Performance Analysis of an Hyperconverged Infrastructure using Docker Containers and GlusterFS

Rodrigo Leite, Priscila Solis and Eduardo Alchieri
Applied Computing Masters Program, Department of Computer Science, University of Brasilia (UnB), Brasília, Brazil

Keywords: Hyperconverged Infrastructures, GlusterFS, Containers, DFS, Cassandra, MongoDB, Docker, Cloud, Storage.

Abstract: The adoption of hyperconverged infrastructures is a trend in datacenters, because it merges different type of computing and storage resources. Hyperconverged infrastructures use distributed file systems (DFS) to store and replicate data between multiple servers while using computing resources of the same servers to host virtual machines or containers. In this work, the distributed file system GlusterFS and the hypervisor VMware ESXi are used to build an hyperconverged system to host Docker containers, with the goal of evaluate the storage performance of this system compared to traditional approach where data is stored directly on the server's disks. The performance of the container's persistent storage is evaluated using the benchmark tool Yahoo Cloud Service Benchmark (YCSB) against the NoSQL databases Cassandra and MongoDB under differents workloads. The NoSQL database's performance was compared between the hyperconverged system with multiples disk configurations and a traditional system with local storage.

1 INTRODUCTION

In the last years, the amount of data generated by different devices and users is growing constantly and exponentially. This brings a challengue to the traditional storage solutions that need to cope with massive data storage and processing. For this, in the last years several large-scale Distributed File Systems (DFS) were developed to overcome these limitations. Some benefits of DFS are scalability, parallelism, the ability to run on commodity hardware and fault-tolerance. These systems assemble many nodes across a network and provide a distributed file system with large storage capacity, where data can be transparently accessed (Roch et al., 2018). The ability to use commodity hardware makes scalability economically viable, allowing the addition of more devices to scale up the system in an incremental fashion. Because the file system runs on several commodity hardware components, which are highly prone to failure, techniques such as replication and erasure codes are used to avoid data loss and increase fault-tolerance in case of hardware failures.

Traditionally, computing and storage resources in datacenters are separated into different pods. The computing pod usually uses virtualization technologies to optimize resources, while the storage pod is usually based on monolithic hardware. Recently, hyperconvergence is an emerging paradigm that has been gaining popularity, both in academia and industry (Verma et al., 2017). This paradigm allows to merge computing and storage components with hypervisors and DFS solutions. While traditional storage systems use dedicated hardware to storage and compute, a hyperconverged system uses a DFS and a hypervisor to aggregate the storage capacity and computing resources, providing an unique pod of compute and storage.

In a cloud computing environment, containers can benefit from a hyperconverged infrastructure. The traditional container deployment uses local storage on the node to start and run. Then, the ability to store container's persistent data on a hyperconverged system may improve scalability, elasticity and fault-tolerance in these environments.

Considering the above, the motivation of this work is to evaluate storage performance of a proposed hyperconverged system, based on GlusterFS as the DFS and VMware ESXi as the hypervisor. The goal is to evaluate the performance and potential of container's storage I/O on an hyperconverged system, by running inside Docker containers two applications highly dependent on storage capabilities, the NoSQL databases Cassandra and MongoDB (Abramova and Bernardino, 2013), under several workloads of the Yahoo Cloud Service Benchmark (YCSB) (Cooper

Leite, R., Solis, P. and Alchieri, E.
Performance Analysis of an Hyperconverged Infrastructure using Docker Containers and GlusterFS.
DOI: 10.5220/0007718003390346
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 339-346
ISBN: 978-989-758-365-0

et al., 2010). The results of the proposed hyperconverged system are compared with a traditional configuration using local storage, with the intention of verifying the viability of its use in place of direct-attached storage.

This paper is organized as follows: Section 2 presents related works and the literature review; Section 3 presents and describes the proposed architecture; Section 4 details the experiments and results and finally, Section 5 presents the conclusions and future work of this research.

