Intelligent Automation & Soft Computing
DOI: 10.32604/iasc.2023.028815
Article

Tech Science Press

Resource Based Automatic Calibration System (RBACS) Using Kubernetes Framework

Tahir Alyas[1], Nadia Tabassum[2], Muhammad Waseem Iqbal[3,*], Abdullah S. Alshahrani[4], Ahmed Alghamdi[5] and Syed Khuram Shahzad[6]

[1]Department of Computer Science, Lahore Garrison University, Lahore, 54000, Pakistan
[2]Department of Computer Science & IT, Virtual University of Pakistan, Lahore, 54000, Pakistan
[3]Department of Software Engineering, The Superior University, Lahore, 54000, Pakistan
[4]Department of Computer Science & Artificial Intelligence, College of Computer Science & Engineering, University of Jeddah, 21493, Saudi Arabia
[5]Department of Software Engineering, College of Computer Science and Engineering, University of Jeddah, 21493, Saudi Arabia
[6]Department of Informatics & Systems, University of Management & Technology, Lahore, 54000, Pakistan
*Corresponding Author: Muhammad Waseem Iqbal. Email: waseem.iqbal@superior.edu.pk
Received: 18 February 2022; Accepted: 24 March 2022

Abstract: Kubernetes, a container orchestrator for cloud-deployed applications, allows the application provider to scale automatically to match the fluctuating intensity of processing demand. Container cluster technology is used to encapsulate, isolate, and deploy applications, addressing the issue of low system reliability due to interlocking failures. Cloud-based platforms usually entail users define application resource supplies for eco container virtualization. There is a constant problem of over-service in data centers for cloud service providers. Higher operating costs and incompetent resource utilization can occur in a waste of resources. Kubernetes revolutionized the orchestration of the container in the cloud-native age. It can adaptively manage resources and schedule containers, which provide real-time status of the cluster at runtime without the user's contribution. Kubernetes clusters face unpredictable traffic, and the cluster performs manual expansion configuration by the controller. Due to operational delays, the system will become unstable, and the service will be unavailable. This work proposed an RBACS that vigorously amended the distribution of containers operating in the entire Kubernetes cluster. RBACS allocation pattern is analyzed with the Kubernetes VPA. To estimate the overall cost of RBACS, we use several scientific benchmarks comparing the accomplishment of container to remote node migration and on-site relocation. The experiments ran on the simulations to show the method's effectiveness yielded high precision in the real-time deployment of resources in eco containers. Compared to the default baseline, Kubernetes results in much fewer dropped requests with only slightly more supplied resources.

Keywords: Docker; container; virtualization; cloud resource; kubernetes

1 Introduction

Public cloud is used for common personnel to allow services and systems comfortably [1]. The development of the Internet has brought about huge changes in cloud computing and has developed into an open collaborative business model that looks for services and further diversifies other energy sources. The Business organization has distributed cloud computing service systems into three levels: Infrastructure as a service, software as a service, and platform as a service.

Community cloud is a collective effort in computing where internal or third party management or internal or external hosting is shared between many organizations in a given community, with a fundamentally special interest in security, compliance, sovereignty [2]. A private cloud provides services and systems that run within the organization. Alternatively, it can be controlled by third parties or coalitions themselves. A hybrid cloud combines a fusion of public and private clouds. The private cloud executes major accomplishments, and non-critical events are accomplished by the public cloud.

Infrastructure as a service is provided with buyers, systems, inventory, processing, and various other computing resources. Consumers can check the operating system, inventory, and published applications. Customers may also have partial rights to choose system administration components, such as owning a firewall. Platform as a Service is a cloud computing model. A third-party provider provides hardware and software tools, usually those needed for application development to users on the Internet. These apps can be accessed from various gadgets through a light client interface like a web browser [3]. Software as a service allows the consumer to use the experiences, for instance, requests put by the service provider that continues to work on the infrastructure.

Regarding the dynamically evolving cloud services architecture, modular software development is a powerful model for ensuring the quality, flexibility, and sustainability of personal services. Using modern development frameworks, cloud services can be designed and developed into a set of small, poorly coupled modules that can be quickly changed, shared, and reused in another instance. As a result, software upgrade modules allow for faster deployment of cloud services by integrating existing public modules. Combining a microservice style of external service architecture with a software development module related to the internal architecture of cloud services is an effective way to address the growing complexity of software-based programs and the need for rapid adjustment to ever-changing needs [4].

While the development of newly developed modules and public availability enables faster software upgrades, satisfying unmet needs such as reliability, efficiency, and security in this situation is a daunting task. Newly expanded modules may become unstable, weak, or unreliable. Indeed, we want to establish an isolated context for these units to benefit from improved security, failure, and perfect resource management, but without the need to create, manage, and deploy dedicated services. The development and maintenance of disproportionate services have been defined as one of the key issues in the construction style of micro-services and lead to performance problems. This led to the use of container virtualization to isolate units in one cloud service from each other, following the definition of isolation limitation. Our approach meets some inactive requirements by automatically changing cloud services consisting of components in a container ecosystem [5].

Technology that enables microservice architecture for containerized applications. Virtualization and container technology is the same, but containers can still deploy or integrate applications, such as libraries, configuration files, and full package duplexing. This container is used to put each software package in its normal configuration [6]. The method specified for operating system virtualization is called containerization as shown in Fig. 1.

