International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V70I1P218

Volume 70 Issue 1, 164-170, January, 2022
©2022 Seventh Sense Research Group®

Enhancing Web Application Using Adaptive Containerized Application Placement Based on Clustering and Content Caching in The Cloud Environment

Mohamed. I.El-Shenawy[1], Hayam Mousa[2], Khaled M. Amin[3]

[1]*Information Technology Department Canadian International College New Cairo, Egypt*

[2,3] *Faculty of Computers and Information Menoufia University Shibin El Kom, Egypt*

moh.shenawy1983@gmail.com, hayam910@gmail.com, kh.amin.0.0@gmail.com

Abstract — *Datacenter traffic increases from day to day due to the massive increase of web applications hosted on the Internet. Some tools are used in resource management and capacity assessment in order to preserve a good performance for these applications. The container is a new trend for packaging and deploying micro-service-based applications. It is widely used to improve performance and achieve high user satisfaction. Autoscaling has become a vital feature in such applications' performance. This article targets to improve the quality of service through increasing resources utilization and reducing the number of application container kills and recreation. These targets can be achieved through dependency on the healthier nodes that have adequate resources. Machine Learning classification algorithms are used to predict healthy hosts. Then, a clustering algorithm is used to cluster healthy nodes into groups of containers workers' hosts based on their CPU and RAM utilization. In addition, content caching service has been integrated to improve application performance. This service decreases the network traffic to hosts nodes which subsequently decreases the required resources to handle these requests. The results ensure that the proposed model can achieve lower node failure with 33% of the default system. It also saves around 36% of bandwidth.*

Keywords — *Cloud computing, containers, autoscaling, virtualization, orchestration, machine learning.*

I. INTRODUCTION

The world has aimed to get high performance and massive demand for cloud service during the last years. Most of these services are built over virtual environments such as virtual machines and containers. Containers are a lightweight alternative to virtual machines that have grown in popularity among developers[1]. Containers save many resources and provide the high performance needed compared with virtual machines [2]. They use the host operating system's kernel to isolate each container by enclosing it with its required services. Therefore, the techniques that use containers offer the best performance, fast isolation, elastic deployment, and powerful resource sharing. They have become widely used by organizations to deploy their workloads on the cloud. As a result, container orchestration platforms have arisen. Orchestration is used to manage containerized applications' deployment.

Container orchestration manages container lifecycles. Containers lifecycle depends on the wise management for many issues, including automatic scaling, automating container deployment, management, networking, and availability[3]. One of the most critical issues is automatic scaling. The automatic scaling allows scaling up or down the used resources based on CPU or memory consumption. Automatic scaling ensures that the application is always available and that sufficient resources are available to prevent performance issues or outages.

Most of the web applications are built on cloud platforms. The Quality of Service (QOS)requirements are often varied and require various levels of support and services [4]. Failure to meet the required level of QOScauses downtimes and reduces the performance of the applications. In addition, failure to do so leads to a loss of revenue for service providers. Providers can offer on-demand performance by rolling out more containers. Moreover, characterization of load also helps predict future resource requirements and allows for a more efficient resource management plan.
Due to the existence of open-source workload traces, research works have been conducted on this topic. The use of prediction has helped prevent many emergencies and has improved the control over various complex systems. For such scenarios, using a prediction methodology can help minimize the complexity and provide better overall performance. A time series is a variable that has a value that is computed at different times. A complex time series can be very challenging to get the correct information out of. Consequently, workload prediction based on machine learning comes in.

This article aims to propose a comprehensive autoscaling model to enhance containerized applications using machine learning models. Also, it shows how that can affect end-users and service providers. A content cashing module has been integrated in order to achieve more

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

satisfaction for end-users. This module increases the response time and saves the bandwidth through string the most accessible data in cash storage. Consequently, the results show how content caching can affect service delivery performance.