2 THEORETICAL CONCEPTS AND RELATED WORK

Permanent storage consists of a named set of objects that are explicit created, are immune to temporary failures of the system, and persist until explicitly destroyed. A file system is a refinement that describes a naming structure, a set of operations on objects described by explicit characteristics. DFS allow multiple users who are physically dispersed in a network of autonomous computers share in the use of a common file system. Cloud storage is a system that provides data storage and assembles a large number of different types of storage devices through the application software which are based on the functions of cluster applications, grid techniques and DFS. Cloud storage is a cloud computing system with large capacity storage. An hyperconverged environment combines compute, storage and networking components into a single system or node based on commodity servers, with the aim to reduce hardware costs when compared to specialized storage arrays, servers and other equipment, which appears today as an interesting option to cloud providers.

In the DFS area, recent studies evaluated the common problem of how to store and handle large amounts of data. For example, in (Roch et al., 2018) there is a comparison between DFS for storing molecular calculations of quantum chemistry. In this study the DFS GlusterFS and Ceph were tested, using 8 nodes distributed in 8 different combinations. The experiments were performed with 7 different patterns of reading and writing. The conclusion was that GlusterFS outperforms Ceph during large I/O workloads and that GlusterFS administration is simplified, requiring less maintenance and installation time. The conclusions about Ceph were that it has greater complexity and needs separate servers for metadata (data about other data), which increases the complexity of the solution.

Another DFS study in a similar area investigates the problem of how to store and transfer large amounts of genetic data between two institutions (Mills et al., 2018). This study shows that the best way to transfer data between two research institutions is to use a combination of a high-speed communications network and a high-performance file system. Finding a balance between these two items enables data to be read from the source at high speed, transmitted quickly, and then recorded at high speed at the destination. The DFS analyzed in this study were BeeGFS, Ceph, GlusterFS and OrangeFS. Experiments were performed with 4 to 8 servers at the source transmitting data via FTP to another 4 to 8 servers at the destination, with BeeFS getting the best performance.

In the work of (Kaneko et al., 2016) a guideline for data allocation in DFS is proposed, recommending that data stored in multiple nodes with few disks have a better performance than a few nodes with many disks. In the experiments, volumes built with disks on different servers were compared with volumes built with several disks on the same server. The comparison was made analyzing the read and write throughput while copying files to and from the volumes. In the results it was verified that the proposed guideline obtained better performance in larger multiplicities. The multiplicity is defined by the author as being the number of clients accessing simultaneously a number of volumes. For example, multiplicity 24 means 4 clients simultaneously accessing 6 volumes. The analysis concluded that distributing the volume across several servers results in improved data throughput because of the parallelism in data access. This guideline can be considered a good practice for building DFS. Building volumes with few server the advantages of a parallel file system are lost and server resources (processor, disk controller, network card, etc.) can impact the performance. Spreading a volume across multiple servers allowed them to be accessed by different servers, thinning resource consumption between each server and improving data throughput.

Studies on performance of hyperconvergence environments are still scarce. The fundamental concept behind this architecture is based on the Beowulf Systems, that already been vastly studied by academia. In a Beowulf system a cluster of commodity-grade computers are networked to run some software that can perform parallel computing and storage. A hyperconverged system is a recent evolution of this concept, mashing virtualized servers and virtualized storage onto the same clusters with the goal of simplifying the infrastructure.

In one of the first studies specifically about hyperconvergence, the General Parallel File System

(GPFS) is presented as a software defined storage solution for hyperconverged systems (Azagury et al., 2014). In this work, several concepts of hyperconvergence and software defined storage are outlined, and the proposed solution is to use GPFS within a hyperconverged architecture using virtual storage appliances (VSA) that manage physical storage resources. The VSA allow the increase of capacity in a scale-out model and data protection through the "GPFS native RAID" (GNR) that makes copies of the data in several nodes.

Recent studies on containers storage performance analyze the possible configurations for storage access (Tarasov et al., 2017) (Xu et al., 2017a), how to provide encryption in image manipulation (Giannakopoulos et al., 2017) and performance using solid state disks (SSD) (Xu et al., 2017b) (Bhimani et al., 2016). In (Tarasov et al., 2017) and (Xu et al., 2017a), the mechanisms for containers access storage areas are described and analyzed, but both are limited to local storage, as well as in (Giannakopoulos et al., 2017) (Xu et al., 2017a). In (Xu et al., 2017b) the authors analyze the use of remote storage for containers, using non-volatile memory express over fabrics (NVMF) technology being accessed by a remote direct access memory (RDMA) over a 40 Gbps ethernet network. This study is closer to our proposal, since the container's data will be stored in a DFS.

