Distributed file systems performance tests on Kubernetes/Docker clusters

Federico Fornari[1], Alessandro Cavalli[1], Daniele Cesini[1], Antonio Falabella[1], Enrico Fattibene[1], Lucia Morganti[1], Andrea Prosperini[1] and Vladimir Sapunenko[1]

[1] CNAF - Italian Institute for Nuclear Physics, Bologna, Italy

E-mail: federico.fornari@cnaf.infn.it

Abstract. Modern data centers need distributed file systems to provide user applications with access to data stored on a large number of nodes. The ability to mount a distributed file system and leverage its native application programming interfaces in a Docker container, combined with the advanced orchestration features provided by Kubernetes, can improve flexibility in installing, monitoring and recovering data management and transfer services. At INFN-CNAF some distributed file systems (i.e. IBM Spectrum Scale, CephFS and Lustre-ZFS) deployment tests with Kubernetes and Docker have been conducted recently with positive results. The purpose of this paper is to show the throughput scores of the previously mentioned file systems when their servers are containerized and run on bare metal machines using a container orchestration framework. This is a preliminary study: for the time being, only sequential read/write tests have been considered.

1. Introduction

INFN-CNAF is one of the Worldwide LHC Computing Grid (WLCG) Tier-1 data center, providing computing, networking and storage resources to a wide variety of scientific collaborations, ranging from particle physics to bioinformatics to industrial engineering. Containerization technologies leverage isolation and resource limitation to improve physical CPU and memory utilization by running services, while container orchestration tools can ease service management in a distributed environment providing features like automation and failover. Therefore, the aim of this work is to analyze the performances of file systems managed and exported by containerized clusters deployed with Docker [1] and Kubernetes [2]. A suitable testbed, providing computing and networking resources through bare metal machines, has been exploited to verify cluster performances and stability, allowing to compare the behavior of different distributed file systems. For a comparison between throughput results obtained with containerized and non-containerized file systems deployed on virtual machines, see [3].

2. Methodology

The testbed cluster consists of 8 16-core Intel Xeon nodes (CPU E5-2609 v4 @ 1.70 GHz) with a total amount of 320 CPUs and 1.5 TB RAM; 4 nodes have been designated as clients and 4 have been configured as servers. Network communication between nodes is ensured by two 10 Gbit/s interfaces coupled with bonding. Each of the 8 nodes is provided with 30 8-TB rotating disks

belonging to JBODs connected via Serial Attached SCSI interface. The cluster provides access to production storage exposed with Ceph, comprising a total number of 240 disks, corresponding to an overall raw storage space of about 2 PB. The OS installed on the nodes is CentOS 7.7.

The same kind of deployment has been replicated using three different distributed file systems: IBM Spectrum Scale [4] (formerly GPFS), CephFS (the Ceph [5] POSIX-compliant file system) and Lustre [6] with ZFS backend. Three disks per node have been used for the Kubernetes-managed distributed file systems to be tested: on each of the 4 servers, one disk has been assigned to GPFS, one to Ceph and one to Lustre. The distributed file systems have been configured in separated Kubernetes namespaces and the tests have been executed on one file system at a time. All the containers have been deployed using Docker 20.10.1 and Kubernetes 1.20.1.

Concerning GPFS (IBM Spectrum Scale 5.0.5-2) cluster deployment, Figure 1 illustrates the setup. It comprises 4 containerized servers deployed on nodes 5 to 8, and 4 containerized clients on nodes 1 to 4, mounting the file system to perform read/write tests. Each GPFS server has a 8-TB disk attached and configured as network shared disk (NSD) and a replica 2 file system has been created over the NSDs.

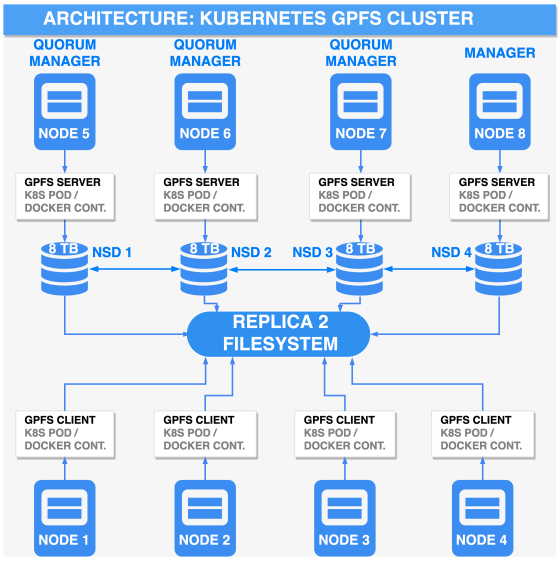

Figure 1. GPFS cluster setup.