The main contributions of this article are:

1. It provides a classification model that predicts the Kubernetes worker's healthy node. Therefore, unhealthy nodes can be filtered out.
2. It adopts a machine learning clustering model to group available healthy workers node into clusters based on their hardware load.
3. It integrates a content cashing module within the proposed model to improve performance.
4. It evaluates the proposed autoscaling model and shows how content caching affects the performance of both application and workers node hardware load.

The rest of the paper is organized as follows: Section II defines problem definition and briefly introduces the related work. Then, the section IIIdiscusses the proposed system. And, section IV presents evaluation and experimental results. Finally, Section V drives the conclusions.

II. RELATED WORK

Over the past years, auto-scaling VMs and containers in the cloud infrastructure have been widely recognized as an exciting research topic in computer science. Different research groups work on different aspects. Meanwhile, cost-efficient resource management based on real-time changes of workloads is critical for auto-scaling the VMs and containers in cloud environments. It helps to achieve the required levels of quality-of-service parameters (QoS) [4]. Several related projects are highlighted here with their approaches to treat this problem.

Moore et al. [5] provide elastic Docker scales up and down. Both CPU and memory are assigned to each container based on the application workload. Lin et al. [6] develop an autoscaling system to monitor network traffic requests and HTTP response time. This allows them to identify application performance in the cloud. Sotiriadis et al. [7] minimized performance degradation in cloud computing by introducing a virtual machine scheduling algorithm. They apply SVM to classify resource usage. Chen et al. [8] proposed a system called CloudScope, which is used in diagnosing performance interference among co-resident VMs. CloudScope measures performance interference using VM profiling information obtained from the hypervisor layer and then reassigns VMs to PMs in a way that interference is minimized. In Yousif et al. [9], Google workload trace is used as a dataset in which tasks are characterized and clustered based on the resources' usage. Xu et al. [10] formulate a generic job scheduling problem for parallel big data processing in heterogeneous clusters and design a K-Means-based task scheduling algorithm called KMTS. Matteo Nardelli et al. [11] propose Adaptive Container Deployment (ACD), a general containerized application deployment and adaptation model expressed as an Integer Linear Programming problem. All related work didn't depend on healthy nodes in their models, which will be proposed in this article in addition to content caching and its effect on system performance.

III. PROPOSED SYSTEM

The proposed model depicted in figure1 focuses only on healthy nodes to get high-performance applications without any hardware failure interruptions. The model starts by capturing the system logs from different sources for predicting healthy nodes. Then, it clusters these healthy nodes into groups based on their loads. Before application deployment, the model checks the application hardware requirements, then based on this requirement, the application is deployed to matched nodes group. The complete scenario is depicted in Figure 1.

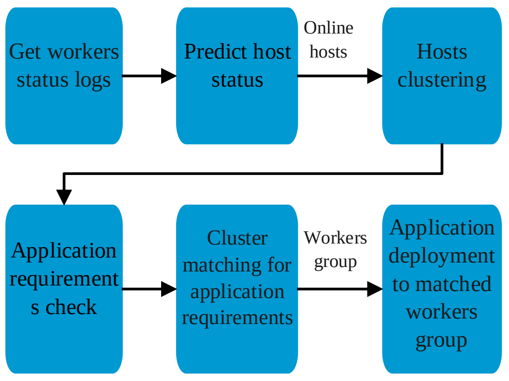
Get workers status logs
Predict host status
Online hosts
Hosts clustering
Application requirements check
Cluster matching for application requirements
Workers group
Application deployment to matched workers group

Fig.1 proposed prediction and clustering model

A. *System architecture*

Many container orchestration frameworks can deploy and manage multi-tiered applications in a cluster. One of the most famous ones of the containers framework is Kubernetes [12]. Kubernetes is an open-source container-management system that automates the deployment process involving scaling and management of computer applications. Google invented it. Cloud-Native Computing Foundation now backs it up. Kubernetes integrates various container tools and runs containers in a cluster with images created with Docker, which is now deprecated in favor of containers. The architecture of Kubernetes is divided as follows:

- **Control-plane** is the central node that manages the whole cluster, such as the workload, communication, and states between nodes. It also manages job scheduling such as starting, removing, and deploying containers.
- **The worker's nodes** are host containers. The cluster workers nodes run a container runtime such as Docker and the services that handle the configuration and communications of these containers.
- **Kubelet**is responsible for the node's operational state and containers hosted in every node. It manages the start, stops, and maintenance of application containers in pods. It continuously checks the pod's state and re-deploys if it is not in the desired state, i.e., failure.