The amount of data generated by IoT devices is addressed in another study, in witch the storage paradigms for store and process data from IoT devices are analyzed. The conclusion points to the hyperconverged architecture as one solution because of its scalability, fault-tolerance and use of COTS hardware (Verma et al., 2017).

3 PROPOSED ARCHITECTURE

The proposed architecture is presented in Figure 1 and works as follows:

- The server runs a Type I Hypervisor with at least two disk controllers: one for hypervisor, VSA and container host ❶; and one for the hyperconverged system ❷.
- On the disks connected to the disk controller managed by the hypervisor on path ❸ are stored the hypervisor itself and both VSA and the Container Host virtual machines, as can be seen on path ❺.
- Using the "PCI Passthrough" feature on the hypervisor, the second disk controller ❷ is directly managed by the VSA virtual machine via path ❹.

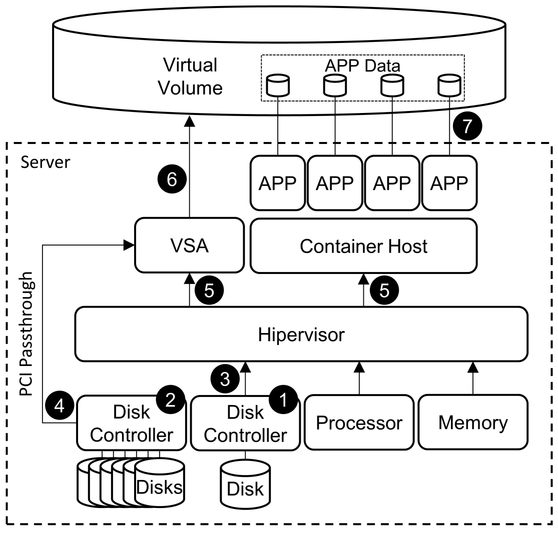

Figure 1: Hyperconverged Server Architecture.

- Each VSA, with its own storage resources, is configured to participate in an existing cluster or start a new one. The nodes provide their individual resources ❻ to create a distributed virtual volume across the nodes of the cluster, as presented on Figure 2. Data replication between VSA nodes is performed over a low-latency, high-bandwidth Ethernet network.
- After the virtual volume is created, it can be accessed by the container engine as a scalable and fault-tolerant data storage unit, as can be seen on path ❼.

The use of a disk controller dedicated to the VSA is a desirable characteristic because it allows direct control of the disks, dropping an abstraction layer in the hypervisor. Another desirable requirement is a high bandwidth and low latency network between the servers to avoid congestion or any kind of degradation during storage operations. The use of disks with same model to build a volume is another important aspect to get a better performance (Kaneko et al., 2016).

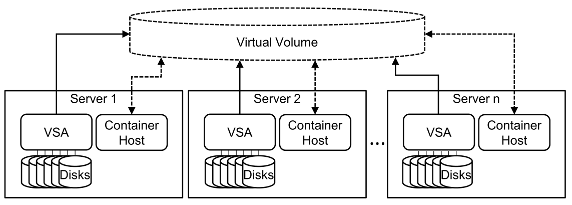

Figure 2: Virtual Volume build by the VSAs.

3.1 Implementation

In the next subsections, we describe the tools that were integrated to implement the proposed arquitecture.

3.1.1 GlusterFS

GlusterFS is an open-source DFS that allows to distribute a large set of data across multiple servers. It has simple operation, being supported by most Linux distributions. It does not separate metadata/data and therefore does not need additional servers for this task (Fattahi and Azmi, 2017). It has a native mechanism of redundancy based on the replication of files between a set of "bricks" (a basic unit of storage) that form a resilient logical unit (Roch et al., 2018). It allows the use of heterogeneous nodes, enabling nodes with different configurations on the same distributed volume. In these cases, it is recommended to group nodes with the same characteristics for better performance (Kaneko et al., 2016).