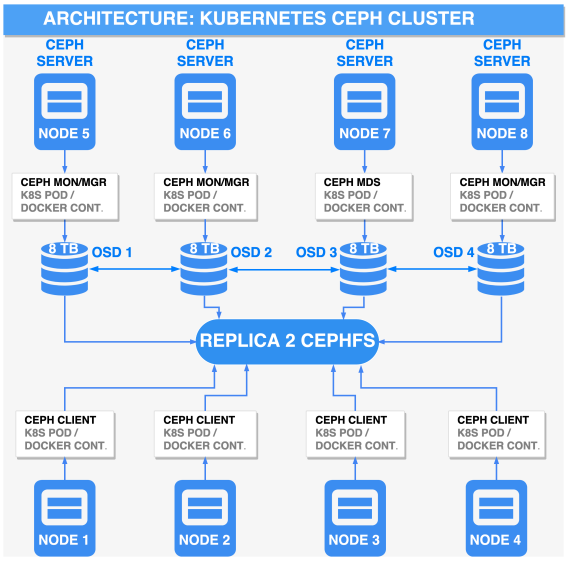

Figure 2. Ceph cluster setup.

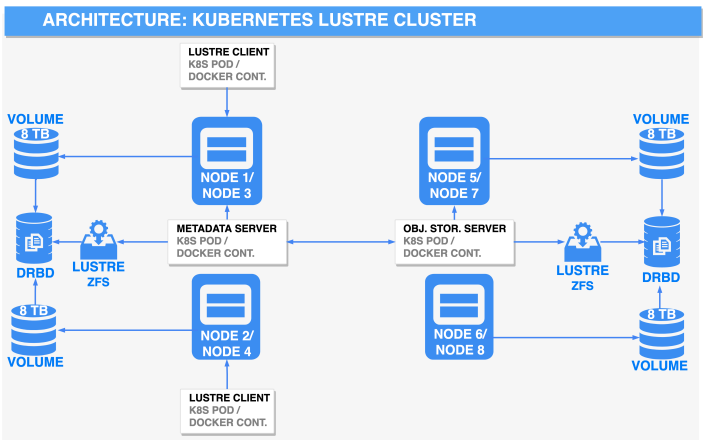

Figure 3. Lustre cluster setup.

In Figure 2 a schematic view of the Ceph Nautilus (14.2.10) cluster with containerized servers is reported. The cluster has been configured and started using customized Kubernetes Helm Charts [7]. The 4 servers have been distributed in the following way: 3 cluster nodes (nodes 5, 6 and 8) host Ceph monitor and manager services, while the CephFS metadata server has been instantiated on node 7. Each server has a Ceph Object Storage Device (OSD) configured on a 8-TB disk. A CephFS replica 2 file system has been created over the OSDs, and 4 client Pods, mounting the file system to perform read/write tests, have been started on nodes 1 to 4.

The Lustre Kubernetes cluster has been setup implementing replica 2 data redundancy using Distributed Replicated Block Device [8] (DRBD) and deploying 4 servers: 2 Object Storage Servers (OSSs) and 2 Metadata Servers (MDSs). Figure 3 shows that 2 OSS Pods have been configured on nodes 5 and 7 to use 8-TB disks replicated respectively on nodes 6 and 8 via DRBD, and the same procedure has been adopted to setup 2 MDS Pods on nodes 1 and 3, using two 8-TB DRBD-coupled disks. 4 client Pods have been started on each of the 4 cluster nodes configured for the 2 MDSs. The Lustre version selected for the setup is 2.12.4, following compatibility matrix with CentOS 7.7 kernel version.

The tests have been carried out with `iozone` [9], alternating sequential write and read operations on binary files with increasing size from 4 to 32 GB, involving a number of parallel threads equally increasing from 4 to 32, in order to write and read 1 TB data at maximum (32 parallel threads reading/writing 32 GB files). `iozone` has been configured to distribute the execution of the processes over the 4 client Pods via *ssh*.

3. Results

Table 1 illustrates typical read and write throughput for a single GPFS NSD server, showing an average throughput during reading phases which is around 150 MB/s for each of the servers. During writing phases, a single NSD server throughput averages on about 180 MB/s. `iozone` surface plots in Figure 5.a reveal that on client side the total average read throughput is about 650 MB/s, while total write throughput reaches about 360 MB/s.

Average read and write throughput for a single server during the 32 GB files phase of the test are presented in Figure 4. Average write throughput seen over the entire test for each Ceph OSD is around 1 Gbit/s, as shown in Table 1. `iozone` surface plots in Figure 5.b testify huge throughput values on client side, especially for read operations.

