The Analysis Of The Content Delivery Network Application Deployment On Kubernetes Platform

Pooja S¹, Meeradevi², Monica R Mundada³, Palaniappan Ramanathan⁴

¹Department of Computer Science and Engineering, Ramaiah Institute of Technology, Bangalore, India.

²Department of Computer Science and Engineering, Ramaiah Institute of Technology, Bangalore, India.

³Department of Computer Science and Engineering, Ramaiah Institute of Technology, Bangalore, India.

⁴Intel Technology India Pvt. Ltd., Bangalore, India.

Abstract

Kubernetes is one of the platforms which provides different features that help in the deployment of applications. This platform provides different ways to enhance the application’s capability. With the better exploration of these ways, we can provide the best Content Delivery Network (CDN) services. Some of the important Kubernetes features that are useful in the deployment of CDN application is highlighted in this paper. The aim of this paper is the analysis of the CDN application deployment and the demonstration of high-speed data delivery from the CDN application. We deployed a single-node Kubernetes edge cluster on the physical machine using Intel’s Open NESS software toolkit. We installed the NGINX-based CDN applications on the Kubernetes cluster. To achieve higher speed, the Single Root I/O Virtualization (SR-IOV) interface is used between the edge node and the CDN client machine. The performance measurements of the CDN application show higher data delivery speed.

Keywords
Kubernetes, CDN, Application Deployment, NUMA, Device Plugin, SR-IOV.

**Introduction**

The management software Kubernetes has reduced the complications involved in the deployment of containerized applications. It’s command-line interface kubectl helps in debugging the deployment issues. The performance of an application can be optimized with the proper tuning of the configurations. Kubernetes provides ways to improve the performance of the deployed application. This includes changing the policies, allocating resources in a certain way etc., This open-source platform eases the operation of the software. Some of the benefits of the Kubernetes platform are the automation of operations, the management of storage and networking. It also takes care of scaling the applications and monitoring the health of services. It gives us many options to control the working of the application. The deployment and enhancement method discussed in this paper is helpful for other applications as well.

The Content Delivery Network (CDN) services are used by many Over The Top (OTT) platforms today. Nowadays, the majority of web traffic has been served through CDN. The CDN involves a geographically distributed group of servers that provide the faster delivery of the cached content to the clients closer to the cache servers. The CDNs surely improve the performance of the website by timely delivery of the content. The service provider has to provide proper web hosting. The website is hosted on one platform and CDN provides the proxies to distribute the content efficiently all over the network. Some of the sites that use CDN services are Facebook, Netflix, Amazon. For improved speed and connectivity, the CDN servers are placed at Internet Exchange Points, Edge locations. Some of the benefits of the CDN service include decreasing the website page load time, reducing the total amount of data the origin server has to provide. This surely reduces the bandwidth cost. CDNs also improve security by providing mitigation for Distributed Denial Of Service (DDOS) attacks.

Edge computing brings the computing of data close to the user to reduce latency and bandwidth. Open Network Edge Services Software (Open NESS) is a software toolkit that helps in the deployment of services on various platforms that use different access mediums. It reduces the deployment impedance faced by Independent Software Vendors (ISVs), Independent Hardware Vendors (IHVs), Network Operators. I have used licensed version of Open NESS software for deployment. The main contributions of our work are: 1) The analysis of the deployment of CDN application on Kubernetes platform 2) The NUMA alignment of resources of the CDN application 3) The deployment methodology to achieve high-speed data delivery from the CDN application using SR-IOV interface.
The remainder of this paper is organized as follows. Literature Review discusses the related work and the analysis of the CDN application deployment factors which is important in the deployment of the CDN application on the Kubernetes cluster. Methodology section gives the details of the proposed NGINX (Engine-X) based CDN application deployment with SR-IOV enabled. Results analysis presents the analysis of the performance measurements, followed by the conclusion and future work.