- **Kube-proxy** combines a load-balancer and a network proxy in addition to networking operations. It routes traffic to the containers based on incoming request port number and IP address.
- **Container runtime** combines the running application and related libraries and any other dependencies. The containers are placed in pods. Containers are accessed from the outside world by exposing the external IP address.

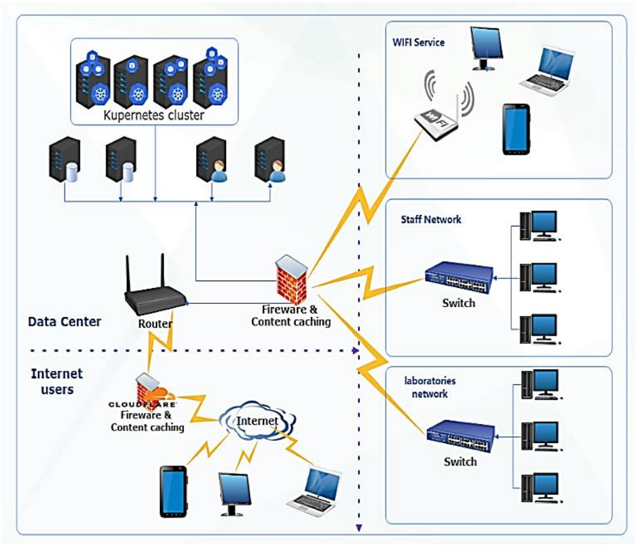

Fig. 2. Kubernetes cluster and datacenter.

The proposed system will be adopted on the described architecture in figure 2. The first stage in our proposal is predicting healthy nodes

B. Host health Prediction model

Providers and customers always need the running systems without any failures. The dependency on healthy nodes ensures better performance and avoids failures as much as possible. Healthy nodes have been defined through this article as the nodes that may not fail during the upcoming period with high probability. Host health depends on many factors, such as hardware failure and overloaded resources, which leads to instability in the performance of the node and subsequently leads to system failure.
Machine learning can enhance system stability by predicting host failures in advance so the system owners can take the necessary action in advance to avoid system failures. As a result, the assessment process avoids unhealthy nodes and depends on health. Thus, better performance can be achieved.

This article proposes a model to predict hosts' health status from Kubernetes logs and monitoring systems. The model filters out the predicted host that may fail. This allows Kubernetes to avoid them during application container deployment and depend only on healthy hosts.

The model uses a training dataset obtained from the Prometheus monitoring system. It focuses on hardware status and constantly alerts logs that are raised from node exports installed on each node to check their status. The dataset has multiple features exported from monitoring systems like host disks failure, host unusual network throughput, unusual disk read and write rate, host out of disk space, host network receive and transmit errors and Host out of memory alerts. These features are used to build the prediction model. These features directly affect the health of the node joined to the Kubernetes cluster.

The Kubernetes creates the required containerized application for workers. As mentioned in the system architecture, the control plan continuously checks the status of workers' nodes. If the worker's node is not available for any reason, the control plan sets the worker's node health status as 'NotReady'.Based on Kubernetes pod eviction timeout, the control plan waits 5 minutes to get the host back online and ready. If the host does not get back after that time, the Kubernetes automatically terminate the process and evict the pod from the failed node. Then, it recreates a new pod on another host with old volumes. If the termination process stocks, the required new pod creation also stucks.This leads to instability in the system pod replicas. Depending on healthy nodes known from the proposed prediction model through the deployment phase makes the system overcome the problem of NotReady nodes. Thus, it allows improving the system performance.
Multiple machine learning models are exploited in this context to predict the worker's host health state. These models are evaluated in the evaluation sections. The models include (SVM, LGBM, Random Forest, XG Boost)

