Project Report

Kubernetes Dynamic Volume Provisioning Benchmark

CS-550 Advanced Operating Systems

May 2023

Under the Guidance of Phd lead - Meng Tang

By

Prajakta Kumbhar
and
Amith Gorthi Srinivasa Prabhakara Narasimha

1. Introduction:

In today's world, the cluster storage systems are continuously pressured for storage space. The ever-growing data due to rapid increase in High Performance Computing (HPC) Systems has raised issues like the IO bottleneck problem.

The capacity to dynamically provision containers to run large-scale applications has been significantly improved by the advent of container orchestration tools like Kubernetes. Resource management, load balancing, disaster recovery, networking is some of Kubernetes's usages which can be valuable for building High Performance Computing Systems (HPCs). Considering the above points, we are motivated to study the storage performance of Kubernetes by dynamic volume provisioning to explore how it can be utilized in High Performance systems.

2. Background information:

Kubernetes is an open-source container orchestration engine for automating deployment, scaling, and management of containerized applications. Kubernetes, as mentioned earlier, enhances its ability to dynamically provision containers to run large-scale applications.

Google Cloud is a cloud computing platform provided by Google. It provides a range of infrastructure, platform, and software services that can be used to build and run applications, store and analyze data, and manage IT operations. Google Cloud Storage (GCS) CSI (Container Storage Interface) driver is a component of the Kubernetes ecosystem that allows you to use Google Cloud Storage as a persistent storage solution for your Kubernetes applications. With the GCS CSI driver, you can use GCS buckets as persistent volumes for your Kubernetes pods. This allows you to store data outside of your pod and make it available to your application even if the pod gets terminated or rescheduled. Some benefits of using GCS CSI driver include easy management of data backups and disaster recovery, automatic scaling of storage resources, and the ability to use GCS features such as object lifecycle management and object versioning for your Kubernetes workloads.

Intel PAT (Performance Monitoring Unit Advanced Topics) is a tool provided by Intel that allows software developers to monitor the performance of their applications at a very fine-grained level. Specifically, it enables the monitoring of events that occur at the micro-architectural level of the CPU, such as cache misses, branch mispredictions, and retired instructions. One potential use case for Intel PAT is in machine monitoring. By monitoring the performance of a machine over time, developers can identify patterns and trends that may be indicative of underlying issues or problems.

We propose to study and run Kubernetes storage volumes using its Container Storage Interface (CSI) by integrating it with GoogleCloud file system. The performance of parallel storage systems can be evaluated utilizing a variety of interfaces and access patterns using the parallel IO benchmark known as IOR. By integrating Kubernetes with IOR at application side we will benchmark the storage performance. This storage performance will then be compared against default linux file systems. By using Intel PAT, we can gain insight into the performance characteristics of their applications and identify potential bottlenecks that may be slowing them down. We will be using the intel PAT tool to analyze the performance of the CPU.

3. Problem statement:

The study of dynamic volume provisioning will plot an idea of performance of Kubernetes in HPC's. By studying the dynamic volume provisioning of Kubernetes, we can understand how well it can be utilized in High Performance systems.

4. Related work:

The [1], evaluates the results of the proposed hyperconverged system as compared with a traditional configuration using local storage, with the intention of verifying the viability of its use in place of direct-attached storage. The evaluation of performance of container's storage is done by running an hyperconverged system inside Docker container. In [2], they have described a hybrid architecture consisting of an HPC cluster and a Cloud cluster, wherein the Cloud cluster's Kubernetes container orchestration can manage containers on the HPC cluster. The study in [3] investigates how Kubernetes might be used by the scientific community on HIPC infrastructure. A direct contrast of its capabilities and performance with those of HPC application execution on bare metal and Docker Swarm is done. In this paper [4], the authors examine the I/O performance of different container solutions on OrangeFS, a parallel storage system, specifically Singularity, Docker, LXC, Podman, and Charlie-Cloud. They deploy five containers and use a single server-single client OrangeFS setup to test the I/O performance of each container. The authors use Intel's Performance Analysis Tool on both the server and client nodes to measure the performance of the host machine on CPU and memory utilization, disk I/O, and network throughput.

