Arbaz Shah (SE – 60)
Moaaz Siddiqui (SE – 76)
Syed Owais Ali (SE – 79)
Taha Hasan (SE – 301)

ENERGY EFFICIENT NETWORKIN WITH KUBERNETES

ABSTRACT

Kubernetes has emerged as a leading container orchestration platform, providing a powerful framework for automating the deployment, scaling, and management of containerized applications. With the increasing demand for energy-efficient networking solutions, Kubernetes plays a vital role in optimizing resource utilization and promoting sustainability. This article explores the detailed functionality of Kubernetes and highlights its importance in enabling energy-efficient networking through features like resource optimization, auto-scaling, and intelligent scheduling. By leveraging Kubernetes, organizations can achieve higher energy efficiency, reduce operational costs, and contribute to a greener and more sustainable IT infrastructure.

INTRODUCTION

In recent years, the rapid growth of digital infrastructure and the widespread adoption of microservices architecture have led to an unprecedented rise in containerized applications. However, this transformation has posed significant challenges in managing and scaling these applications efficiently. To address these challenges, container orchestration tools like Kubernetes have gained immense popularity. Kubernetes provides a comprehensive solution for automating the deployment, scaling, and management of containers across a cluster of machines. Beyond its fundamental benefits of scalability, high availability, and disaster recovery, Kubernetes also plays a crucial role in promoting energy-efficient networking.

Energy-efficient networking has become a critical consideration for organizations striving to reduce their carbon footprint and optimize resource consumption. By leveraging Kubernetes, organizations can achieve higher energy efficiency in their IT infrastructure through various mechanisms. These include intelligent resource allocation and optimization, dynamic auto-scaling based on workload demands, and smart scheduling algorithms. In this article, we will delve into the detailed functionality of Kubernetes and explore how its features contribute to energy-efficient networking. We will examine resource management, scalability, and scheduling aspects of Kubernetes, highlighting their importance in creating sustainable and eco-friendly IT environments.

LITERATURE REVIEW

In [1], the authors discussed the challenges involved in using container orchestration tools like Kubernetes in heterogenous HPC environments. A heterogenous HPC environment is one in which that consists of a variety of different types of hardware, such as CPUs, GPUs, and FPGAs. One challenge is that Kubernetes is designed for homogeneous environments, and it can be difficult to get good performance in heterogeneous environments. Another challenge is that Kubernetes can be complex to manage, and it can be difficult to ensure that workloads are running efficiently [1]. The authors proposed a few solutions to these challenges:

1. Produce different types of images and layers for various HPC applications. This can help to improve performance and efficiency by reducing the amount of time that it takes to start and stop workloads.
2. Potential changes in HPC schedulers. HPC schedulers can be used to manage workloads in a heterogeneous HPC environment. By making changes to the scheduler, it may be possible to improve performance and efficiency.
3. Significant improvements in energy management process. By making improvements to the energy management process, it may be possible to reduce the energy footprint of a heterogeneous HPC environment.

In [2], the authors discussed the design and implementation of a low carbon scheduling policy for Kubernetes orchestrator. The scheduler chooses which compute nodes to use based on how much carbon is emitted to generate electricity in the region where those nodes are located. Real-time APIs that report grid carbon intensity are available for an increasing number of regions, but not all regions around the world.They proposed to use renewable energy sources, migrate workloads to low carbon intensity regions, and reduce the number of idle nodes in a Kubernetes cluster, as the idle nodes consume energy.

In [3], the authors proposed a heterogeneous task allocation strategy for Kubernetes clusters. Their proposition used 3 techniques:

1. Task packing: The containers are packed into existing resources to optimize resource utilization.
2. Autoscaling: Size of cluster changes dynamically
3. Rescheduling: Reschedule underutilized VM instances to save cost and prevent task loss.

The Australian National Cloud Infrastructure (Nectar) evaluation of their methodology revealed that, when compared to the standard Kubernetes architecture, it might lower overall costs for various types of cloud workload patterns by 23% to 32%.

In [4], the researchers have proposed a new Kubernetes Container Scheduling Strategy (KCSS) that uses artificial intelligence (AI) to make more informed decisions about where to place containers. KCSS considers a variety of factors, such as the number of available resources, the workload of the node, and the user's requirements, to make the best possible decision.