C. Host Classification model

The Kubernetes cluster receives the application container creation during deployment. This process is based on the application replica. The controller automatically deploys the container on available workers in the cluster. Taking into consideration that each worker hasa different CPU, RAM, and disk IO. This difference in hardware load affects the application performance. Assigning the container to the most suitable worker improves the system’s performance. In this way, the workers with high resource capabilities are used with containers of high requirements. In addition, it minimizes the possible redeployments due to worker failures to cope with the application requirements.

The proposed Machine learning clustering model is depicted in figure 1. It starts by predicting the health workers' nodes, as illustrated. Then, it groups the worker's nodes in clusters based on their workload. A machine learning clustering algorithm is used for this purpose. The dataset features that were used for building the clustering model are based on multiple features. These features include CPU cores, CPU usage percentage, memory usage percentage, disk read throughput per second, disk write throughput per seconds, network received throughput per seconds, and network transmitted throughput per second. Here, the proposed model focus on clustering the nodes based on CPU usage percentage and memory usage percentage. These two features are the major ones that can give an indication of the worker's capabilities and availability.

Before the application is deployed, the application hardware requirement is checked. The application container is deployed to the worker node on the suitable group that matches the application container's required resources[13].

The proposed model aims to affect positively and directly the application performance. Each worker node in the Kubernetes cluster gets a different application workload according to its capabilities. The model is evaluated using the K-means cluster algorithm. K-means is used because it was one of the successful clustering algorithms [14,15,16].

D. Application content caching

The primary purpose of this research focuses on how to get the containerized application to serve the end-user efficiently and provide a high-performance application. By examination, it is found that if caching service is used, it will enhance the performance of running containers application. Content caching means storing subsets of requested data in high-speed storage layers, such that every time the end-user access the application, the cached content is loaded fast instead of every time call from the application workers node. Caching is a technique that can be applied to various layers of technology to reduce latency for large-scale applications.
Content caching was proposed in different research in 5G networks, wireless networks, and the cloud. [17,18,19,20,21]. To the best of our knowledge, the effect of integrating the content cashing service within containerized applications has not been investigated up to writing this article. The performance of this integration is evaluated on the containerized application in the evaluation section.

IV. PERFORMANCE EVALUATION

A. Evaluation environment settings

a) Test environment

The test was done on Huawei 1288H V5 server with specs "28 CPUs x Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, 512 GB RAM ", There was 20 virtual machine host Kubernetes cluster. The Kubernetes cluster uses the ubuntu 20.04 LTS Linux operating system. The mentoring server is Prometheus version 2.25.0, Prometheus/alert manager version 0.21.0, and node exporter version 1.1.1. For metrics visualization, Grafana version 7.4.2 was used. For security, the Huawei firewall, Cloudflare, was used.

b) Dataset

The dataset used in this paper is of our system resources usage history for a month. It is collected every 5 minutes. Data has been collected, and preprocessing carried on it. The dataset has been divided into standard train and test as 70:30, respectively. Missing data was cleaned and scaled to 15 minutes.

c) Evaluation metrics

The prediction model will be evaluated according to accuracy as depicted in equations (2). The model was tested to predict the Kubernetes workers node heath using unseen test data.

$$Acuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad (2)$$

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives

B. Evaluation results

In this Section, the proposed ML prediction models and their results were evaluated. The test results are summarized in Table 1.

Table 1: ML Perdition algorithms results for predicting healthy nodes.