Table 1 shows that the average write throughput for a typical Lustre Object Storage Server is 140 MB/s, while average read throughput is around 160 MB/s. `iozone` surface plots for Lustre (Figure 5.c) are pretty similar to Ceph ones, with very high throughput on client side, especially for read operations.

Distributed File System	Average Read (MB/s)	Average Write (MB/s)
GPFS	153	178
Ceph	85	122
Lustre	158	140

Table 1. Average throughput for each distributed file system, measured for one of the 4 servers, over the entire test.

4. Discussion

The aggregated average write throughput of the 4 GPFS NSD servers is around 700 MB/s, roughly corresponding to 6 Gbit/s. Such throughput value is in accordance with expected performance for a single rotating disk connected via 6Gb SAS interface: the sustained transfer

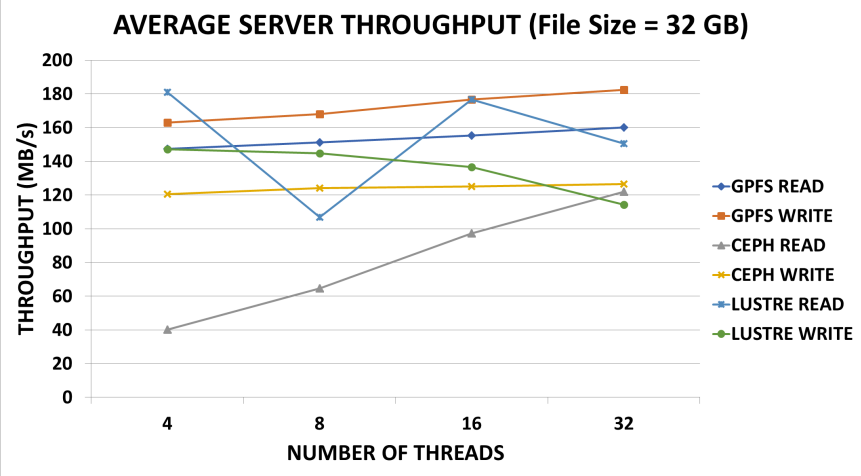

Figure 4. Distributed file systems server throughput, measured for one of the 4 servers, with 32 GB files.

rate specified by the vendor is generally around 200-250 MB/s per rotating disk. No evident cache effects are detectable on GPFS clients, as visible in Figure 5.a. GPFS is provided with a client-side cache design: the cache is kept in a dedicated and pinned area of each application node's memory called the *pagepool*, which is 1 GB by default.

The data reported in Figure 4 for Ceph OSD throughput does not include read throughput outliers due to strong cache effects. However, considering the case of 32 threads, where total amount of read/written data reaches 1 TB (which is 4 times larger than a single server RAM), a read throughput around 1 Gbit/s shows up, and this occurs for all the OSD servers. This is expected behavior by `iozone`, since it caches blocks or files, and does not wait for the storage system to acknowledge that a write is fully committed before moving on. If the size of read/written data gets sensibly higher than the server's RAM size, cache effects are ruled out. As a matter of fact, Figure 5.b shows that the average Ceph client throughput in the region of big files ($\geq$ 16 GB) vs. many threads ($\geq$ 16) gets back to a range of values similar to GPFS ones, standing between 300 and 400 MB/s.

Lustre shows the effects of caches with small files similarly to Ceph. On client side, big files ($\geq$ 16 GB) vs. many threads ($\geq$ 16) region in Figure 5.c shows average throughput values going down towards the expected range of values, around 300-400 MB/s.

5. Conclusions

Some conclusions can be drawn about containerized distributed file system clusters.

The 3 file systems have manifested good stability during the execution of our tests. This has been a preliminary study: only sequential read/write operations have been run. The presented data allows to determine that GPFS guarantees the highest read/write throughput on server side and the minimum cache effects on client side. Although dominated by cache contributions during small files read/write operations, Ceph and Lustre show to resemble GPFS performances when total amount of read/written data gets significantly higher than a server's RAM. Focusing on single servers, Lustre shows higher average throughput than Ceph. In our tests, Lustre servers have achieved half the overall throughput with respect to Ceph and GPFS, since Lustre data replication has been implemented with DRBD.

The main result of our work has been to successfully verify that Kubernetes can be used to efficiently manage distributed file systems installation and operability, with good level of performances regardless of the specific file system type.