**Literature Review**

The content distribution system involves placing content servers, selection of suitable servers for the specific content and the content replication. Meenakshi Gupta et al. (2014) reviewed the strategies for content replication and the working of content delivery networks. They have explained how CDN is an effective approach to reduce the congestion on the network and the response time. The surrogate server placement, content consistency management, content replication methods are discussed. Sachchidanand Singh et al. discusses the benefits of container-based virtualization, they give an overview of the docker container and the use of cloud technology in hosting them. They have compared the monolithic and microservices architecture. They have discussed how container-based virtualization helps the provisioning of the applications in the cloud.

Leila Abdollahi Vayghan et al. examines the availability of the application in the Kubernetes environment and have conducted experiments related to service availability. They explain the architecture of the system for deploying containerized applications in the cloud using Kubernetes platform. The Kubernetes tool can be used with clusters deployed in the private and public clouds. Kubernetes acts as an orchestrator and helps in self-healing, configuration management and service discovery. The authors have discussed pod failure and node failure scenarios for the streaming application deployed in the Kubernetes cluster. Yuedui Wang et al. explains the architecture of the CDN systems based on cloud systems. There are different types of file systems available for storage. The performance of the underlying drive determines the performance of the overall system. Guangyu Zhu et al. evaluates the performance of traditional file systems and Non-Volatile Memory (NVM)-aware file systems on Intel Optane DCPMM and NVM e SSD. They have also highlighted their findings for simple read/write operations and complex workloads.

With the increasing number of mobile devices, we can see a huge increase in the data that is being generated every day. The computation of this data is a complicated task. Though everyone has a sophisticated phone nowadays, sometimes the application has to be offloaded to either cloud or edge for complicated computation. This computation needs hardware accelerators such as Graphics Processing Unit (GPU), Field Programmable Gate Array (FPGA) for faster results and better performance of the application. Sunghyun Kim
et al. have proposed a Kubernetes-based cloud framework that uses hardware accelerators. It ensures that the container is deployed in the proper node where the accelerator is installed. This framework keeps the container lightweight and also improves performance by accelerator-aware service scheduling. The GPUs are needed for gaming applications. The entire game or some levels of the game is cached in CDN storage.

The 5G technology has some significant features that cater to the need of the data-intensive, latency-sensitive applications like CDN, Live TV, Virtual Reality, and Augmented Reality. N. Calabretta et al. investigate the SDN-controlled network at Edge that has 5G traffic. The SDN controller slice the traffic based on requirements, allocate physical/logical resources to different slices. The cloud providers manage the configuration and Kubernetes infrastructure freeing the users of the management burden. Arnaldo Pereira Ferreira et al. evaluate the performance of the containers running in Amazon, Google, Microsoft cloud. They compare Google Kubernetes Engine (GKE), Amazon’s Elastic Kubernetes Container Service (EKS), Microsoft Azure Kubernetes Service (AKS). Robert Botez et al. give the implementation of an application based on microservices. Each microservice runs in a separate docker container. The software is orchestrated by Kubernetes and hosted on the Google Cloud Platform.

In most of the above-discussed papers, the authors have deployed the Kubernetes cluster and application in the cloud. Hence, they have not addressed the system features in detail. We have deployed the Kubernetes cluster in the physical machine and installed CDN applications on the cluster. We have presented a detailed analysis of the deployment of Kubernetes cluster in the physical machine in this paper. The data transfer speed of the system is not addressed in most of the papers. This paper focuses on data transfer speed which is one of the most important requirements for the streaming applications like CDN. For a higher data transfer rate, the SR-IOV interface is used between the edge node and the client machine.

1) Analysis of CDN Application Deployment Factors

The content delivery network is used to improve the website rendering speed and performance. To minimize the distance between the visitors and the website’s server, a CDN server stores the cached version of its content in multiple geographical locations. The CDN servers perform Central Processing Unit (CPU) intensive, disk-intensive, network-intensive and memory-intensive tasks to serve the video on demand and for the live stream. The CDN applications being latency and throughput intensive, require dedicated compute, memory, input/output resources and sophisticated mechanisms. These servers need Multi-core CPUs for parallel processing, High-speed RAM-DDR4 and Optane Memory, High-Speed Disks - Multiple SSD and High-speed interfaces – NVM e, high-Speed Network.
The CDN application with many benefits is one of the most required applications which runs on the edge servers. We have analysed the different deployment options available for the CDN application in the Kubernetes platform. The below discussed method improves the performance of the CDN applications.