Model	**Accuracy**
SVM	0.86
LGBM	0.95
Random forest	0.98
XG-Boost	**0.99**

Table 1 shows the test results. The best accuracy result was 0.99 using XG-Boost. There is no big difference between models except for SVM. It gets an accuracy of 0.86. It has poor performance compared to other models. Based on these results, XG-Boost is used in the prediction model in the rest of the experiments. After getting the results of the healthy hosts from the prediction model, the results are going to be fed into a clustering algorithm that groups hosts based on their loads. K-Mean clustering model is used. Table 2 shows each worker's node and its assignment to related clusters. Workers' nodes are grouped into five clusters as per table 2 and graph 3. The common characteristics of the group are as follows:

- Cluster 1 for Idle CPU and low memory usage Nodes.
- Cluster 2 for low CPU and high memory usage Nodes
- Cluster 3 for Normal CPU and Normal Memory usage Nodes
- Cluster 4 for semi-heavy CPU and moderate memory usage Nodes
- Cluster 5 for heavy and overload nodes

Table 2: The results of clustering healthy nodes into groups using k-mean with K=5

Cluster Number	**C1**	**C2**	**C3**	**C4**	**C5**
Workers Host Name Per cluster	H2 H4 H9 H2	H7 H10 H4 H16 H19 H20	H3 H5 H8	H11 H17	H1 H6 H13 H15 H18

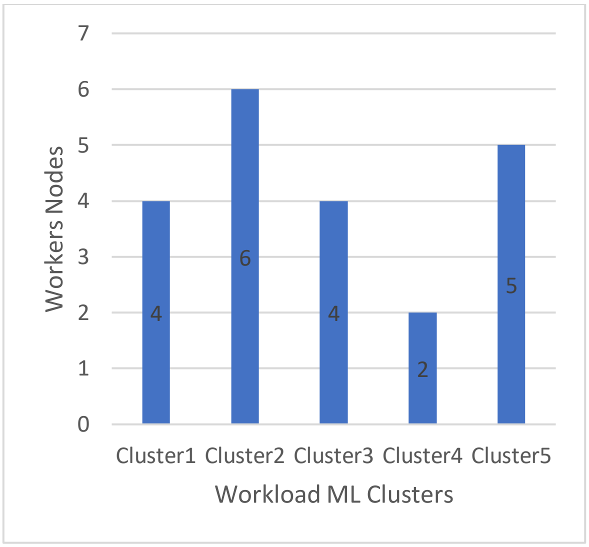

Fig. 3. Workers hosts and their assigns to clusters.

Before the requested application is deployed to containers, the application container hardware requirements are checked. This is done to assign the application to the most suitable cluster. First, check the application resources requirement from the application coding stage. Thus, the application container with heavy hardware requirements is assigned to cluster 1 or cluster2 based on available resources per clusters workers' hosts. In contrast, a medium application container is assigned to clusters 3 or 4 based on available resources per clusters workers hosts. However, in case of weak demand, the application container is assigned to cluster 5. The ML clustering model runs on a scheduled based to check the worker's node load and assign them to a suitable cluster. Each time an application needs to be deployed, it is deployed based on assignment rules stated earlier. The process of dividing the application container's hardware requirements into groups and assigning them to the suitable worker cluster has achieved high performance. Application containers differ in their requirements. Assigning the container to the most suitable node allows to improve the system resource utilization and to avoid failure nodes and stuck nodes subsequently as much as possible. The number of deployed containers, failed and stucked, are shown in Table 3 and figure 4 for both proposed ML-based deployment and Default models, Given 100 containers is deployed.

Table 3: Our model containers creation results vs. defaults model.

	Deployed Containers	Failed Containers	Failed Nodes	Stucked Containers
Default System	100	9	15	15
Our Model	100	**6**	**3**	**3**

By investigating table 3, the overall failed result of the default model, which did not consider the healthy nodes, was about 39 % of the running container. However, the proposed prediction, which uses a healthy node during deploying a containerized application, has failed results about 12 %. Also, when deploying containers, as seen in table 3, the number of failed containers during deployment has decreased from 9 to 6. This is because the healthy node is reflected in the deployment process. Since the hardware is always ready for tasks deployment, these results indicate that the proposed model enhances application stability and performance from 91% to 94%, with an enhancement of about 33% from the failed containers.