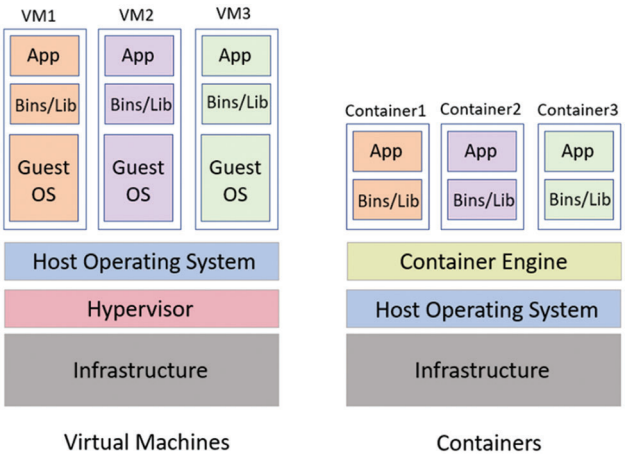

Figure 1: Virtualization and containerization

Serverless computing introduced large-scale parallelism and was created primarily for event-driven applications that require light processing in response to a trigger (e.g., to do a minor image manipulation when a file is uploaded to an Amazon S3 bucket or to access a NoSQL back-end to retrieve data upon invocation of a REST API). However, the programming model imposed by these services, such as Amazon web services (AWS) Lambda, makes it difficult to use this service for general application execution.

Kubernetes is a system that revolutionizes the distribution, expansion, and management of container applications. Kubernetes collects application tools into logical units for easy management and discovery. Kubernetes is an expanded, portable, and open-source platform for tracking workload and containerized services that make it easy to configure entry and automation. There is a large and rapidly emerging ecosystem [7].

In Fig. 2 the control plane is directly connected with worker node 1 and worker node 2. Kubectl is connected through user interface through User interface/Command Line Interface(UI/CLI. Control Plane is further divided into API Server, scheduler, controller manager and etcd. Worker node is responsible for the managing Pods containing different containers through kubelet and kube-proxy.

Etcd stores the available position data for each hub in the group. It is a very accessible relationship store that can extend between hubs. It can only be opened by the Kubernetes API server, as it may contain confidential data. The application programming interface (API) server uses API to complete all tasks. With the interface tool that programs the interface server, devices and libraries can immediately communicate with the interface [8].

Kube-config is a software package next to the server-side device for communication. The scheduler module is the section that is responsible for decentralizing the remaining tasks at hand, tracking the workload usage on the group hub, and then setting up the remaining activities at hand to access and identify the tasks at hand. It applies changes to make the current state of the server ideal by receiving the related conditions of the groups [9]. Arguably, this is the component responsible for assigning cases to accessible hubs. The dispatcher is responsible for special expenses and assigns points to the new hub. The controller manager is liable for managing the status of collectors and performing most of the tasks for the collectors. It is generally considered a daemon process as it runs in an endless loop and is responsible for collecting data and sending it to the API server as shown in Fig. 2.

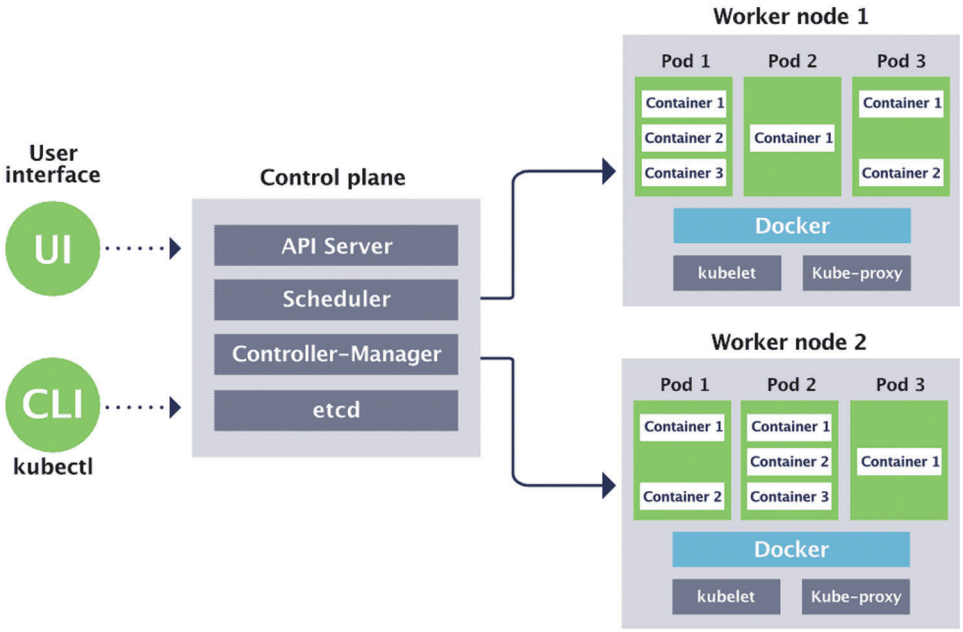

Figure 2: Kubernetes architecture

- Pod: It acts as the default server and assigns a virtual Internet protocol (IP) address to the main server.
- Kubelet: The Kubernetes node operator and running on each node of the System. It also has authority for managing and reporting node events such as pod status and node resources that the main node must use. It is an audit pod assignment through the main server to maintain these filters on nodes.
- cAdvisor: It is a monitoring tool for resource utilization of open-source containers that automatically collect all information of containers in the node regarding resource usage.

The main contribution of the proposed work is to introduce a scalable and comprehensive controller for kubernetes in the eco-container environment of cloud structure. The objective is to minimize resource utilization with the goal of improving performance and proficiency for containerized virtualization and optimizing scheduling or mapping of containers to available nodes in multi-constraint kubernetes orchestration.

2 Literature Review

Deployment of virtual machines in amazon Elastic computing cloud (EC2) to simulate in parallel. The infrastructure design is comprised of EC2 and local computing resources with a central manager, condor (middleware that runs simulation tasks), responsible for distributing tasks and scheduling. Using a cloud in a pandemic scenario analysis performance must speed up and colleagues introduced the cloud sim and container base cloud sim add-on to compare container planning and provisioning policies better. To give sponsorship for modeling and simulating cloud computing environments in containers. They developed a modeling architecture for container clouds and have implemented it as an extension of cloud sim [10].