- **Non-Uniform Memory Access (NUMA)**

The term NUMA stands for Non-Uniform Memory Access. It is a memory system design available on multiple CPU systems that allows different CPUs to access different parts of memory at different speeds.

A NUMA (Byonggon Chun., 2019) node is formed by the local memory, CPU cores and peripheral devices (disks, network interface cards) that are local to the CPU. The memory, peripheral devices that are directly connected to the CPU is considered as “local” to that CPU and can be accessed very fast. The memory that is not directly connected to the CPU is considered as “non-local” and will have variable access times. The access time depends on how many interconnects must be passed to access the memory. In Figure 1, the CPUs 0-3 are said to be part of NUMA node 0, whereas CPUs 4-7 are part of NUMA node 1. Likewise, GPU 0 and NIC 0 are part of NUMA node 0 because they are attached to socket 0 that is connected to CPUs 0-3. The GPU 1 and NIC 1 are part of NUMA node 1 and are connected to socket 1. Socket 1 is connected to CPUs 4-7.
In Figure 2, we can see that the application is not aligned properly with a NUMA node. The application is using CPU 0 that belongs to NUMA node 0 and peripheral devices A1, B1 that belongs to NUMA node 1. If an application uses resources of different NUMA nodes, then the application is said to be “not NUMA aligned”. This adversely affects the performance of the application because the application has to access non-local peripheral devices for every operation of those devices. For better performance, it’s recommended for an application to use the resources (memory, CPU, peripheral devices) attached to the single NUMA node as shown in Figure 3. The resources are NUMA aligned as the application is using CPU1, peripheral devices A1 and B1 which are on the same node- NUMA node 1. Ensuring NUMA alignment decreases the memory access time thus improving the performance of the CDN application. The CPU1, peripheral devices A1 and B1 are directly connected. We have ensured NUMA alignment of the resources for the better performance of the deployed CDN applications. The steps for ensuring NUMA alignment are discussed in Methodology section.

- **Device Plugins**
It is not a sustainable solution to expect every hardware vendor to add their vendor-specific code inside Kubernetes to make their devices usable. The vendors should be able to advertise their resources to kubelet and monitor them. Kubernetes provides vendors a mechanism called “device plugins” to advertise their devices, to monitor them by performing frequent health checks, to execute the device-specific instructions and to make sure that the device is ready or available in the container. The information provided by device plugins gives all the required information about the devices available in the system to the device manager which in turn interacts with Kubernetes Topology Manager. This helps in ensuring NUMA alignment of resources which improves the performance of the application. The kubelet component – “Topology Manager” helps the resources to be aligned at the pod and container level in a certain way. The program and all its dependencies form a container. The pod is the smallest deployable unit of computing in Kubernetes. A pod is a group of one or more containers with shared resources. The topology manager policy determines the strategy of alignment of resources. The supported policies are None(default) – It is the default policy and does not try to ensure any alignment, Best-effort – If possible, the resources will be allocated in a preferred way, else the pod is assigned the available resources, Restricted – If the preferred allocation is not possible, the topology manager rejects the pod. The resources are not allocated for the pod and the application is not started in the system. Single-numa-node - The topology manager tries single NUMA node alignment for that container. It tries to allocate resources from any single NUMA node. If it’s not possible, the topology manager rejects the pod. The resources are not allocated for the pod and the application does not start in the system.