5. Proposed Solution:

In this study we are proposing to study the storage performance of dynamic volumes of Kubernetes by integrating with DFS backends like Google Cloud using Kubernetes's Container Storage Interface (CSI).

Below is a high-level view of how we plan to integrate all these systems:

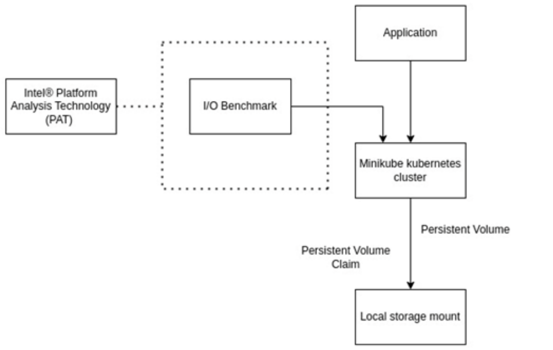


Figure 1 Local Storage Mount

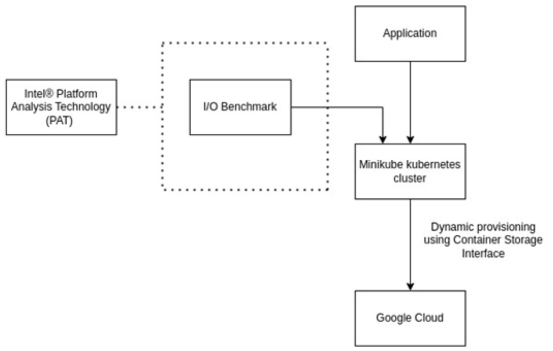

Application
Intel® Platform Analysis Technology (PAT)
I/O Benchmark
Minikube kubernetes cluster
Dynamic provisioning using Container Storage Interface
Google Cloud

Figure 2 Dynamic Storage Mount

To analyze the performance, we used tools like io benchmark tool and Intel PAT. To mount the local volume to the cluster we created a persistent volume and persistent volume claim and deployed the yamls for pod creation.

To mount the Google cloud volume to the minikube cluster we created a google cloud bucket and integrated it with the cluster using kubernetes Container storage interface.

6. Methodology:

Integration of Google Cloud with Kubernetes using CSI drivers

a. The CSI drivers enable Kubernetes to dynamically provision google cloud volumes and mount them to containerized workloads.

Evaluation of storage performance using IOR benchmark

a. Benchmark the storage performance.

Analysis of performance using Intel PAT

a. The performance characteristics of applications and identify potential bottlenecks that may be slowing them down.

7. Test Results

Testing strategy - For understanding the performance of both the systems under different ior conditions, testing for mpi and ior was done by varying the parameters like - mpi -np, ior -b and -t.

Here's what each parameter does:

- -np 1 specifies the number of MPI processes to use, in this case, one.
- -b specifies the block size to use for I/O operations in bytes. A larger block size generally results in better performance for sequential I/O operations, while a smaller block size can be better for random I/O operations.
- -t specifies the transfer size to use for I/O operations in bytes. This parameter determines the size of the data transferred in each I/O operation.
- -i specifies the number of iterations to perform for each I/O operation. Multiple iterations are performed to ensure accurate measurement of the I/O performance.

Step 1: On the local volume of kubernetes, first, for getting a blocksize which gives highest bandwidth, we kept the mpi processes to 1 and the transfer size as 1k constant and varied the block sizes like - 4k, 16k, 64k, 256k. We found that with mpi = 1, b= 64k and transfersize = 1k, gave the maximum bandwidth for both the systems.

Step 2: For the second variation, the block size was then kept constant b = 64k, and the transfer size was varied from 4k to 256k, by keeping mpi =1. We could see in this case the maximum bandwidth was again seen at t=64k.

Step 3: Thirdly the mpi was changed as mpi -np 2 and carried the step 1 and step 2 again. It was seen that for these the b=64k and k=64k gave the highest bandwidth.

Step 4: For the batch to get the best bandwidth, we tried changing the mpi processes. First -np was set to 1 and step 1,2&3 were performed. Then np was set to 2 and steps 1,2 &3 were performed, and finally -np was set to 4 and step 1,2 &3 were performed.