Recent studies classify and systematically compare existing container technology research organizations and their cloud applications as a service. And as a container-based middleware for packaging applications and how distribution infrastructure works to understand the platform better. It is recommended to use a lightweight container automatic scale to achieve flexible automatic measurement. Provide detailed information about system architecture and automatic measurement, including monitoring mechanisms, history recorder, decision mechanisms, and implementation mechanisms. There are no more experimental

standards for more comprehensive evaluation in the public cloud. This article does not classify workloads as memory constraints and disk-attached Central processing units (CPUs). There is no preventive automatic scaling tool that can combine the autoreactive scaler with some mature prediction methods to scale the container according to the number and scale of the containers [11].

Provide automatic scalers that meet the needs of flexible container applications. This article focuses on the deployment, configuration, and management of the Virtual, flexible computing group (VIC) to manage scientific workloads. It has developed a mixed-scale strategy based on a resource demand forecasting model that is matched with service level agreements to cope with rapidly changing workloads. The System is used in a central cloud environment for private containers, and experimental results show that the volume is automated [12]. The nodes of the scientific block are stored in a repository running on bare metal. Introduce the Docker open-source elastic cluster (EC4Docker) integrated with Docker Swarm, automatically creating a scalable virtual computer cluster in warehouses across distributed systems [13].

New improvement storage concept, introducing a new model for allocating storage resources in the cloud system. A new hybrid algorithm is being implanted to obtain the best possible allocation of storage resources. The Lion's algorithm (LA) was introduced with the support of Whale random update (WR-LA), and it is a hybrid form of the LA and the Whale Improvement Algorithm (WOA) [14].

Moreover, an optimal resource allocation solution has been developed, considering minimum, group usage balance, system failure, total network distance. The scientific contributions reviewed include a combination of technical use cases, analysis, and solution architecture (the implementation of Google kubernetes engine, and the concept of threatening the storage environment in the transfer of application repositories. Create Google Cloud Platform (GCP) resources from the local command line management program, and then send, install, and configure the required Google software development kit (SDK) command-line configuration gcloud [15].

A large number of studies have shown that compared with traditional cloud-based virtualization and hypervisors, storage-based virtualization has many advantages and reduces the cost of creating new solutions. Besides microsystem architecture assists [16] to construct scalable software that can be used in cloud environments. The intelligent microservice monitoring used in kubernetes is an automated learning engine for intelligent analysis and a core element of many cloud management solutions [17].

3 Proposed Methodology

The proposed kubernetes infrastructure model, which aims to consider the main factor of the cloud framework, namely resource estimation. Kubernetes attempts to use Vertical pod auto-scaler (VPA) to solve the resource estimation problem, which does not oblige users to input resource requirements. The VPA creates estimates based on the application's current resources and then improves the estimates by reassessing the application with the new forecast resources. The correction method requires closing the application and then reinstalling with the estimated resources. Although this method can be applied to stateless services, it has serious shortcomings in the state- and performance-sensitive applications [18].

RBACS model is proposed used to solve the resource estimations problem in kubernetes settings. The model includes several modules, which have been integrated into an effective operating model as shown in Fig. 3. All modules work together to improve reliability. The main goal of this research is to explore new ways to enhance the thorough management of kubernetes containers. The method evaluates the effective number of resources required by the application, moves constantly, updates the allocation of resources, and carries on the application's execution. The proposed model of the kubernetes infrastructure includes the following modules:

- RBACS components include the kubernetes master and docker and RBACS with metrics servies.
- Elastic Compute Cloud (EC2) cluster of worker nodes with Network File System (NFS)

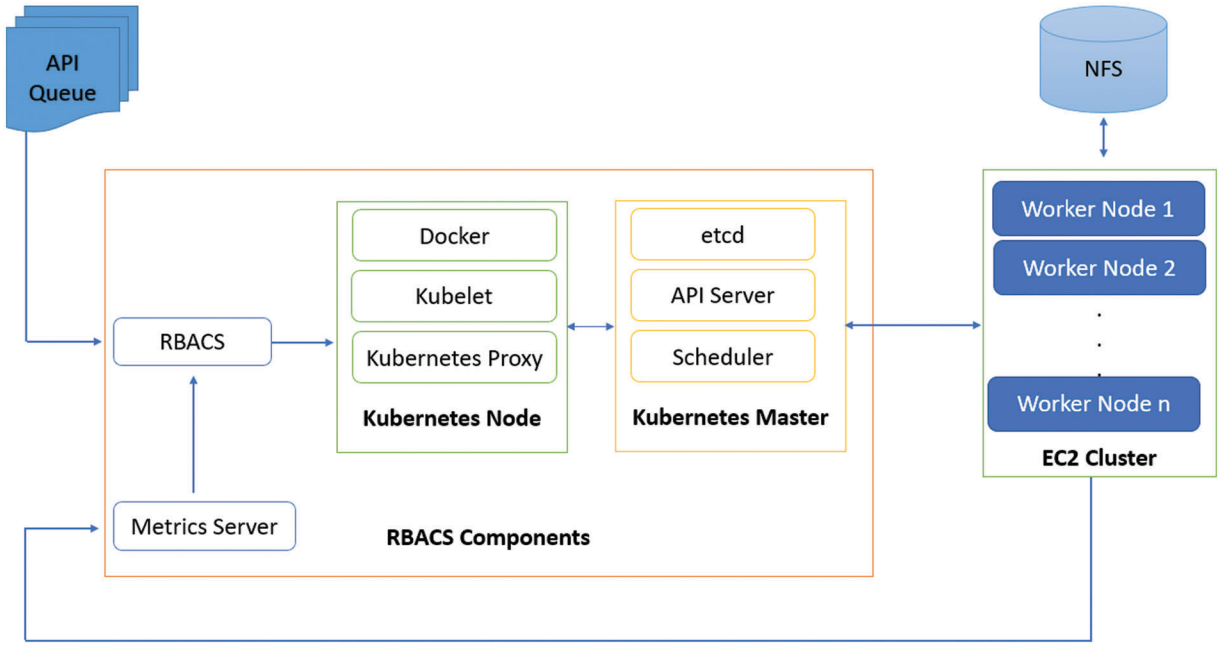

Figure 3: Proposed RBACS framework

All applications queue will send to RBACS components where kubectl will check the worker node of desired request. RBACS will also check the docker engine through metrices servers with EC2 cluster. Users do not need to provide CPU and memory resources for pod in the currently enabled VPA implementation. Initially, by default, kubernetes allocation is the smallest or smallest configurable allocation of guaranteed resources in the pod. VPA will run a recommendation program to monitor the pod usage history and suggest the latest boundaries as needed.