Bricks are the basic storage unit in GlusterFS, represented by an export directory on a server. A collection of blocks forms a "Volume" in which GlusterFS operations take place. GlusterFS allows you to group Bricks to form five different types of volumes:

- Distributed Volume: Distribute files between bricks. The final volume capacity is the sum of bricks' capacity. It has no redundancy. Failure on a brick corrupts the volume. Recommended for cases where availability is not important or is provided by another mechanism.
- Replicated Volume: The files are replicated between the volume bricks. Recommended for cases where high availability and reliability are required.
- Distributed Replicated Volume: Distributes files between replicated subvolumes. Recommended for cases where scalability, high availability and good reading performance are required.
- Dispersed Volume: They are based on erasure codes and provide efficient protection against disk or server failures. A coded fragment of the original file is stored in each brick, so that from a set of fragments it is possible to retrieve the original file.
- Distributed Dispersed Volume: Distributes files between dispersed subvolumes. It has the advantages of a distributed volume, but using dispersed storage within the subvolume.

3.1.2 ESXi

VMware ESXi is a hypervisor that runs directly on the host hardware, i.e., it is a virtual machine monitor (VMM) type I (bare metal). It takes minimal memory from the physical server and provides a robust, high-performance virtualization layer that abstracts the server's hardware resources and enables the sharing of these resources across multiple virtual servers.

3.1.3 Docker

Docker is a application that performs operating-system-level virtualization also known as containerization. It uses the resource isolation features of kernel to allow independent containers to run within a single host, avoiding the overhead of starting and maintaining virtual machines (VMs) (Xu et al., 2017a). Containers and virtual machines have similar resource isolation and allocation benefits but different architectural approaches, which allows containers to be more portable and efficient compared to bare metal and virtual machines (Bhimani et al., 2016).

3.1.4 Virtual Storage Appliance

The core component of our proposed architecture is the Virtual Storage Appliance (VSA), a virtual machine that runs a minimal version of the Linux operating system and the GlusterFS software. GlusterFS is configured to manage the physical storage resources presented to the VSA and use these to build a DFS. The choice of GlusterFS for the DFS is due to the fact that data and metadata are stored together in the nodes, without a specific node to handle metadata (Fattahi and Azmi, 2017). This feature makes GlusterFS ideal for use in a hyperconverged architecture since the nodes are independent and can run without external dependencies, simplifyng the architecture and allowing the deployment of just one VSA per hypervisor and nothing else. The VSA nodes, each with its own storage resources, allow the creation of DFS volumes that span across many server and can be used by hypervisor and container engine to store data with scalability and fault-tolerance.

4 EXPERIMENTAL ANALYSIS

4.1 Testbed

In our experiments we built a hyperconverged system using three IBM HS23 servers with dual Intel Xeon CPU E5-2670 2.60GHz CPUs, 96GB of RAM, one 16GB SSD (to host ESXi, VSA and Docker Host), two 1TB HDD (to the hyperconverged system), hypervisor ESXi version 6.5 and network connection via two gigabit Ethernet ports. The VSA on each hypervisor has 4 vCPU, 8 GB of RAM, runs CentOS Linux release 7.5.1804, GlusterFS version 3.12.15 and XFS as the file system for disks managed by Gluster. The Container Host has 12 vCPUs, 80 GB of RAM, runs CentOS Linux release 7.5.1804 and Docker 18.09. The connection between the Docker Host and the virtual volume built by the VSAs is made through Gluster Native Client. To evaluate the storage I/O performance of the hyperconverged system we choose the NoSQL databases Cassandra (version 3.11.3) and MongoDB (version 4.0.3) running in Docker containers and storing persistent data on the hyperconverged virtual volume.

To compare results from the hyperconverged system we built a traditional system where storage resources are provided by local disks on the server. For this system we used another IBM HS23 server with same configuration as the hyperconverged servers, but the persistent storage for the containized NoSQL databases were provided by the hypervisor using the local server's disk.