Step: 5: Step 1 to 4 were then performed on Google cloud storage.

Based on the results for all the above-mentioned combinations, the best of 9 test cases having the highest bandwidths were selected. Below are those 9 test cases for both the systems.

9 cases having highest bandwidth for local volume of kubernetes	9 cases having highest bandwidth for google CSI volume
mpirun -np 1 ior -b 4k -t 1k -i 3	mpirun -np 1 ior -b 4k -t 1k -i 3
mpirun -np 1 ior -b 16k -t 1k -i 3	mpirun -np 1 ior -b 16k -t 1k -i 3
mpirun -np 1 ior -b 64k -t 1k -i 3	mpirun -np 1 ior -b 64k -t 1k -i 3
mpirun -np 1 ior -b 64k -t 4k -i 3	mpirun -np 4 ior -b 64k -t 4k -i 3
mpirun -np 1 ior -b 256k -t 16k -i 3	mpirun -np 4 ior -b 64k -t 16k -i 3
mpirun -np 1 ior -b 64k -t 64k -i 3	mpirun -np 4 ior -b 256k -t 64k -i 3
mpirun -np 1 ior -b 64k -t 64k -i 3	mpirun -np 1 ior -b 64k -t 64k -i 3
mpirun -np 2 ior -b 64k -t 64k -i 3	mpirun -np 2 ior -b 64k -t 64k -i 3
mpirun -np 4 ior -b 64k -t 64k -i 3	mpirun -np 4 ior -b 64k -t 64k -i 3

Using these cases, we then ran PAT analysis further to analyze what systems performance for these cases.

From all of the above-mentioned tests, the test with -np=4, -b =64k and -t =64k was the one with the highest performance. We then evaluated these 2 conditions using the Intel PAT tool further. The next section explains the evaluation of these 2 results.