It is recommended that the program use the following default formula for new resources. Although these equations can be modified, this resource correction method has its shortcomings. For instance, maximum usage of memory may be a transient peak in an application, however, the goal of VPA is to distribute this amount throughout the life of the application. Also, after changing the distribution the current VPA implementation will restart pod. As a result of this destructive process, the condition of the container is lost. Although docker has released docker updates, it cannot be applied directly to large distribution systems (such as kubernetes) to update resources allocated to repositories.

RBACS contains two key modules: runtime analysis and approximation system. This is because several units, such as schedulers, controllers, and kubelet, depend on the initial allocation to make decisions. If these modules are incompatible with each other, it is currently difficult to use the latest update feature of docker for VPA. Recent work has demonstrated the efficiency of related runtime analysis systems. A vertical auto-scaling system can modify the allotment (if needed) without interruption during operation. Based on data collected every 60 seconds, the following formula used to estimate: RBACS fully organizes applications when it reaches the cluster and collects resource consumption statistics every single second.

Compiling and analyzing the data involves the use of CPU and memory provided by the matrix server. A new estimate is made every 60 seconds built on this data. The following formula is used to make an estimate based on the data collected every 60 seconds. Eq. (1) will buffer the required resources and calculate the observations.

$$\text{Buffer} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(yi-\bar{y})^2} \quad (1)$$

$$\textit{Required Resource} = \textit{Median of Observations} + \textit{Buffer} \quad (2)$$

In the formula, the total count of observations is symbolized by N, and Yi represents i. The observed value, y is the arithmetic mean of the observed value. The buffer is the absolute deviation of the observed value. n is the total number of observations, yi is the ith observation, and y is the arithmetic mean of the observations in the formula. The buffer is the space between two points. Since the application was primarily programmed apart from resource allocation, the following approach is used to fix or set the limit (disruptive: the application is restored after debugging), as shown in Eq. (2). Use it when restarting the application, it will not affect the results. Non-disruptive: scan the application and restart using the checkpoint. This process may involve migrating containers [19].

It explores a new method of enhancing the containers orchestration in kubernetes, which can dynamically estimate the number of resources required by an application, not just migration, update resource allocation and continue execution of the application. If the approximated number of resources are not accessible on the same node, the program is moved to an alternative available point. Even if there is no backup on any node, the program will continue to run in its final state. The estimation and correction procedure continues until the application is accomplished.

This study introduces the design and implementation of a new non-disruptive RBACS program for kubernetes. The program includes: making trade-offs in various RBACS performance design choices, including configuration intervals, application restart overhead, and storage migration. A flexible and configurable framework, RBACS, is used to solve the problem of resource correction in Kubernetes settings. RBACS uses user-space checkpoint restoration (CRIU) to create checkpoints and restore docker execution, so there is no need to destroy and restart the application. Suppose the usage of the application exceeds 10% of the allocated resources. In that case, RBACS will create another assessment, check the container running the application, as well as use the new estimate to schedule it on the cluster. RBACS is most suitable for target applications, requiring few resource changes, and its design can also adapt to target applications that are frequently changed through multiple migrations.

Determining the right resources for end-users is challenging because program requirements can vary for each run, depending on different configuration values, such as input file size, optimized components, selection of input parameters, And kernel applications. Miscalculations in custom application resource requirements are a well-known issue and are not limited to Kubernetes [20].

The Fig. 4, shows the detail of the proposed algorithm steps from start to end. In step 1 the application is submitted to RBACS then monitoring resource usage will monitor the id from the kubernetes cluster in step 2. RBACS creates the startup specification required to perform tasks on the kubernetes cluster. The specification contains a unique identifier for monitoring application resource usage. The requirement description is sent to kubectl in step 3, a component of kubernetes for cluster management. Kubectl dispatches the work to the kubernetes master in step 4. Kubernetes schedules request to master working nodes. In step 5, the metric server accumulates detailed information about the resource usage of all the clusters running containers.

In step 6, RBECS examines the use of each container contrary to resource distribution. If the allocation and use are the same, RBACC allows the process to continue. If allocated and used do not go along with. In step 7, RBACS dispatch two commands to docker, one instruction is used to test the container, and the other instruction creates an image containing the data collected from the container. In step 8 creation of image made by docker. In Step 9, images and checkpoints are saved on NFS so it may be available to the rest of the cluster. In step 9a, the generated checkpoint and image information is referred to as RBACS.