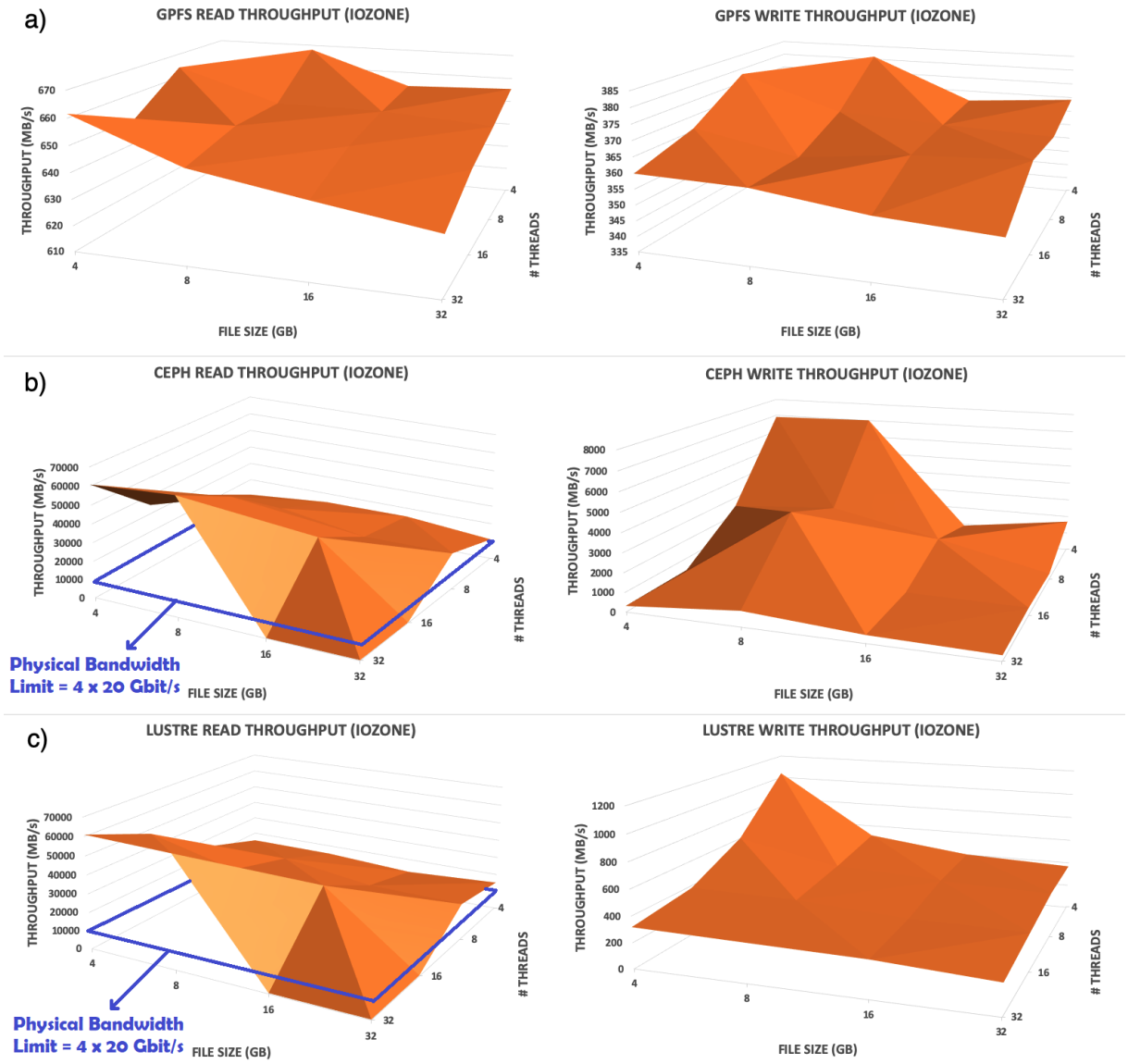

Figure 5. Read and write throughput as measured with `iozone` benchmark for GPFS (a), Ceph (b) and Lustre (c) file systems.

References

[1] `https://www.docker.com` (Last seen: September 2022)
[2] `https://kubernetes.io` (Last seen: September 2022)
[3] `https://pos.sissa.it/378/020/pdf` (Last seen: September 2022)
[4] `https://www.ibm.com/products/spectrum-scale` (Last seen: September 2022)
[5] `https://ceph.io` (Last seen: September 2022)
[6] `https://www.lustre.org` (Last seen: September 2022)
[7] `https://github.com/ffornari90/ceph-helm` (Last seen: September 2022)
[8] `https://www.drbd.org` (Last seen: September 2022)
[9] `https://www.iozone.org` (Last seen: September 2022)