For this work, during the deployment of the Kubernetes cluster, we had set the topology manager policy to “single-numa-node”. Hence the resources for the CDN applications installed were allocated from a single NUMA node- NUMA node 0. The Topology Manager feature of Kubernetes enables NUMA alignment of CPUs and peripheral devices, allowing the workload to be run in an environment optimized for low latency. The device plugins give all the required information to the device manager. It provides the NUMA topology hints to the device manager. The “device manager” is the kubelet’s component that interacts with the topology manager and makes resource assignment decisions.

After successful registration of the device plugin, the kubelet interacts with the device plugin using these two functions:

- A List And Watch function helps the kubelet to discover the devices and their properties. The device plugin advertises the list of available devices as well as notifies of any device status change like devices getting unhealthy.
An Allocate function is called before creating a user container that needs the exported devices. When creating containers, the kubelet calls the device plugin's allocate function before so that it can run device-specific instructions and instruct kubelet how to make the device available in the container.

With these features, the operator need not be aware of the NUMA constraints for the devices. The application can be deployed automatically and the NUMA alignment is ensured by Kubernetes. Intel’s SRIOV network device plugin has this extension. We have used this device plugin to enable Single Root I/O Virtualization (SR-IOV) on the edge node. The SR-IOV interface is used to achieve the higher speed of data delivery. A detailed explanation of enabling SR-IOV is given in next section.

**Methodology**

1) **Proposed CDN Application Deployment on Kubernetes Cluster**

NGINX is an open-source software used for caching, web serving, load balancing, reverse proxying, media streaming. It is designed to provide maximum performance and stability. The lab setup for the NGINX based CDN application is as shown in Figure 4. We deployed the single-node Kubernetes cluster on the physical machine using the Open NESS software toolkit. Intel’s Open NESS software enables the deployment of edge services (Dario Sabella, 2019) on various platforms. It supports extended functionalities, adding capabilities to existing edge platforms. It supports all the required capabilities for the edge node and controller. The Kubernetes acts as the orchestrator for the applications that run on the cluster. In the single-node cluster, a single physical machine acts as both edge controller and edge node. The functioning of the edge node is controlled by the edge controller. The cluster deployment creates kube-system pods, harbor pods. The kube-system pods perform the system functions for the cluster and the harbor pods are used to store the docker images. The docker (Jay Shah, 2019) engine is the container runtime that provides the components required to run and manage the containers. We used a high configuration physical machine for deployment with RAM - 376GB, Disk – 385G and 95 CPU cores, OS-centos-7.9-min. But it is not necessary to have a higher configuration machine. The minimum configuration of the machine recommended is RAM - 32 GB, Disk - 20GB and 8 CPU cores. The physical machine that was used for deployment had two NUMA nodes. The Kubernetes topology manager policy was set to “single-numa-node” during the deployment of the cluster. The two important parameters that were set while deploying the cluster are - Container Network Interface (CNI) and topology manager policy. These parameters were specified in the cluster deployment file. The Kube-OVN and SR-IOV CNIs are enabled for the cluster. The configuration of the content server machine is RAM - 16 GB, Disk – 40GB, 4 CPU cores.
We installed the NGINX based CDN application on the edge node using the helm chart. The helm chart is the collection of files that describe the deployment resources. The chart template consists of different files that can be modified based on our requirements. The various parameter values for the CDN application are given using the values. yaml file. There are separate template files like policy. yaml, deploy. yaml, service. yaml, configmap. yaml for defining application policy, deployment, service and configuration. These files take values from values. yaml file. The content server stores the original data that would be transmitted. Its IP address is given in the values. yaml file. We can modify the values in the values. yaml file and install the CDN application. The client machine and edge node are connected with the network interfaces as shown in Figure 4. When the client machine sends the request for the content, the content would be fetched from the content server for the first time. The content is delivered to the client and also cached in the edge node. If the same content is requested by the client again, it is delivered directly from the edge node. The different values of the CDN application as defined in values. yaml file is given in Figure 5, which can be modified for every installation. We installed two CDN applications, first application - CDN application1 with SR-IOV not enabled. In this setup, we used the lab-bridge interface to access cached content. This interface is indicated as No.1 in Figure 4. The second application - CDN application2 with SR-IOV enabled. In this setup, the SR-IOV interface is used to access the cached content. This interface is indicated as No.2 in Figure 4. The NUMA alignment is ensured for both the CDN applications.
Figure 5 CDN application values. yaml file

```
[root@localhost -]# helm install -f ~/edgeapps/applications/cdn-caching/nginx/helm/values.yaml cdnapplabinterface01 ~/edgeapps/applications/cdn-caching/nginx/helm/
NAME: cdnapplabinterface01
LAST DEPLOYED: Wed May 26 20:26:20 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figure 6 Installation of CDN application1

The Figure 6 shows the deployment of the CDN application1(cdnapplabinterface01). The IP addresses and machine names in the figures are masked for security purposes. The client machine and edge node are connected using the lab interface. The SR-IOV is not enabled for application1. We can see that the CDN application is running in the default namespace in the Kubernetes environment. With the installation of the CDN application, two CDN services get created. The service running in port 30007 serves the HTTP requests and the service running in port 30008 serves the HTTPS requests. The network policy of the CDN application is deleted to allow client access.