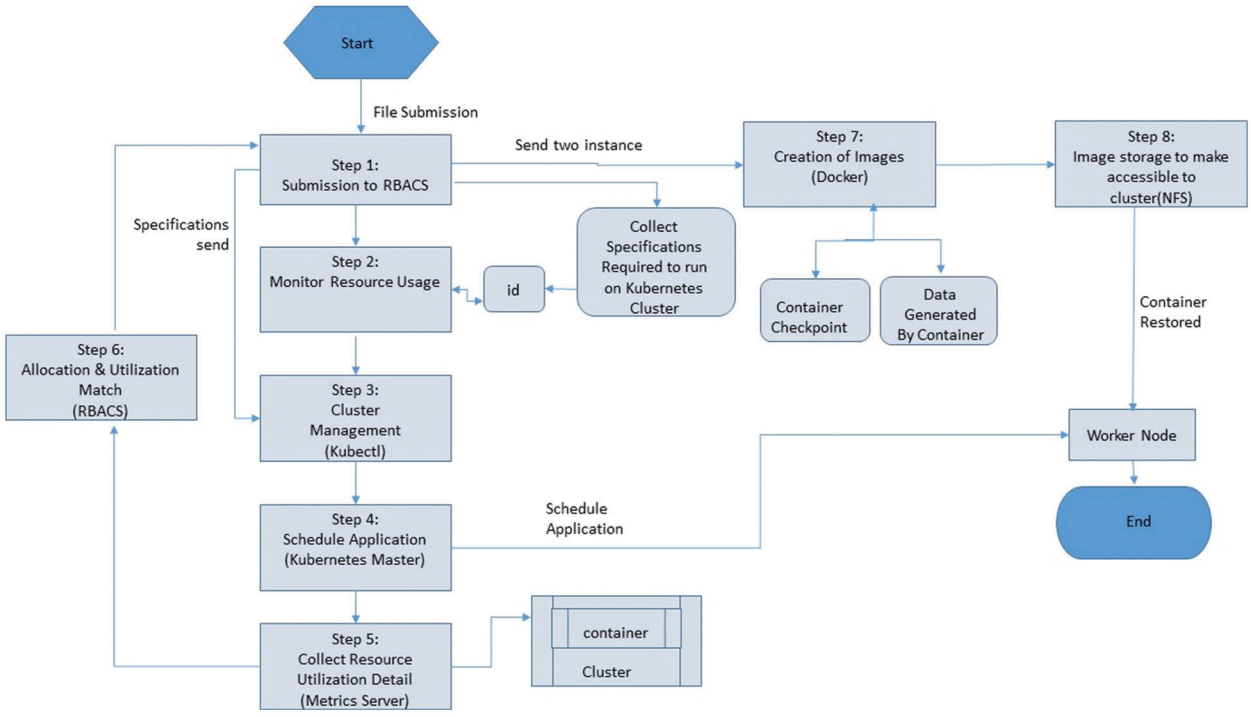

Figure 4: Proposed algorithm steps

In step 10, the RBACS prepares the launch description with the image and checkpoint icon and put forward to kubectl. The kubectl sends the work to the kubernetes master, which performs the task scheduling on the worker nodes. Then upload the image and checkpoint to the specific computer and restore the container. The proposed algorithm includes 8 steps, as shown in the Fig. 4. The algorithm will take input files, measure the required resources, and provide resource estimates, and schedule applications on working nodes. The proposed framework uses the RBACS estimation method to provide the best resource requirements for any application that installs bubernetes applications. Then use the container and docker to install the System [21].

4 Experimental Results

The experimental cluster configuration consists of 1 kubernetes master node and 8 kubernetes worker nodes, all in the AWS cloud. In Tab. 1 lists the descriptions of the settings.

Table 1: Experimental details

Software/OS	Version/Description
AWS EC2 Nodes (28)	c5.2xlarge compute-optimized, 7 core 2.5 GHz processor, 32 GB
Operating System	Ubuntu 18.04.1 LTS operating system
Docker	17.06.2
Kubernetes	1.12.4

Kubernetes linux manager node:

- Machine type: n1-standard-1
- OS: Ubuntu 18.04 LTS
- CPU: 1 vCPU
- Memory: 3.75 GB memory

Kubernetes windows worker node:

- Machine type: n1-standard-2
- OS: Windows server 2019 datacenter
- CPU: 2 vCPU

Docker swarm windows worker node:

- Machine type: n1-standard-2
- OS: Windows server 2019 datacenter
- CPU: 2 vCPU
- Memory: 7.5 GB memory

Simulations show differences in execution time or performance when the RBACS estimation method is deployed. An observation interval of 60 seconds was used for the experiment. On linux, we reduced execution time by 49.10%, and on windows by 23.65%. Using migrations can always improve runtime. As seen in the application, the runtime decreases as the migration image increases.

Conversely, if the initial estimate is incorrect or the application changes the operating system requirements, the benefits of the migration are more evident. For instance, on windows, applications slowly increase their usage, which can lead to incorrect estimates and several migrations or reboots. Fig. 6 equates the migration costs of RBACS when expanding locally on the same node and expanding on another node. The node selector option is used in kubernetes pod-spec to manage in-place migration and remote node migration. By connecting the pod to a specific node, In-place migration is performed, thus obtaining runtime benefits. It is observed that the windows runtime decreased by 4.1% and that of linux by 2.09%.

At the same time, when creating evaluation resources and determining that the application should be scaled, a checkpoint is created regardless of the available resources. The concept is designed to deal with situations where claims are canceled due to a lack of improper resources and allowed to be executed permanently.

The Fig. 5 depicts the average run times for default Kubernetes, kubernetes VPA, and RBACS. In both settings have a similar set of applications with equivalent resource requirements, *i.e.*, + 40% optimal resources. In simulations work, it is noticed that the default kubernetes, which cannot modify resources, takes much longer. On average, the default kubernetes took 21, 134 seconds.

Kubernetes with VPA worked much better than kubernetes by default, even restarting pods. By comparison, the kubernetes VPA has a much better run time, averaging 12, 141 seconds. This reduces execution time by approximately 47%. Restarting the packages were supposed to be a malicious move, but because VPA can fix the distribution and make resources available, kubernetes will run more applications, which will reduce runtime.

The proposed approach, RBACS, accomplishes upgrades by dropping the cost of reboots by incorporating the migration of containers. It is observed that using RBACS further reduces the execution time to 8993 seconds. It represents an improvement of approximately 25% over kubernetes VPA and approximately 42% over the default setting of kubernetes.