KCSS has been shown to significantly improve the efficiency of Kubernetes scheduling. In one study, KCSS was able to reduce the average cost of running containers by 20%. KCSS also improved the performance of Kubernetes, as containers were able to start up more quickly and were less likely to be evicted from nodes.

KCSS is a promising new approach to Kubernetes scheduling. It has the potential to significantly improve the efficiency and performance of Kubernetes, while also reducing costs.

METHODOLOGY

KUBERNETES BASIC COMPONENTS

1. **Pod**: The smallest and most basic unit in the Kubernetes ecosystem is a pod. It stands for a set of one or more containers that are deployed collectively on a node. A Pod's containers can connect with one another using localhost and the same IP address. The Pod provides an abstraction over the containers and serves as a logical host for them.
2. **Node**: A Node is a worker machine in the Kubernetes cluster. It can be a physical server or a virtual machine. Nodes are responsible for running Pods and other Kubernetes components. Each Node has the necessary services to communicate with the master and manage the running Pods.
3. **Service**: A Service is an abstraction that defines a logical set of Pods and provides a stable network endpoint for accessing them. It acts as a load balancer and exposes the Pods to other services or external clients. Services provide a consistent way to access and discover Pods, even if their IP addresses or underlying infrastructure change.
4. **Ingress**: Ingress is an API object that manages external access to services within a cluster. It serves as the cluster's gateway or entry point and manages the routing of HTTP and HTTPS traffic to the relevant Services in accordance with the guidelines set forth in the Ingress resource. With just one external IP address, the cluster is able to give many services external access.
5. **ConfigMap**: ConfigMap is a Kubernetes resource used to store configuration data in key-value pairs. It allows decoupling configuration from the container images, making it easier to manage and update configurations without rebuilding the containers.
6. **Secret**: Another Kubernetes resource called Secret is used to store private data like SSH keys, tokens, and passwords. Secrets are base64 encoded and can be exposed as environment variables or mounted as files inside of Pods to provide secure access to sensitive data.
7. **Volume**: Volume is a directory that can be mounted into a container within a Pod. It provides a way to persist data beyond the lifetime of the container. Volumes can be backed by various storage types, including local disk, network-attached storage, or cloud-based storage solutions.
8. **Deployment**: A higher-level resource called deployment controls and specifies the ideal state for pods. It guarantees that a certain number of Pod clones are active at all times. The Deployment controller produces or updates Pods to reflect the intended state whenever a Pod fails or has to be updated.

THE NEED FOR CONTAINER ORCHESTRATION TOOLS

The rise of microservices and containerization led to the need for container orchestration tools like Kubernetes. Here are some reasons why Kubernetes is essential:

1. **Scalability**: Kubernetes enables effortless scaling of applications by automatically managing the deployment of additional instances (replicas) of containers based on demand, allowing applications to handle increased traffic and workloads.
2. **High Availability**: Kubernetes ensures high availability by automatically rescheduling containers or replicas to healthy nodes in case of node failures, ensuring that applications remain accessible and operational even in the presence of failures.
3. **Resource Optimization**: Kubernetes optimizes resource utilization by intelligently scheduling containers and efficiently packing multiple containers onto nodes, maximizing the utilization of compute resources.
4. **Service Discovery and Load Balancing**: Kubernetes provides service discovery and load balancing mechanisms, allowing applications to be exposed internally or externally via a stable network endpoint. This simplifies the configuration and management of network traffic routing to different containers or services.
5. **Fault Tolerance and Disaster Recovery**: Kubernetes supports declarative configuration and state management, making it easier to recover from failures. It allows for the automated creation of backups, monitoring, and recovery processes, ensuring fault tolerance and enabling disaster recovery strategies.

KUBERNETES NODES:

A- WORKER NODES

Worker nodes, also known as worker machines or worker servers, are the computing nodes that form the underlying infrastructure for running containerized applications. The processes running on worker nodes can be categorized into the following:

1. **Container Runtime**: On the worker node, containers are managed and run by the container runtime. Numerous container runtimes, including Docker, containerd, and CRI-O, are supported by Kubernetes. The container runtime downloads container images, builds and launches containers, and manages their resources and isolation.
2. **Kubelet**: Kubelet is an agent that runs on each worker node and communicates with the Kubernetes control plane. It receives instructions from the control plane and ensures that the containers specified in the pod manifests are running and healthy on the node. Kubelet monitors the node's resources, reports status to the control plane, and manages the lifecycle of containers on the node.
3. **Kube-proxy**: Kube-proxy is responsible for network proxying and load balancing within the cluster. It runs on each worker node and maintains network rules to forward traffic to the appropriate pods or services. Kube-proxy implements the Kubernetes Service abstraction, enabling transparent service discovery and load balancing for applications running on the worker nodes.