```
[root@localhost -]# helm install -f ~/edgeapps/applications/cdn-caching/nginx/helm/values.yaml cdnapplabinterface01 ~/edgeapps/applications/cdn-caching/nginx/helm/
NAME: cdnapplabinterface01
LAST DEPLOYED: Wed May 26 20:30:15 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figure 7 Installation of CDN application2

The Figure 7 shows the installation of the CDN application2(cdnappsriovinterface01). The SR-IOV is enabled for this application. The values passed for the installation of this application is given in Figure 5. First, we have to check whether the machines (client & edge node) support SR-IOV. Kubernetes has the SR-IOV device plugin. We need to ensure the availability of SR-IOV NIC in the Edge node. The SR-IOV functionality of the edge node is enabled using the Open NESS toolkit. During the Kubernetes cluster deployment, we have to specify the container network interfaces (CNIs) - SR-IOV, Kubeovn in the
cluster deployment yaml files. The required number of virtual functions should also be specified. This creates the platform that supports SR-IOV when the Kubernetes cluster is deployed. Node Feature Discovery (NFD) is a Kubernetes add-on that detects and advertises hardware and software capabilities of a platform and this facilitates the scheduling of a workload. We have to specify the node selector to use this feature to utilize different peripheral devices that are available in the nodes of the system. For this purpose, the following kubectl command is executed after the deployment of the Kubernetes cluster.

```
kubectl label nodes <node-name> feature.node.Kubernetes.io/network-sriov.capable=true
```

We have defined the following application requirement in the CDN deployment file – “only the node that supports SR-IOV should be selected to run the CDN application”. The Kubernetes scheduler selects the node after matching the application requirement. If the match is not found, the pod won’t be scheduled. In our deployment case, only one physical machine (edge node) was available in the cluster for the application to run since it’s a single node cluster. This edge node was labelled using the above-given command. This labels the edge node indicating that the node supports SR-IOV. The NGINX based CDN application2 is installed on the cluster. The Kubernetes scheduler schedules the CDN pod to run on the edge node.

With the help of SRIOV device plugin and the SR-IOV CNI in the edge node, the NIC is allocated which in turn allocates the specified virtual functions. These virtual functions directly communicate with the application container and give high speed. During cluster deployment, four virtual functions for the SR-IOV interface were created. The NIC allocated for the application is connected to the same socket, on which the nginx CDN application pod is scheduled to run. The NIC, CPU are connected to NUMA node 0 in the physical machine. The CDN application is using the resources of the single NUMA node – “NUMA node 0” ensuring NUMA alignment. The high-speed data delivery is achieved using the SR-IOV interface.
In Figure 8, we can see that both the CDN application pods are running in the default namespace. The harbor, kube-system pods of the cluster are running in harbor and kube-system namespaces respectively. The information is fetched using the command `kubectl get pods -A`. Figure 9 shows the information about two CDN services. The information is fetched using the command “kubectl get svc”. The CDN service with SR-IOV enabled is running on 30011(HTTP) and 30012(HTTPS) ports.