Since the objective is to analyze storage performance, the memory cache feature on both NoSQL databases were disabled, meaning that all database operations had to hit the storage volume and consequently the underlying disks.

4.2 Workload

To evaluate our proposal we used the Yahoo Cloud Serving Benchmark (YCSB) framework (Cooper et al., 2010), a tool built do load, run and measure performance of different NoSQL databases. After loading data on the desired database, there are six core workloads that can be run to evaluate database's performance. These core workloads are named A, B, C, D, E and F, each with different I/O patterns. Besides those, YCSB also allows using custom workloads, which we used to test three more workloads: two from Microsoft's datacenters (Huang et al., 2017) and one developed by us, called "Heavy Update", with a heavy write load. The 9 different workloads used to evaluate our propose is summarized in Table 1.

Table 1: Application workloads used for evaluation.

Workload	I/O Pattern
YCSB-A	50% read, 50% update
YCSB-B	95% read, 5% update
YCSB-C	100% read
YCSB-D	95% read, 5% insert
YCSB-E	95% scan, 5% insert
YCSB-F	50% read, 50% read-modify-write
Microsoft Cloud Storage	26.2% read, 73.8% update
Microsoft Web Search	83% read, 17% update
Heavy Update	100% update

4.3 Metrics, Validation and Factors

The metric evaluated during the execution of the experiments is the operations per second (ops/sec), i.e., how many read, update or insert operations can be submitted to the NoSQL database every second.

The validation of the results is performed evaluating the results from the proposed hyperconverged system in comparison with the results from a traditional system where data is stored on the local server's disks. For our experiments, the factors we chose to analyze were: type of GlusterFS volumes; type of NoSQL database; and number of threads.

4.4 Evaluation Scenarios

In the hyperconverged system built for this study, GlusterFS can be configured to create five types of volumes: Distributed, Replicated, Distributed Replicated, Dispersed and Distributed Dispersed. Since the Distributed volume does not offer fault-tolerance and the Replicated volume does not scale, we did not used these two types of disk configurations. The other three types of volume offer equal fault-tolerance protection, but different types of performance. So we tested all those three types of volumes with all workloads.

To evaluate the hyperconverged storage I/O performance we used the NoSQL databases Cassandra and MongoDB, each one submitted to the same workloads and storing persistent data on the same types of volumes. The choice for these two is due to the fact that databases are applications that depend on the storage performance where data is stored and retrieved, with Cassantra and MongoDB being popular among recent studies in both academia and industry.

Both NoSQL databases were loaded with 1.000.000 records of 1KB using the YSCB tool. Then the databases were submitted to the workloads with different threads numbers (ranging from 1 to 100, in steps of 10). In each thread test, 100.000 transactions were made. The mean resulted of the amount of transactions made in each second were calculated with a 95% confidence interval.

4.5 Experiments

- **Experiment 1 - YCSB Core Workloads.** This experiment was conducted using the six core workloads available in the YCSB framework against Cassandra e MongoDB containerized instances. The results can be seen on Figure 3.
- **Experiment 2 - Microsoft Workloads.** This experiment was conducted using two Microsoft's

