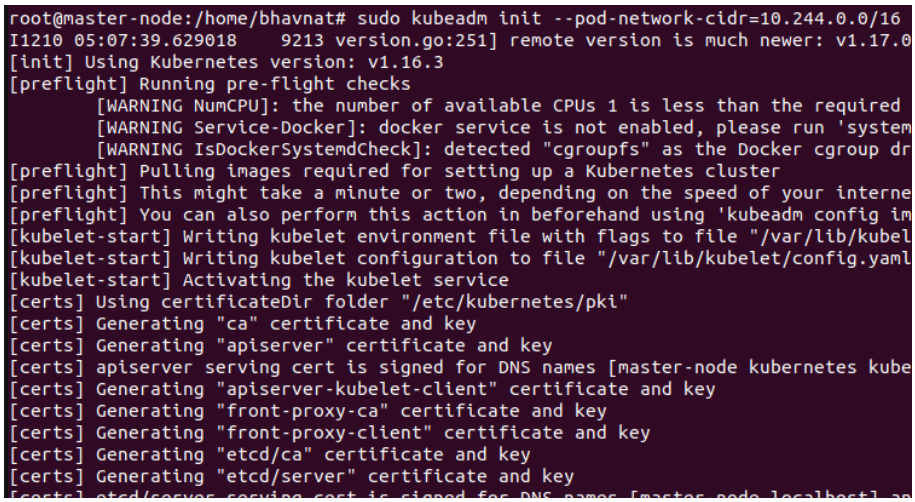


Figure 7: Kubernetes Initialization

.

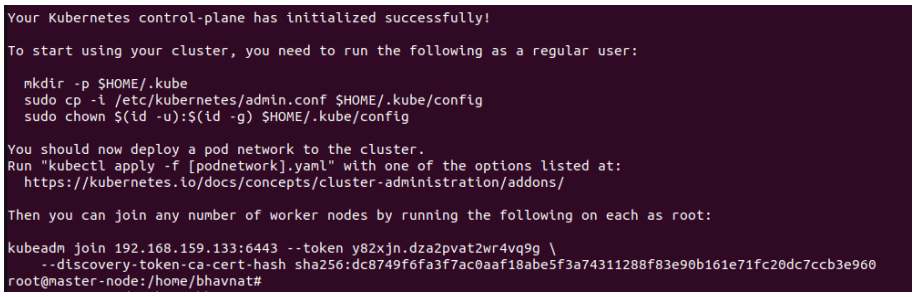


Figure 8: Output of Kubernetes Initialization

.

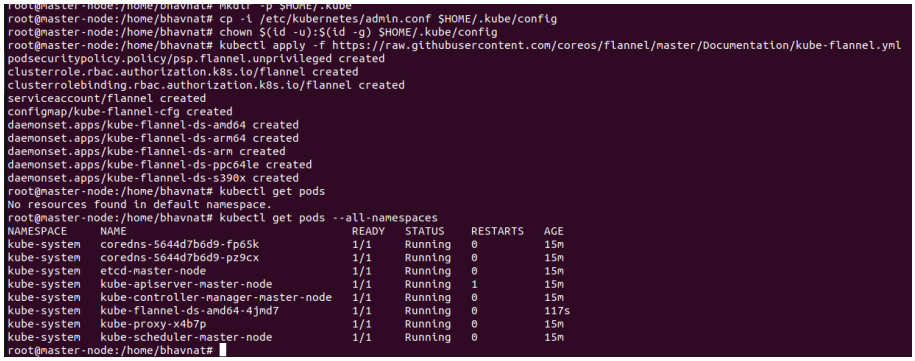


Figure 9: Flannel deployment and running pods

.

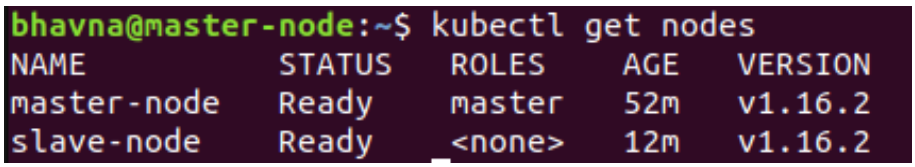


Figure 10: running nodes

.