**Results Analysis**

The wrk tool is used for taking performance measurements. It’s a HTTP benchmarking tool capable of generating significant load. This tool is installed in the client machine to check the performance of the CDN applications running on the edge node. The file info1.html of size 2.7 MB is stored in the content origin server. The content was cached in the edge node when it was fetched for the first time. The cached content is accessed from the client machine. The connections(c) are evenly distributed among threads(t).
Figure 10 The wrk output1

Figure 10 shows the wrk output1 of the CDN application1. The CDN services corresponding to application1 are running on ports 30007 and 30008 as shown in Figure 9. The CDN service (HTTP) running on port 30007 is accessed from the client machine. The wrk command given in Figure 10 is executed on the client machine. The benchmark is run for 20 seconds (s) using 20 threads (t), keeping 20 connections (c) open. The HTTP requests are sent from the client machine. The network speed of the connection depends on the network interface used. The maximum lab network interface speed was 125MB/sec. The data transfer speed available as measured using wrk tool was 110 MB/sec (880Mbps). All the performance readings can be seen in Figure 10.

![Figure 10](image1)

Table 1 The comparison of the performance of CDN applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Interface</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDN Application 1</td>
<td>Bridge</td>
<td>110 MB/sec</td>
</tr>
<tr>
<td>CDN Application 2</td>
<td>SR-IOV</td>
<td>1.09 GB/sec</td>
</tr>
</tbody>
</table>

Figure 11 shows the wrk output2 of the CDN application2. The SR-IOV is enabled for this application. The CDN services corresponding to application2 are running on ports 30011 (HTTP) and 30012 (HTTPS) as shown in Figure 9. The CDN service (HTTP) running on port 30011 is accessed from the client machine. The wrk command is executed on the client machine. The benchmark is run for 20 seconds using 20 threads, keeping 20 connections open. The HTTP requests were sent from the client machine. The high-speed

![Figure 11](image2)
SR-IOV network interface was used for the connection between the edge node and client machine as indicated by No.2 in Figure 4. The maximum SR-IOV interface speed available was 10000Mbps. The data transfer speed as measured using wrk tool was 1.09GB/sec(8720Mbps). The comparison of speed is given in Table1. There is a huge increase in the speed of the delivery of data from the CDN application2 compared to CDN application1. The difference in the speed for the two applications is due to the network interfaces used. CDN application 1 uses bridge interface and CDN application 2 uses high-speed SR-IOV interface.

**Conclusion and Future Work**

Kubernetes provides the platform for application deployment and management. We have explained some of the Kubernetes features that help in the deployment of the application in detail. The CDN application carries out operations that are latency-sensitive, network-intensive and resource-intensive. So, we need to ensure that each pod has the best configuration and the best possible resources are allocated for the CDN service. In this paper, we have discussed how ensuring NUMA alignment increases the performance of the CDN application, the importance of the Kubernetes device plugin mechanism. We deployed single-node Kubernetes cluster and installed NGINX based CDN applications on it. We achieved high-speed data delivery from the CDN application using the SR-IOV interface. The future work includes the deployment of other CDN applications based on Varnish, Apache Traffic Server using different interfaces on Kubernetes platform and comparison of the performance of the applications. We want to explore the node level constraints of the cluster that impacts the performance of deployed applications and carry out a detailed study of available Kubernetes device plugins to explore how these plugins can give more information about the peripheral devices and help in device management, allocation of pod resources.

**References**


and lessons learned. IEEE 11th International Conference on Cloud Computing, 970-973, San Francisco, CA, USA. https://doi.org/10.1109/CLOUD.2018.00148


N. Calabretta., B. Pan., E. Magalhaes., F. Wang., X. Xue & N. Tessema. (2019). SDN controlled metro access network with network slicing and edge computing for 5G applications. Asia Communications and Photonics Conference, Chengdu, China