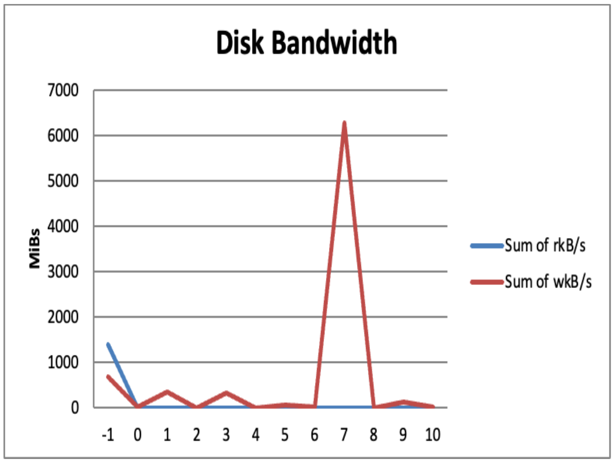
8. Evaluation

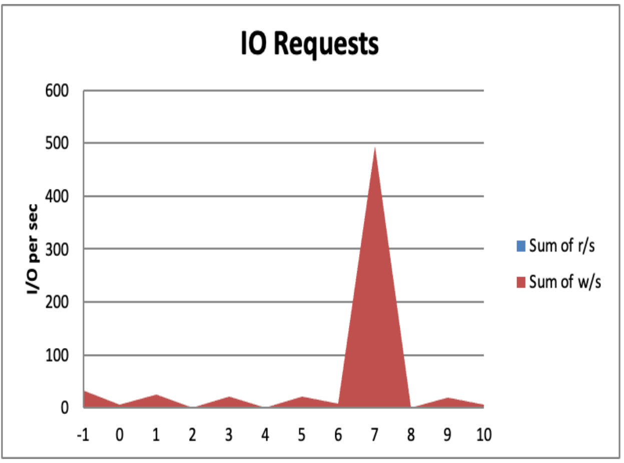
1) PAT Analysis of local volume

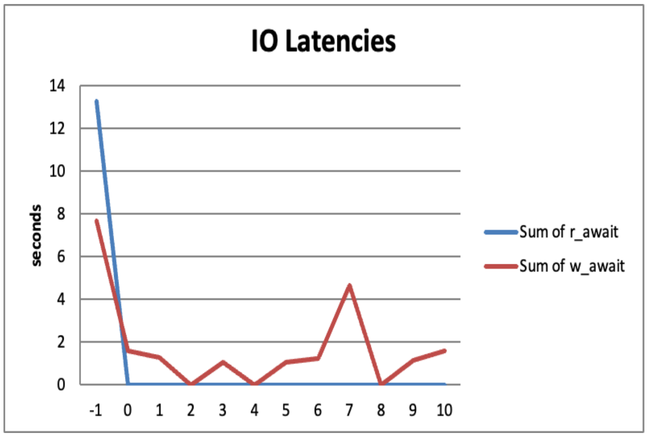
The test involved one node with four tasks per node, and three repetitions were performed. Each I/O transfer size was 65536 bytes, and the block size was also 65536 bytes. The aggregate file size was 262144 bytes.

The results of the test showed a write bandwidth of 1129.88 MiB/s, a read bandwidth of 13056.59 MiB/s, a maximum IOPS of 223696.21, and a minimum IOPS of 190650.18. The maximum and minimum transfer sizes were 13981.01 MiB and 11915.64 MiB, respectively. The latency was 0.00022 s for writes and 0.00002 s for reads. The test took a total of 0.00022 s to complete.

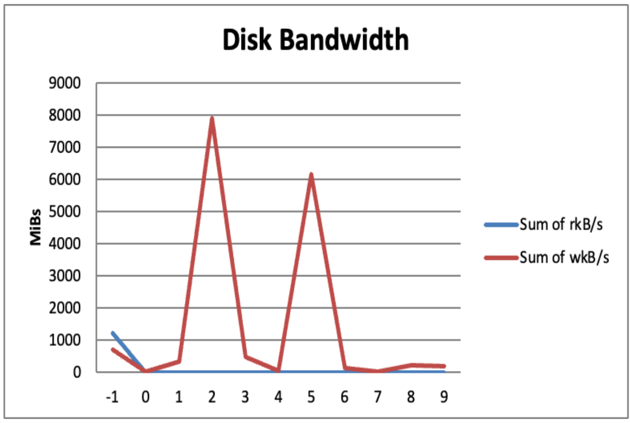
Overall, the test showed good I/O performance, with high bandwidth and IOPS and low latency.

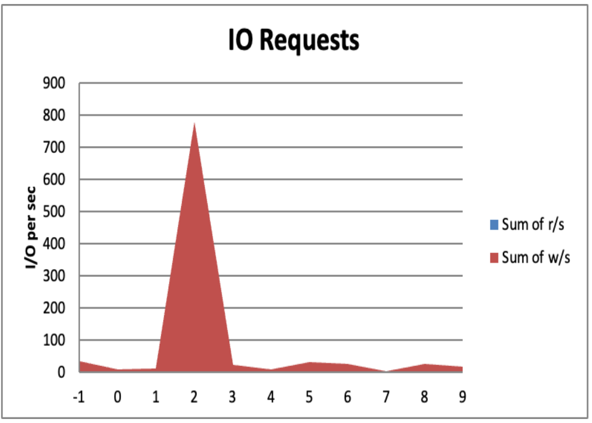


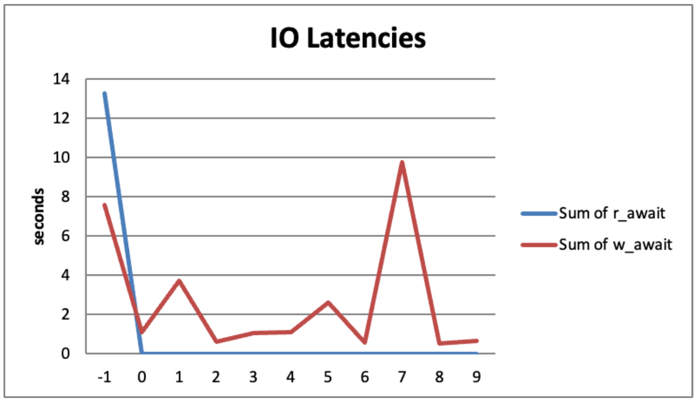




2) PAT Analysis of kubernetes CSI volume







The test involved writing and reading data of size 64KB with a block size of 64KB and repeating the test 3 times.