4.4.4 Deploying Pod Network

For creating a pod network, flannel is deployed by using the command:
sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documen flannel.yml .
To check the running pods, run the command: **kubectl get pods –all-namespaces**.
Output will be seen as in Figure 19:

4.4.5 Joining nodes in cluster

Ensure the command produced in the initialization part is saved as it will be used to join nodes in a cluster. Run the command generated in Figure 8 in the worker nodes to join them in the cluster:
kubeadm join 192.168.159.133:6443 –token y82xjn.dza2pvat2wr4vq9g –discovery-token-ca-cert-hash sha256:dc8749f6fa3f7ac0aaf18abe5f3a74311288f83e90b161e71fc20dc7cc

4.4.6 Check status of cluster

To see whether the nodes are in Ready state and the running Pods, run the command:
sudo kubectl get nodes
Output is as in Figure 10:
sudo kubectl get pods

5 Deploying Applications

To create a custom scheduler or to deploy a new application, a Dockerfile needs to be generated that will hold all the application details as shown in the example in Figure 11.

```
FROM busybox
ADD ./_output/local/bin/linux/amd64/kube-scheduler /usr/local/bin/kube-scheduler
```

Figure 11: Shows the dockerfile creation for Kube-Scheduler [3]

A simple existing application is deployed that was created for running and testing Prometheus [4]. Figure 12 shows the deployment file used for running Prometheus in the cluster.

Also, an nginx application is deployed by running the following command:
kubectl apply -f https://k8s.io/examples/application/deployment.yaml

For deploying application through Yaml file in the cluster, run the command in the master node [5]:
kubectl create -f prometheus-deployment.yaml

This will give the following result if successful(Figure 13)

5.1 Kube Proxy

Kube-proxy provides a way to implement the application running to be exposed as a network service by forming a proxy network. To enable the deployment and running of deployed applications in the cluster run the command(Figure 14):
kubectl proxy

6 Validation

To validate the running architecture integration, first check the applications deployed are running in the cluster or not. To check this run the command:
kubectl get deployments.

If there are any services running, like prometheus or kube-metrics, its status can be checked by the following command:
kubectl get services.

To check whether the application are running in containers and specified types, check with the following command:
kubectl get pods.
Figure 15 gives an overview of deployments. Wait till the status for all the containers changes from "pending" to "running". Also, Kubernetes master node allocates the containers by checking the available capacity, if the application is heavy that requires more capacity than available, the container will remain in "pending" state and then move to "terminated" state eventually.

After confirming the cluster is up and running, run the following command to get the detailed description of the instances along with resource usage and the percentage of utilization:
kubectl describe nodes.

Figure 16 shows the command description given by Kubernetes and Figure 17 shows the output of the command for nodes.

```yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: default
---
```

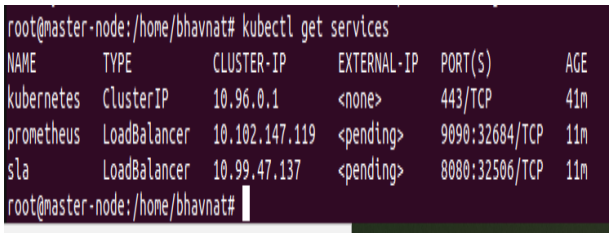
Figure 12: Shows the Prometheus Deployment Yaml File

```
root@master-node:/home/bhavnat/sla/k8s# kubectl create -f deploy.yaml
deployment.apps/sla created
service/sla created
root@master-node:/home/bhavnat/sla/k8s# kubectl create -f prometheus-deployment.yml
clusterrole.rbac.authorization.k8s.io/prometheus created
serviceaccount/prometheus created
clusterrolebinding.rbac.authorization.k8s.io/prometheus created
configmap/prometheus-config created
service/prometheus created
```