Then enhancements that the proposed model get because when using the default system setting if for any reason a workers node failed, the master node will wait for 5 minutes before start terminating the containers and start to recreate it on another healthy node. If the master node cannot terminate the container, the master node cannot create it again and is stocked in the creation process. However, with our model, ML is used to predict the worker's node health status. Thus, containers are usually assigned to healthy nodes. However, the faulted containers occur because of a sudden failure in the network connectivity and unplanned issues.

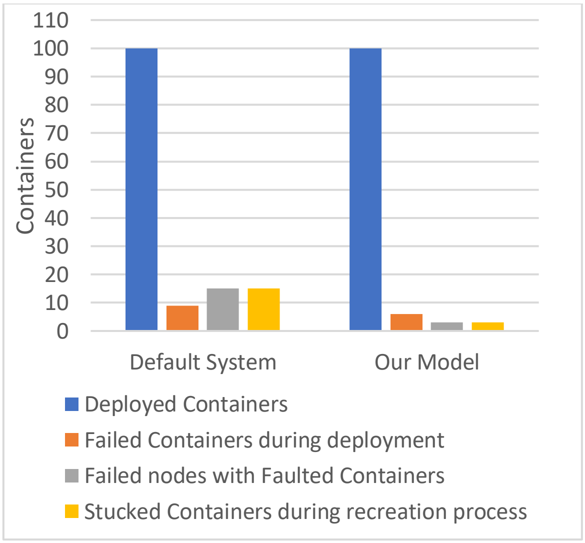

Fig. 4. Our model containers creation results.

In this part, we will show how content caching[22] affects the performance of the containers system and how it directly affects the consumed hardware resources to respond to the end-user request. Figures 5 & 6 show the data log of monitoring incoming traffic to our model for 7 days. The total number of visitors was 34,220 users. The total requests against the model were 5,230,375 requests. The total network traffic that incoming to containerized applications system was 105GB. By using the content caching service, it saved about 38GB from a total of 105GBof data served in the incoming bandwidth to the containerized running applications. This bandwidth saving directly affects the network connection and CPU and RAM that responded to this request. This is reflected in saving the bandwidth by about 36%.

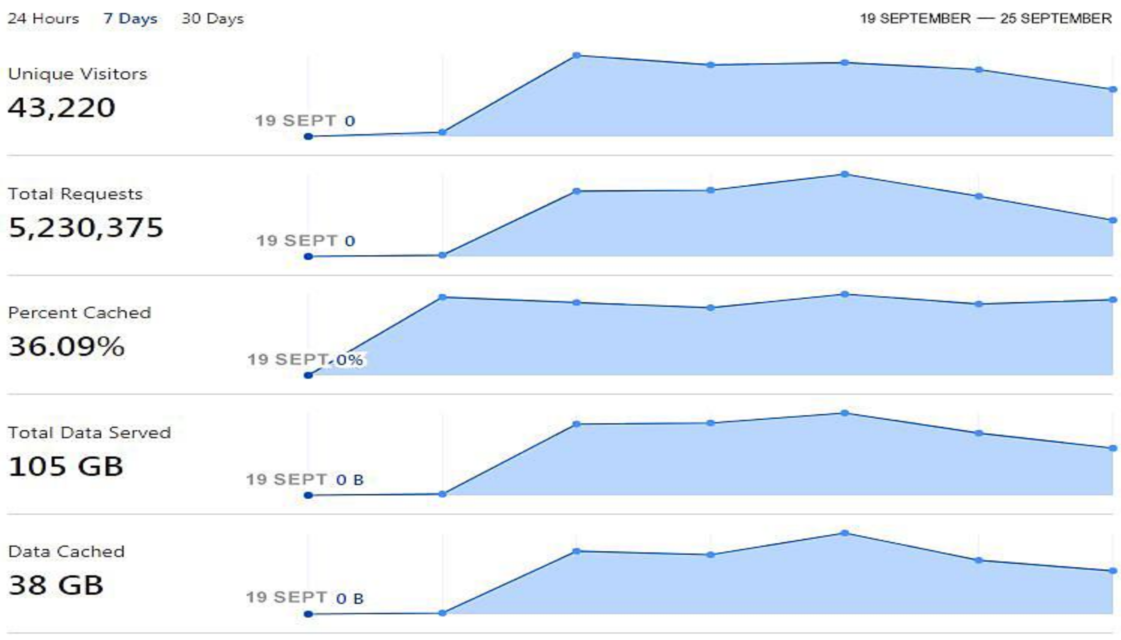

Fig. 5. Content caching bandwidth saving over 7 days.