The results show that the maximum bandwidth achieved for writes was 1072.16 MiB/s, with a minimum of 879.68 MiB/s and a mean of 993.62 MiB/s. For reads, the maximum bandwidth achieved was 7919.96 MiB/s, with a minimum of 1041.43 MiB/s and a mean of 7506.45 MiB/s. The test took an average of 0.00025 seconds for writes and 0.00003 seconds for reads.

Overall, the test showed that the system can achieve high bandwidth for parallel I/O operations using the selected configuration.

Final Comments:

The results from the two tests of the 2 systems indicate different performance metrics.

Comparing the two tests, the CSI volume test showed better write performance, with a higher write bandwidth compared to the local volume. However, the read performance was better for local volume, with a higher read bandwidth compared to for the CSI volume. The read performance is better for the local volume it could be because it does not have the additional overhead of communication between the Kubernetes API server and the CSI drive the write performance was better in the CSI volume compared to the local volume. One possible reason for this could be that the CSI driver used in the CSI volume has better optimizations for handling write requests than the local volume implementation. It is also possible that the underlying storage system used by the CSI volume provides better performance for write operations. However more detailed study can be done to comment on this.

9. Division of work:

Prajakta	Amith
Kubernetes platform study	Kubernetes installation for minikube
Creating pv, pvc and pod	Shell script for installing minikube
Studying and Installing beegfs and its CSI driver	Ior study
Creating google cloud bucket, google service account	Ior installation
Deploying CSI drivers for google cloud	Ior testing on kubernetes
Shell script to automate pod, pvc and pv creation	Intel PAT configuration and testing

10. Conclusion

- **Things we learnt in this project:**

  1. Basic understanding of the kubernetes environment. Installing minikube, setting up Pods, Persistent Volume Claim(PVC) and Persistent Volume(PV) and dynamic volume using CSI.
  2. IOR concepts - how to install ior, run ior, parameters of ior testing
  3. Running ior inside kubernetes. Using docker images to run on minikube.
  4. Installation and basic understanding of Beegfs.
  5. Installation of Google cloud storage CSI
  6. Writing yamls and shell scripts

- **Challenges faced while implementing:**

1) Replacing Beegfs with Google cloud : We first proposed to study the kubernetes CSI by integrating dynamic Beegfs volume. As we deployed beegfs and its CSI driver(developed by NetApp), the driver failed to deploy. The necessity for beegfs driver to work is it needs to get the beegfs-client configuration. When we deployed the CSI driver, it was not able to pick the installed beegfs-client, due to which the mounting was failing. We have raised a case with the NetApp team to look into the issue. So in order to complete the goal of our project we integrated Google cloud storage as a dynamic backend to our kubernetes CSI.

2) Carrying out ior tests on minikube cluster: As there is very little research on ior and kubernetes testing, we faced some challenges in getting the docker images for ior and mpi to make ior run on kubernetes. We had to build a docker image to run ior and then use the same to deploy an ior testing pod.
3) Configuration of Beegfs: Beegfs configuration was a bit challenging because the installation documentation on their official website is not that apt. We had to search for various solutions while installing beegfs.

References

[1] Rodrigo Leite, Priscila Solis and Eduardo Alchieri, “Performance Analysis of an Hyperconverged Infrastructure using Docker Containers and GlusterFS”, *9th International Conference on Cloud Computing and Services Science*

[2] Naweiluo Zhou, Yiannis Georgiou, Marcin Pospieszny, Li Zhong, Huan Zhou, Christoph Niethammer, Branislav Pejak, Oskar Marko and Dennis Hoppe, “Container orchestration on HPC systems through Kubernetes”, *Journal of Cloud Computing: Advances, Systems and Applications, 2021*

[3] Angel Beltrel, Pankaj Sahal, Madhusudhan Govindaraju, Andrew J. Younge, and Ryan Eric Grant, “Enabling HPC workloads on Cloud Infrastructure using Kubernetes Container Orchestration Mechanisms”

[4] Izzet Yildirim, Meng Tang, Anthony Kougkas, Xian-He Sun, “Performance Analysis of Containerized OrangeFS in HPC Environment”