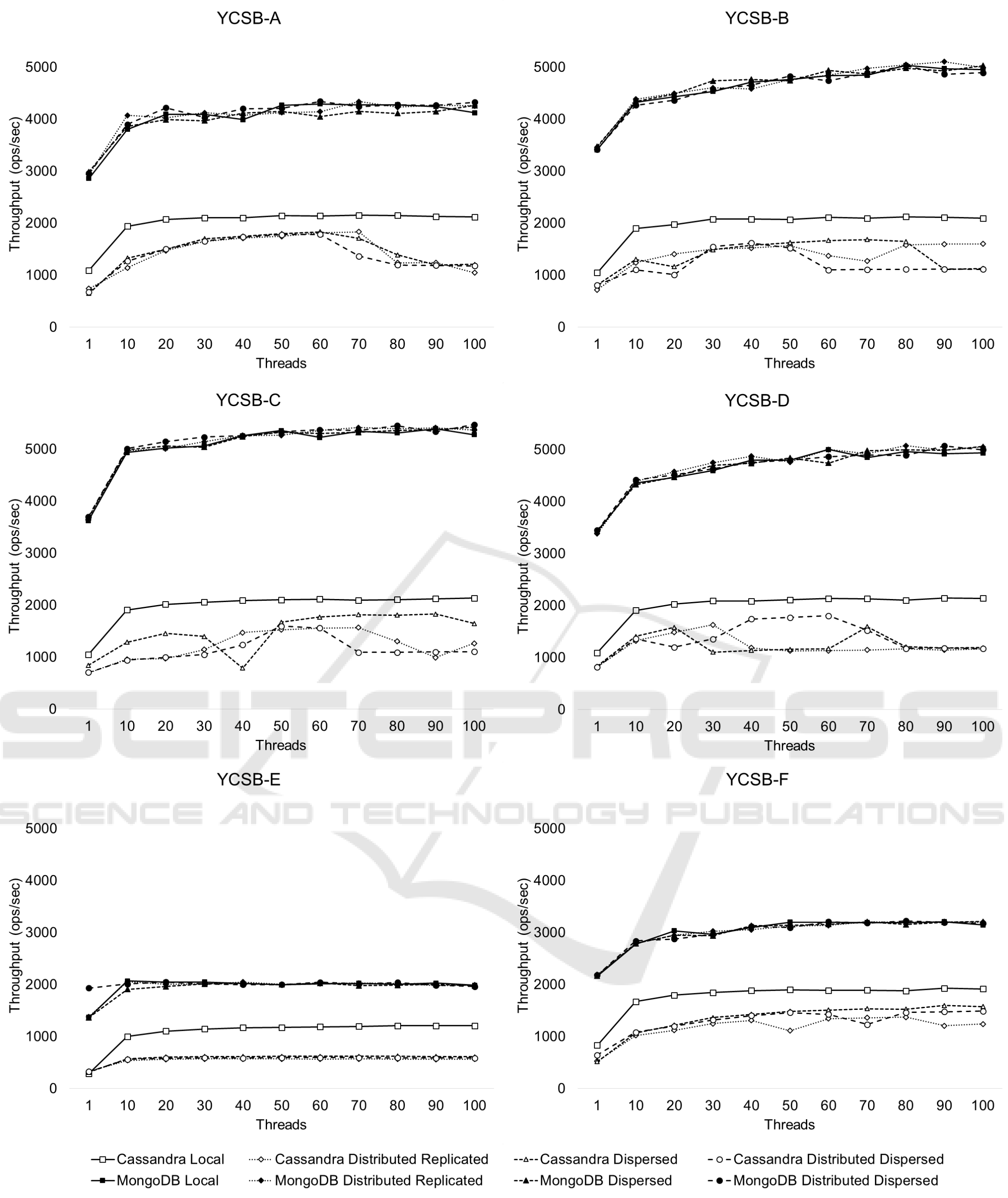

Figure 3: YCSB Workloads Results.

datacenters workloads. The results can be seen on Figure 4.

- **Experiment 3 - Heavy Update Workload.** To analyze the performance of write I/O storage operation, we this experiment with only updates on the database. The results can be seen on Figure 5.

4.6 Analysis of Results

The experimental results shown on Figures 3, 4 and 5 indicates that MongoDB performed better than Cassandra in all experiments. MongoDB running on the hyperconverged system presented a performance equivalent to that of a traditional system, independent