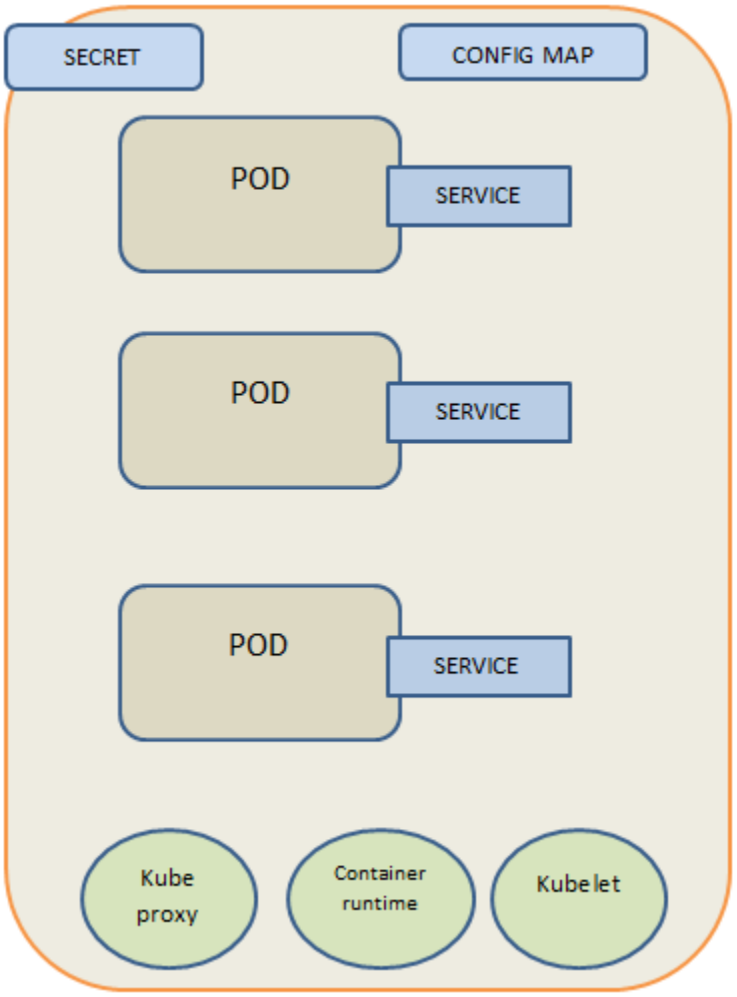


Fig.1 TYPICAL COMPONENTS AND PROCESSES IN A KUBERNETES NODE

B- MASTER NODES

The master node is responsible for managing and controlling the cluster. It comprises several components that work together to provide the control plane for the entire Kubernetes cluster. Each component performs specific tasks and contributes to the overall management and operation of the cluster. Here are the key components and processes running on the master node:

1. **API Server**: The API server is the central control point and primary interface for interacting with the Kubernetes cluster. It exposes the Kubernetes API, which allows users, administrators, and other components to communicate and manage the cluster. The API server validates and processes requests, enforces security and access controls, and maintains the desired state of the cluster.
2. **Controller Manager**: The controller manager is responsible for running various controllers that regulate the state of the cluster. Controllers monitor the cluster's desired state, compare it with the current state, and take actions to converge the two. For example, the Node Controller monitors the availability of nodes, the Replication

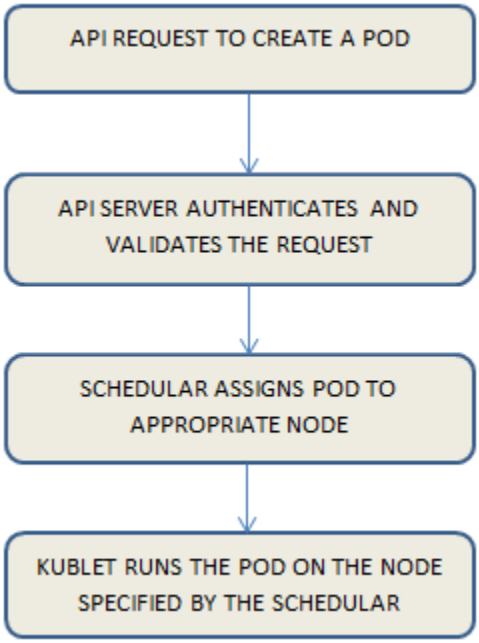
Controller manages the desired number of pod replicas, and the Service Controller ensures the stability and routing of services within the cluster. The controller manager runs these controllers and maintains the overall desired state of the cluster.