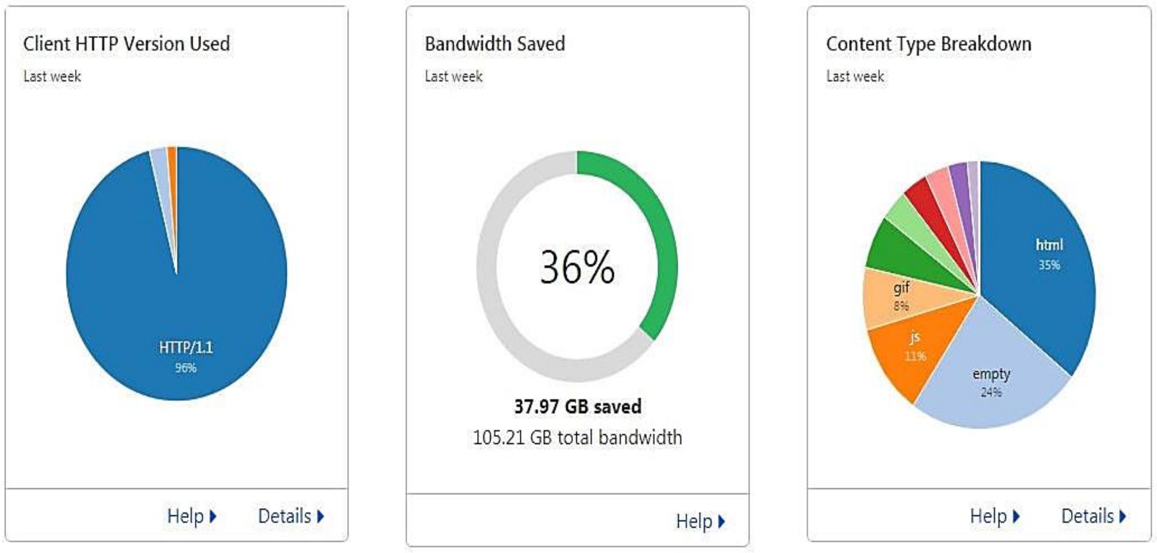

Fig. 6. Content caching content breakdown and bandwidth saving.

V. CONCLUSION AND FUTURE WORK

The container has become the trend of the current deployment of most applications. Many cloud vendors widely use containers today. This technology has many developments to meet the immediate need for elastic resource provisioning using autoscaling methods. This research provides a machine learning prediction model to predict healthy Kubernetes worker nodes. The XG-boost prediction model has an accuracy of 0.99 for predicting healthy nodes. This model has decreased the eviction of containers in case a worker's node fails. Then, a machine learning clustering model is applied to group health workers into clusters based on their resource load of CPU and memory. It checks the container application resources needed before deploying them into the Kubernetes cluster. The application is deployed to the matched cluster group based on the required resources. This model shows that it builds an application with high performance where each application gets its needed hardware which decreases the migration process and its consequences. About 33% of fault nodes have avoided being faulty because they suit the application deployed. Also, content caching saved about 38 GB from all requested 105GB with 36%bandwidth saving. This directly affects the incoming network bandwidth to the containerized application, which saves the required CPU and memory needed to serve other requests. For future work, prioritizing the application before deploying is one of the ideas that will be investigated to enhance resources utilization.

REFERENCES

[1] Celesti, D. Mulfari, M. Fazio, M. Villari, and A. Puliafito, Exploring Container Virtualization in IoT Clouds, 2016 IEEE International Conference on Smart Computing (SMARTCOMP), (2016) 1-6.