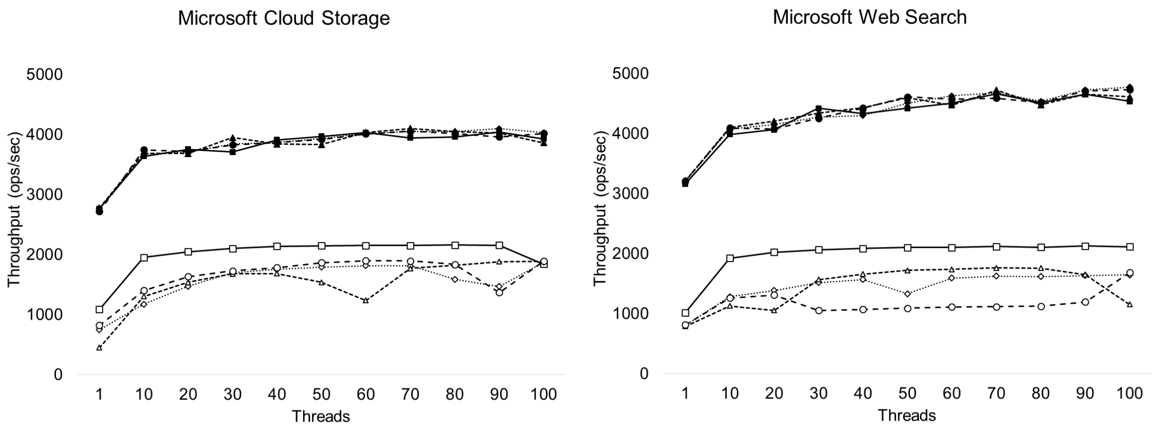

Figure 4: Microsoft Workloads Results.

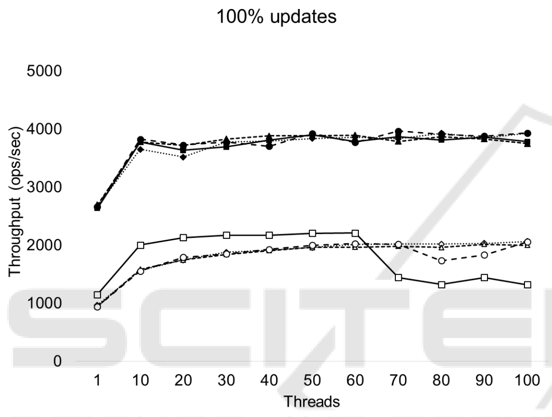

Figure 5: Heavy Update Workloads Results.

of the number of threads. In some specific cases we archived a 7% better performance on the hyperconverged system than on local storage. We consider this a good result because, even with the abstractions in storage layer imposed by the hyperconverged system, the performance was similar to the traditional system where data is saved in a direct-attached storage.

Cassandra performance on the hyperconverged system was below the expectations on almost all workloads. The only workload where our proposal performed better than direct-attached storage was in Heavy Update workload above 70 concurrent threads, where the hyperconverged system had 56% better performance than on local storage.

The difference in performance results between Cassandra and MongoDB can be explained by the way these NoSQL databases store their data in the file system. MongoDB stores data in large files, while Cassandra does it in several small files. Storage performance differences between using large and small files is already known in the literature due to overhead imposed by the DFS (Roch et al., 2018).

We also note that the performance of Cassandra is changed by the disk configuration used in GlusterFS. Significant differences in throughput value for the same workload and number of threads can been observed in the results obtained with the YCSB-B, C and D workloads on Figure 3. This behavior indicates that workloads operating with small files in GlusterFS have different performance depending on the volume disk configuration used.

It should be mentioned that during the experiments no network congestion, high CPU usage or high memory usage were observed, meaning that these parameters did not interfered on the experiment's results.

5 CONCLUSIONS AND FUTURE WORK

This paper presented a storage performance analysis on a hyperconverged infrastructure that uses an open-source DFS to store and replicate data between multiple servers while using the computing resources of the same servers to host containers and store its persistent data. The proposal was evaluated under several different scenarios, using 2 NoSQL databases, 9 workloads, 3 disk configurations and 11 different number of concurrent sessions (threads), compared to a traditional direct-attached storage infrastructure.

The results show that applications working with large files in the hyperconverged system performs similarly to traditional local storage. But a hyperconverged system adds desired features for cloud computing systems when compared to conventional storage, like scalability, elasticity and fault-tolerance. A hyperconverged system can grow easily by adding more servers, which may have different computing and storage configurations. Meanwhile in a traditional architecture, new stand-alone servers can be added to the infrastructure, but computing and storage resources are confined to each server. Besides that, a small hyperconverged system can easily bypass the resources of a traditional local storage-based system.