Figure 13: Shows the Prometheus Deployment result

```
bhavnat@master-node:~$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Figure 14: Shows proxy server running



(a) services running

(b) deployed pods

Figure 15: shows deployed services and applications in the cluster

describe	kubectl describe (-f FILENAME | TYPE [NAME_PREFIX | /NAME | -l label]) [flags]	Display the detailed state of one or more resources.

Figure 16: kubectl describe command [6]

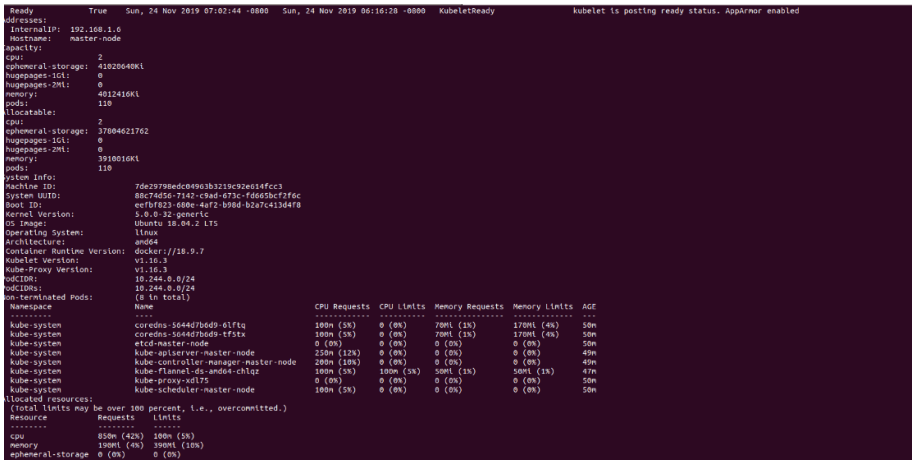


Figure 17: kubectl describe command output for nodes

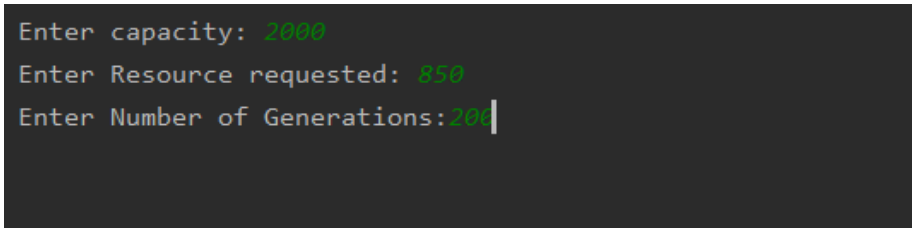


Figure 18: Genetic Algorithm code inputs

Figure 19: Application pod yaml update with new values

6.1 Genetic Algorithm

Taking the values given by describe command, enter it in Genetic Algorithm code attached in the zip file, by putting the requested value of the node in "request" parameter and total allotted resource in "pop_size"(Figure 18). Once done, run the code and a fit value will be presented alongwith the graphical representation of the evolution process.

6.2 Pod Yaml Deployment

Taking these generated values, reconfigure the application yaml file to assign the Genetically evolved values as resource request values for the application and change the "requests" tag in the yaml template with updated CPU and memory values(figure 19).

Deploy the application again in the cluster with updated requirements by running the command:

kubectl apply -f deploy.yaml

Change the limits tag of the template for achieving different observations of resource utilization levels by running the describe command again. An output will be seen as in figure 20

For checking the status of the cluster and the logs, run the command:

kubectl cluster-info

kubectl cluster-info dump