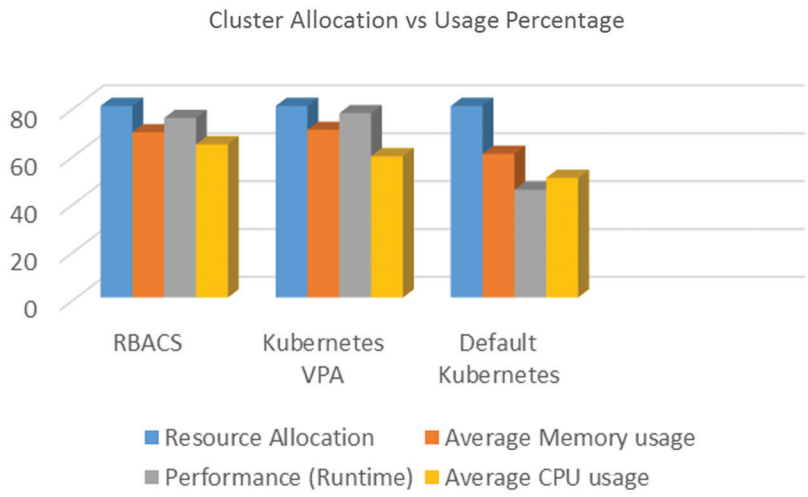

Figure 5: Cluster allocation *vs*. usage percentage

In Fig. 6 shows the use of a processor that matches the distribution. By default, in kubernetes, it is noted that only about 39% of the allocated processor is exhausted by programs. It shows that more than half of the allocated CPU remains idle. In this way misappropriate resources increase the overall runtime cause many applications waiting queue. Kubernetes with VPA was able to adjust the distribution and thus with much better use about 72%.

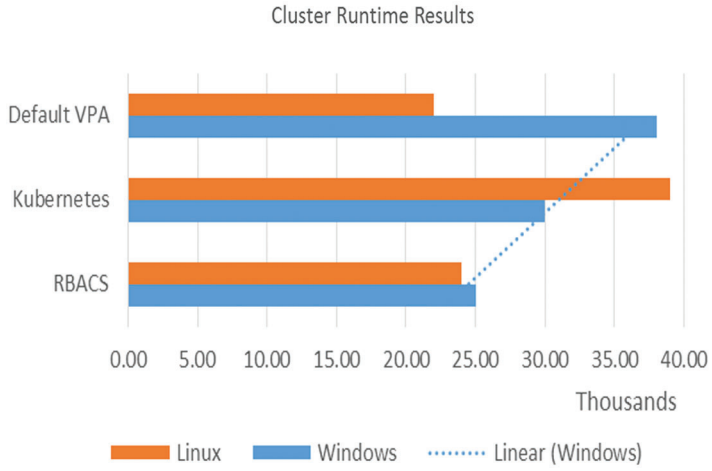

Figure 6: Cluster runtime results

Though, Kubernetes VPA has the disadvantage of restarting applications. Instead of immediately restarting the program, we noticed that Kubernetes queued the task because this causes persistence in the application implementation. In those circumstances in which the application provides a service, the wait can be long until the application is restarted. The application waiting to be continued has the highest priority in the RBACS queue. Using RBACS, with the help of container migration, better CPU utilization of around 5.98% has been observed in Fig. 7 with no state of failure.

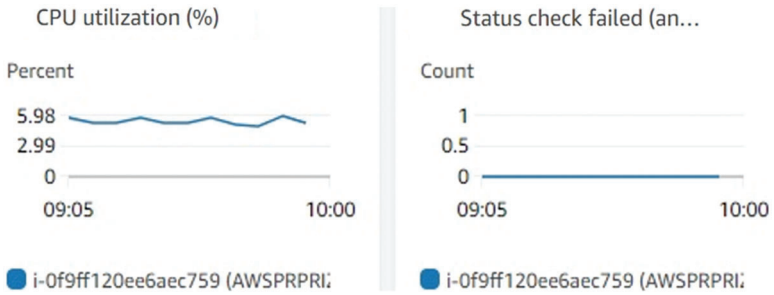

Figure 7: CPU usage

In Fig. 8 shows the average cluster memory usage using the default Kubernetes, Kubernetes VPA, and RBACS. The Kubernetes default average memory usage was 832 MB although the allotment was 90% and 192 MB is still available. We also evaluated memory performance, a simple program for measuring immortal memory bandwidth. It uses a data set significantly bigger than the cache accessible in the processing environment to perform four types of vector functions: add, copy, and unpack to reduce the latency of limited caches and prevent memory reuse. Regardless of vector operations, all lightweight virtualization, and native systems show similar performance. It's because of lightweight virtualization that can revert unused memory to other containers and the host, thereby improving memory utilization.

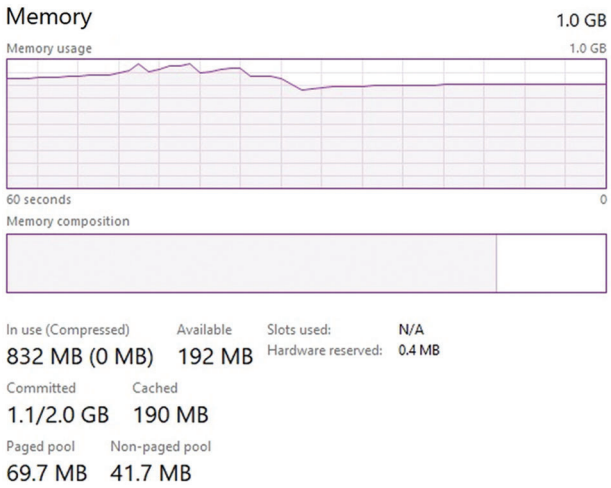

Figure 8: Disc performance

The result of default virtualization is the worst, with an average overhead of 4.1% compared with the local system. This overhead is caused by the virtualization layer based on the hypervisor, which converts memory access instructions, causing Performance drawbacks. The experiment is also used to estimate stored virtual disks' I/O performance. Create and measure various file operations and access patterns (initial write, read, overwrite, etc.). The experiment was compared with a file size of 10 GB and a recording size of 4 KB in Fig. 8.