We believe that the use of our proposal of hyperconverged infrastructure is feasible for cloud service providers in the provision of IaaS services (e.g. vitual machines and containers) and PaaS services (e.g. containerised databases). An user's application that requires more storage resources than a single server can provide, has challenges for scaling in a non-convergent system but would have the resources allocated easily in a hyperconverged architecture. This type of feature is desired by datacenters from cloud providers because it optimizes infrastructure by allowing two resource pods (computing and storage) to collapse into one, increasing efficiency.

The network requirements for the servers of a hyperconverged infrastructure must be properly sized to minimize the possibility of congestion. The throughput between the server and its disks should occur without limitations imposed by throughput of the network. As the scale of the hyperconverged system built for the experiments was small, the network infrastructure did not interfere with the results. However if the experiment's testbed scale were larger, then network resources would need to be increased.

In further research, we intend to evaluate our proposal with bigger clusters, other hypervisor solutions, different type of disks (e.g., SSD), server configurations and workloads.

REFERENCES

Abramova, V. and Bernardino, J. (2013). Nosql databases: Mongodb vs cassandra. In *Proceedings of the international C* conference on computer science and software engineering*, pages 14–22. ACM.

Azagury, A. C., Haas, R., Hildebrand, D., Hunter, S. W., Neville, T., Oehme, S., and Shaikh, A. (2014). Gpfs-based implementation of a hyperconverged system for software defined infrastructure. *IBM Journal of Research and Development*, 58(2/3):6–1.

Bhimani, J., Yang, J., Yang, Z., Mi, N., Xu, Q., Awasthi, M., Pandurangan, R., and Balakrishnan, V. (2016). Understanding performance of i/o intensive containerized applications for nvme ssds. In *Performance Computing and Communications Conference (IPCCC), 2016 IEEE 35th International*, pages 1–8. IEEE.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM.

Fattahi, T. and Azmi, R. (2017). A new approach for directory management in glusterfs. In *Information and Knowledge Technology (IKT), 2017 9th International Conference on*, pages 166–174. IEEE.

Giannakopoulos, I., Papazafeiropoulos, K., Doka, K., and Koziris, N. (2017). Isolation in docker through layer encryption. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2529–2532. IEEE.

Huang, J., Badam, A., Caulfield, L., Nath, S., Sengupta, S., Sharma, B., and Qureshi, M. K. (2017). Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds. In *FAST*, pages 375–390.

Kaneko, S., Nakamura, T., Kamei, H., and Muraoka, H. (2016). A guideline for data placement in heterogeneous distributed storage systems. In *Advanced Applied Informatics (IIAI-AAI), 2016 5th IIAI International Congress on*, pages 942–945. IEEE.

Mills, N., Feltus, F. A., and Ligon III, W. B. (2018). Maximizing the performance of scientific data transfer by optimizing the interface between parallel file systems and advanced research networks. *Future Generation Computer Systems*, 79:190–198.

Roch, L. M., Aleksiev, T., Murri, R., and Baldridge, K. K. (2018). Performance analysis of open-source distributed file systems for practical large-scale molecular ab initio, density functional theory, and gw+ bse calculations. *International Journal of Quantum Chemistry*, 118(1).

Tarasov, V., Rupprecht, L., Skourtis, D., Warke, A., Hildebrand, D., Mohamed, M., Mandagere, N., Li, W., Rangaswami, R., and Zhao, M. (2017). In search of the ideal storage configuration for docker containers. In *Foundations and Applications of Self* Systems (FAS* W), 2017 IEEE 2nd International Workshops on*, pages 199–206. IEEE.

Verma, S., Kawamoto, Y., Fadlullah, Z. M., Nishiyama, H., and Kato, N. (2017). A survey on network methodologies for real-time analytics of massive iot data and open research issues. *IEEE Communications Surveys & Tutorials*, 19(3):1457–1477.

Xu, Q., Awasthi, M., Malladi, K. T., Bhimani, J., Yang, J., and Annavaram, M. (2017a). Docker characterization on high performance ssds. In *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*, pages 133–134. IEEE.

Xu, Q., Awasthi, M., Malladi, K. T., Bhimani, J., Yang, J., and Annavaram, M. (2017b). Performance analysis of containerized applications on local and remote storage. In *Proc. of MSST*.