Multiple URLs will be presented for the applications hosted in Kubernetes cluster as well as logs (Figure 21). If needed, run the Prometheus URL in the browser and select the total duration metrics to compare the change in the total running duration in seconds

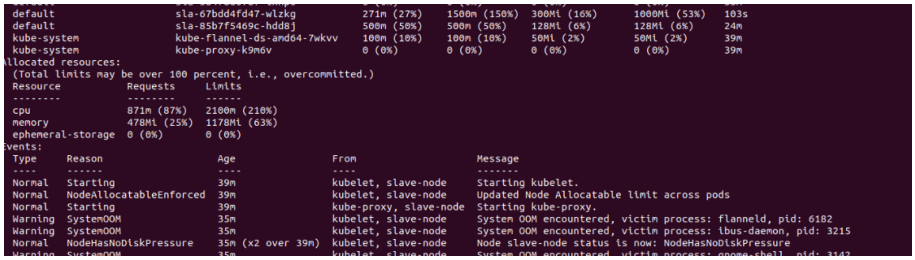


Figure 20: shows the outcome for the new resource requests

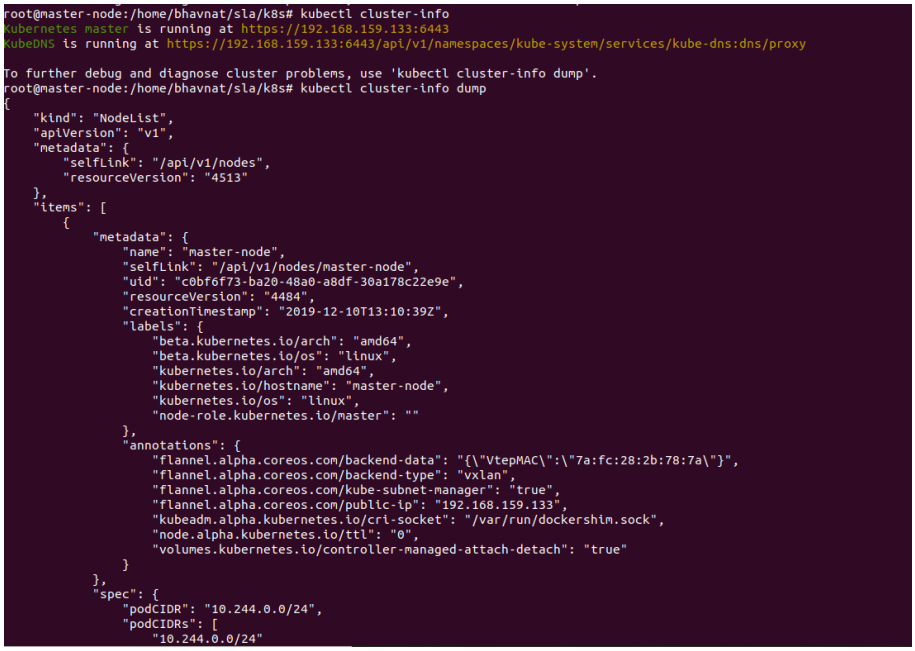


Figure 21: shows the cluster information and logs

according to different time intervals [7].

References

[1] V. L. Compendium, "Install and deploy kubernetes on ubuntu 18.04 lts,*https://vitux.com/install-and-deploy-kubernetes-on-ubuntu/*." [Online]. Available: https://vitux.com/install-and-deploy-kubernetes-on-ubuntu/

[2] M. Cezar, "How to permanently disable swap in linux,*https://www.tecmint.com/disable-swap-partition-in-centos-ubuntu/*." [Online]. Available: https://www.tecmint.com/disable-swap-partition-in-centos-ubuntu/

[3] Kubernetes, "Configure multiple schedulers,*https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/*." [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/

[4] K. Hendry, "Testing prometheus instrumentation metrics,*https://github.com/kaihendry/sla*." [Online]. Available: https://github.com/kaihendry/sla

[5] Kubernetes, "Deployments kubernetes,*https://kubernetes.io/docs/concepts/workloads/controllers/d* [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[6] Kubernetes.io, "Overview of kubectl,*https://kubernetes.io/docs/reference/kubectl/overview/*." [Online]. Available: https://kubernetes.io/docs/reference/kubectl/overview/

[7] ——, "Accessing clusters,*https://kubernetes.io/docs/tasks/access-application-cluster/access-cluster/*." [Online]. Available: https://kubernetes.io/docs/tasks/access-application-cluster/access-cluster/