The results for kubernetes and RBACS are similar because kubernetes and RBACS share the file system. The results for docker and open virtualization are almost similar, but if you read it, docker averages 4.1% compared to the local system, while open virtualization is 67%. The results show that they have an individual file system and the cost of RBACS and kubernetes VPA systems is lower than that of disk

quotas and I/O scheduling. Because the hypervisor-based virtualization layer requires drivers to assist disk I/O, the worst-case results are monitored for all I/O processes in the default Kubernetes.

Network performance was evaluated under various conditions to measure network performance. It performs simple tests, such as ping pong, by sending step-by-step messages between two processes on a network. Message sizes are regularly selected and sent to simulate errors and fully test the communication system. Every Single data point contains several ping-pong assessments to provide an exact time to calculate the delay. The max network performance in (bytes) is 180 k with minimum of 89.9 k where as the network performance remains in below 106 (bytes) thoright network out.

In Fig. 9 shows a comparison of network bandwidth in each lightweight container virtualization. In simulations work, it is noticed that the default kubernetes, which cannot modify resources, takes much longer. On average, the default kubernetes took 21, 134 seconds. By comparison, the kubernetes VPA has a much better run time, averaging 12, 141 seconds. This reduces execution time by approximately 47%. kubernetes with VPA worked much better than kubernetes by default, even restarting pods. Restarting the packages were supposed to be a malicious move, but because VPA can fix the distribution and make resources available, kubernetes will run more applications, which will reduce runtime.

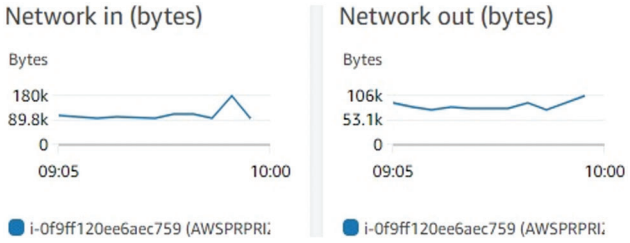

Figure 9: Network performance

Kubernetes achieved native-like performance, followed by docker and open virtualization. The ugliest result was observed in the default VPA because the deterioration in the performance of the VPA network causes additional complexity in the reception and transmission of packets. We did not configure a real NIC in the open virtualization system because this would decrease scalability due to the limited number of NICs that typically exist on the host.

It is noticed that RBACS can move ad-hoc tasks and continue running with the ability to scale them vertically. The vertical scalability of service operations does not require migration, because services usually do not need to maintain state after resuming. Also, service tasks often include replication tasks that assist load balance, so restarting these tasks is not very important. Though state sensitive jobs must maintain the status of implementation otherwise, they must restart execution from the beginning.

The container continues to move from one machine to another without affecting work cause a ripple effect in the migration of the container. It occurs when the estimated time interval is short and the assessment is not accurate. Sometimes, migrating the container is not the finest solution. On behalf of stateless applications, there is no need to maintain a state. Therefore, restarting the application with appropriate resources is better than migration. To balance the load, most stateless applications are usually reproduced. If one of the reproduced applications is resumed, the service will be temporarily degraded until the new replication replaces it. Reliability with multiple container migrations is more than four times during the simulations, it is observed that the migration image is sometimes created flawlessly by the existing limitation.

Migration downtime container migration involves multiple time-consuming steps. Inappropriate estimates can result in longer execution times in applications that generate large amounts of data takes a

long time because the generated data must be wrapped in a docker image and then copied to the destination computer, a significant increase in the implementation time of the transfer process by moving it to different operating systems. The application's execution time on windows exceeds 300 seconds at 15-second intervals and is less than half at 60-second intervals. Smaller observation intervals will lead to erroneous estimates, which leads to longer execution times. Create CRIU performs a control point-this process. Create a checkpoint for the existing state of docker. Yet, it cannot control the changes made to the container by the Docker image. Create Docker checkpoint images: CRIU does not create checkpoints for data generated by the repository, so RBACS must manage them separately.

RBACS generates a new docker image from an operating container and copies all data created from that container. Its time taking process and the total cost depends on the data generated size. Transfer a checkpoint and a verified point docker image to the target computer: after Kubernetes reprogram the application on a node, the selected point, and the verified point docker image must be transferred to the target node. This can be expensive if the application is restarted on another node. In Fig. 8, this behavior is observed for Windows and Linux systems, where the transmission time is approximately 15–20 seconds.

Accurate resource assessment is a test for end clients because the prerequisites for the application can vary from time to time, as they are subject to various dependencies, such as information size, input parameter selection, optimization labels and application core. The proposed system investigates a better approach to expanding the container orchestra at kubernetes that dynamically estimates the size of the resources required by the application, does not migrate with difficulty, speeds up resource allocation, and proceeds with application implementation.

Since it provides abstraction at the infrastructure layer, kubernetes can run containers almost everywhere. Operating containers across various networks and services from cloud to virtual computers to bare metal increases scalability and simplifies sharing and decision-making. Instead of maintaining the underlying systems, DevOps will concentrate on developing software. It also has built-in redundancy mechanisms, such as high availability, automatic failover, and the ability to decommission, duplicate, and spin up new containers and facilities to essentially self-heal.

5 Conclusion

With the development of cloud computing and big data, the conception of virtual machines has been swapped by lightweight containers. It has a built-in job scheduler and synchronization function. However, the Kubernetes community has not yet reviewed key factors, such as accurate estimates of performance and resource disruption. RBACS offers an efficient solution for redistributing the source of applications executing in the kubernetes cluster. We compared RBACS with the latest kubernetes VPA, designed to allocate resources accurately. RBACS brings many benefits to kubernetes VPA by using an uninterrupted method for automatic scaling. This is a more accurate estimation method that can increase memory and cluster CPU usage by 9% and reduce the running time by 12%.