[2] A. M. Potdar, N. D G, S. Kengond, and M. M. Mulla, Performance evaluation of docker container and virtual machine, Procedia Computer Science, 171 (2020) 1419–1428.

[3] E. Casalicchio, Container Orchestration: A survey, Systems Modeling: Methodologies and Tools, (2018) 221–235.

[4] P. R. Desai. A survey of performance comparison between virtual machines and containers. ijcseonline. Org, (2016).

[5] K.B. Laura R.Moore, T.Ellahi. A coordinated reactive and predictive approach to cloud elasticity, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, (2013).

[6] C.-C. Lin, J.-J. Wu, P. Liu, J.-A. Lin, and L.-C. Song, Automatic Resource Scaling for web applications in the cloud, Grid and Pervasive Computing, (2013) 81–90.

[7] S. Sotiriadis, N. Bessis, and R. Buyya, Self-managed virtual machine scheduling in Cloud Systems, Information Sciences, 433-434 (2018) 381–400.

[8] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, and W. Knottenbelt, CloudScope: Diagnosing and Managing Performance Interference in Multi-tenant Clouds. In: 2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, (2015).

[9] S. A. Yousif & A. Al-Dulaimy, Clustering Cloud Workload Traces to Improve the Performance of Cloud Data Centers, In Proceedings of the World Congress on Engineering Conference, (2017).

[10] M. Xu, C.Q. Wu, A. Hou, Y. Wang, Intelligent scheduling for parallel jobs in big data processing systems, 2019 International Conference on Computing, Networking and Communications, (2019) 22–28.

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

[12] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration, Sensors, (2020).

[13] Kubernetes. Managing Resources for Containers (2021). [online] Available at: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

[14] M. Zekri, S. E. Kafhali, N. Aboutabit, and Y. Saadi, DDoS attack detection using machine learning techniques in cloud computing environments, In:2017 3rd International Conference of Cloud Computing Technologies and Applications CloudTech, (2017) 1-7.

[15] S. K. Sood, R. Sandhu, K. Singla, and V. Chang, IoT, big data and HPC based smart flood management framework, Sustainable Computing: Informatics and Systems, 20 (2018) 102-11.

[16] A. Yassine, S. Singh, M. S. Hossain, and G. Muhammad, IOT big data analytics for smart homes with fog and cloud computing, Future Generation Computer Systems, 91 (2019) 563–573.

[17] M. Liu, F. R. Yu, Y. Teng, V. C. M. Leung and M. Song, Computation Offloading and Content Caching in Wireless Blockchain Networks With Mobile Edge Computing, in IEEE Transactions on Vehicular Technology, 67(11) (2018) 11008-11021.

[18] J. Tang, T. Q. S. Quek, T. Chang and B. Shim, Systematic Resource Allocation in Cloud RAN With Caching as a Service Under Two Timescales, in IEEE Transactions on Communications, 67 (2019) 7755-7770,.

[19] O. Ayoub, F. Musumeci, M. Tornatore, and A. Pattavina, Energy-Efficient Video-On-Demand Content Caching and Distribution in Metro Area Networks, in IEEE Transactions on Green Communications and Networking, 3(1) (2019) 159-169.

[20] J. Song, M. Sheng, T. Q. S. Quek, C. Xu, and X. Wang, Learning-Based Content Caching and Sharing for Wireless Networks, in IEEE Transactions on Communications, 65(10) (2017) 4309-4324.

[21] T. Chen, B. Dong, Y. Chen, Y. Du, and S. Li, Multi-Objective Learning for Efficient Content Caching for Mobile Edge Networks, 2020 International Conference on Computing, Networking and Communications (ICNC), (2020) 543-547.

[22] D. E. Jayanti, R. Umar, and I. Riadi, Implementation of Cloudflare hosting for access speed on trading websites, SISFOTENIKA, 10(2) (2020) 227.