3. **Scheduler**: The scheduler is responsible for assigning pods to worker nodes based on resource requirements, quality of service constraints, and other policies. It evaluates the available resources on each worker node and determines the most suitable node for placing a pod. The scheduler takes into account factors like affinity and anti-affinity rules, resource availability, and workload constraints to make optimal scheduling decisions.
4. **etcd**: etcd is a distributed key-value store that serves as the cluster's central database for storing and retrieving configuration data and cluster state information. It provides a highly available and consistent data store that enables coordination and synchronization among the different components of the control plane. The etcd cluster is typically deployed on separate machines or as a distributed system to ensure fault tolerance and data durability.

These processes work collaboratively to manage the cluster's state, facilitate communication, and enforce desired configurations. The API server acts as the primary interface for external entities and internal components to interact with the cluster. The controller manager continuously monitors the cluster's state and takes actions to reconcile any discrepancies. The scheduler ensures efficient resource allocation and pod placement decisions based on various constraints and policies. The etcd cluster provides a reliable and consistent data store for maintaining the cluster's configuration and state.

PROCESS FLOW OF CREATING, SCHEDULING, AND EXECUTING A POD

1. The control manager includes various controllers responsible for managing different resources in the cluster. When a request to create a pod is received, the relevant controller (in this case, the ReplicaSet or Deployment controller) in the control manager creates a desired state for the pod based on the requested specifications. The desired state includes information like the number of replicas, container image details, resource requirements, and other configuration settings.
2. The scheduler is responsible for assigning pods to worker nodes based on resource availability, constraints, and policies. It evaluates the available resources on each worker node and selects the most suitable node for the pod. The scheduler takes into account factors such as resource requirements, affinity rules, anti-affinity rules, node taints, and tolerations to make an optimal scheduling decision. Once the scheduler determines the node for the pod, it updates the desired state of the pod with the assigned node information.
3. After the scheduler assigns the pod to a specific worker node, the kubelet running on that node is responsible for executing and managing the pod. The kubelet communicates with the API server to retrieve the pod's configuration and instructions. It then pulls the necessary container images from the container registry, creates and starts the containers within the pod, and monitors their status. The kubelet also reports the pod's status back to the API server and handles tasks like restarting containers in case of failures or scaling up/down the replicas if needed.



PROCESS FLOW OF CREATING, SCHEDULING AND EXECUTING A POD

CONCLUSION

Kubernetes is a popular container orchestration system that can be used to manage a variety of workloads. However, Kubernetes's default scheduling algorithm is not optimized for energy efficiency. This can lead to increased energy costs and environmental impact. Therefore, proper measures need to be taken to reduce the energy consumption and environmental impacts of running large scale kubernetes clusters.

REFERNECES

[1] V. Dakic, M. Kovac, and J. Redzepagic, "Optimizing Kubernetes Performance, Efficiency, and Energy Footprint in Heterogeneous HPC Environments," Annals of DAAAM & Proceedings, vol. 10, no. 2, 2021.

[2] A. James and D. Schien, "A low carbon Kubernetes scheduler," in ICT4S, Jan. 2019.

[3] V. Dakic, M. Kovac, and J. Redzepagic, "Optimizing Kubernetes Performance, Efficiency, and Energy Footprint in Heterogeneous HPC Environments," Annals of DAAAM & Proceedings, vol. 75, pp. 1-7, 2021.

[4] Jorge-Martínez, D., Butt, S. A., Onyema, E. M., Chakraborty, C., Shaheen, Q., De-La-Hoz-Franco, E., & Ariza-Colpas, P. (2021). Artificial intelligence-based Kubernetes container for scheduling nodes of energy composition. International Journal of Systems Assurance Engineering and Management, 12(3), 333-344. doi:10.1007/s13198-021-01195-8