We determined the cost of container migration to RBACS at different stages of migration, established experiments to determine the optimal size of the storage monitoring interval for the account, compared the distribution method of RBACS and kubernetes VPA, and evaluated the value of RAC site measurement. We can see that the overall cost of runtime for each application varies from 4% to 18%, but these applications have an almost accurate distribution that allows the cluster to be used by other applications. RBACS also reduces the number of reboots using kubernetes VPA.

Acknowledgement: Thanks to our families and colleagues, who provided moral support.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding this study.

References

[1] F. Rossi, V. Cardellini, F. Presti and M. Nardelli, "Geo-distributed efficient deployment of containers with Kubernetes," *Computer Communications*, vol. 159, no. 3, pp. 161–174, 2020.

[2] G. Diouf, H. Elbiaze and W. Jaafar, "On byzantine fault tolerance in multi-master kubernertes clusters," *Arxiv*, vol. 109, pp. 407–419, 2019.

[3] A. Nuovo, S. Varrasi, A. Lucas, D. Conti, J. McNamara *et al.,* "Assessment of cognitive skills via human-robot interaction and cloud computing," *Journal of Bionic Engineering*, vol. 16, no. 3, pp. 526–539, 2019.

[4] H. Hamzeh, S. Meacham and K. Khan, "A new approach to calculate resource limits with fairness in kubernetes," in *Proc. of 1st Int. Conf. Digital Data Processing*, London, UK, pp. 51–58, 2019.

[5] M. Aly, F. Khomh and S. Yacout, "Kubernetes or openshift which technology best suits eclipse hono IoT deployments," *Proc. of IEEE Int. Conf. Server Computing Applications SOCA*, Kaohsiung, Taiwan, vol. 19, pp. 113–120, 2019.

[6] S. Kho and A. Lin, "Auto-scaling a defence application across the cloud using docker and kubernetes," *Proc. of 11th IEEE/ACM Int. Conf. Utility and Cloud Computing Companion, UCC Companion*, Zurich, Switzerland, vol. 2018, pp. 327–334, 2019.

[7] N. Parekh, S. Kurunji and A. Beck, "Monitoring resources of machine learning engine in microservices architecture," in *Proc. of IEEE Annual Information Technology Electronics and Mobile Communication Conf. IEMCON*, Burnaby, Canada, vol. 2018, pp. 486–492, 2019.

[8] J. Shah and D. Dubaria, "Building modern clouds: Using docker, kubernetes google cloud platform," in *Proc. of IEEE 9th Annual Computer Communication Workshop Conf. CCWC*, Nevada, USA, vol. 2019, pp. 184–189, 2019.

[9] P. Townend, "Improving data center efficiency through holistic scheduling in kubernetes," in *Proc. of 13th IEEE Int. Conf. Server System Engnieering SOSE 2019*, USA, vol. 23, pp. 156–166, 2019.

[10] F. Zhang, X. Tang, X. Li, S. Khan and Z. Li, "Quantifying cloud elasticity with container-based autoscaling," *Future Generation Computer System*, vol. 98, no. 1, pp. 672–681, 2019.

[11] A. Nadjaran, J. Son, Q. Chi and R. Buyya, "ElasticSFC: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds," *Journal of System Software*, vol. 152, no. 1, pp. 108–119, 2019.

[12] B. Ahmed, B. Seghir, M. Al-Osta and G. Abdelouahed, "Container based resource management for data processing on IoT gateways," *Procedia Computer Science*, vol. 155, no. 2018, pp. 234–241, 2019.

[13] A. Celesti, D. Mulfari, A. Galletta, M. Fazio, L. Carnevale *et al.,* "A study on container virtualization for guarantee quality of service in cloud-of-things," *Future Generation Computer System*, vol. 99, no. 1, pp. 356–364, 2019.

[14] K. Vhatkar and G. Bhole, "Optimal container resource allocation in cloud architecture: A new hybrid model," *Journal of King Saud University - Computer Information Science*, vol. 19, no. 1, pp. 159–163, 2019.

[15] P. Pathirathna, V. Ayesha, W. Imihira, W. Wasala, D. Edirisinghe *et al.,* "Security testing as a service with docker containerization," in *Proc. of 17th Int. Conf. of Advance ICT Emerging Regions*, Colombo, Sri Lanka, vol. 2018, pp. 1–18, 2018.

[16] M. Chen, W. Li, Y. Hao, Y. Qian and I. Humar, "Edge cognitive computing based smart healthcare system," *Future Generation Computer System*, vol. 86, no. 1, pp. 403–411, 2018.

[17] M. Chen, Y. Zhang, M. Qiu, N. Guizani and Y. Hao, "SPHA: Smart personal health advisor based on deep analytics," *IEEE Communication Magazine*, vol. 56, no. 3, pp. 164–169, 2018.

[18] Z. Cai and R. Buyya, "Inverse queuing model-based feedback control for elastic container provisioning of web systems in kubernetes," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 337–348, 2022.

[19] M. Sebrechts, S. Borny, T. Wauters, B. Volckaert and F. De Turck, "Service relationship orchestration: lessons learned from running large scale smart city platforms on kubernetes," *IEEE Access*, vol. 9, no. 1, pp. 133387–133401, 2021.

[20] L. Toka, G. Dobreff, B. Fodor and B. Sonkoly, "Machine learning-based scaling management for kubernetes edge clusters," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 958–972, 2021.

[21] M. Sebrechts, S. Borny, T. Wauters, B. Volckaert and F. De Turck, "Service relationship orchestration: Lessons learned from running large scale smart city platforms on kubernetes," *IEEE Access*, vol. 9, pp. 133387–133401, 2021.