Random and Sequence Workload for Web-Scale Architecture for NFS, GlusterFS and MooseFS Performance Enhancement

Mardhani Riasetiawan, Nashihun Amien
Department of Computer Science and Electronics
Faculty of Mathematics and Natural Sciences, Universitas Gadjah Mada, Yogyakarta, Indonesia

***Abstract*—The problem in the data storage method that can support the data processing speed in the network is one of the key problems in big data. As computing speed increases and cluster size increases, I/O and network processes related to intensive data usage cannot keep up with the growth rate and data processing speed. Data processing applications will experience latency issues from long I/O. Distributed data storage systems can use Web scale technology to assist centralized data storage in a computing environment to meet the needs of data science. By analyzing several distributed data storage models, namely NFS, GlusterFS and MooseFS, a distributed data storage method is proposed. The parameters used in this study are transfer rate, IOPS and CPU resource usage. Through testing the sequential and random reading and writing of data, it is found that GlusterFS has faster performance and the best performance for sequential and random data reading when using 64k block data storage. MooseFS uses 64k power storage blocks to obtain the best performance in random data read operations. Using 32k data storage blocks, NFS achieves the best results in random writes. The performance of a distributed data storage system may be affected by the size of the data storage block. Using a larger data storage block can achieve faster performance in data transmission and performing operations on data.**

***Keywords*—Component; network storage; container; NFS; GlusterFS; MooseFS; random workload; sequence workload**

I. Introduction

The current digital era is driven by data derived from a variety of information, both individuals, companies and governments, which is more available than ever before. Currently, various information technology-based companies produce big data [1]. Large volumes in big data require large data storage. Digital products such as Twitter can generate 7 Terabytes (TB) a day of data, while Facebook produces 10 TB of data a day. Several similar enterprise companies have the same tendency to produce data per day [2].

Big data storage and processing needs require huge resources [9]. The gap between computing and data storage is quite large. With multicore technology in the processor, the ability of the CPU has increased for big data needs. However, the ability of data storage to serve data processing and storage still experiences its obstacles. The characteristics of the physical storage media account for most of the slow data storage performance. Even though optimization has been carried out in the Input / Output (I/O) layer in data storage, the I/O layer remains volatile. Problems related to the inefficiency in data management in storage media become more visible when multi-tasking big data in a shared resource environment [3].

An architecture is needed that can match resources with storage and data processing needs [8]. The web-scale architecture can handle the fast-growing processing and storage needs efficiently without rearranging the existing architecture [4]. As computation speed increases and cluster size increases, the I/O and network processes associated with intensive data use cannot keep up with the growth and data processing speed. Data processing applications will experience latency problems from long I/O [5].

Problems in data storage methods in the network that can support data processing speed are one of the crucial issues in big data. The research was conducted by analyzing the results of observations and exploration of storage performance by comparing architectures and file systems by considering parameters such as I/O performance, seek time, memory, and network usage. The results of data analysis from this study are expected to be a reference in building a data storage center for big data needs.

II. Literature Review

Research on measuring the performance of data storage was conducted [10]. This study measures the performance of high-performance data storage systems on baremetal clouds such as AWS, Azure, CGE, and OCI that run Hadoop. The test parameters in this research are I/O, CPU, memory and throughtput. Researchers run the method for benchmarking I/O with TestDFSIO, Flexible I/O tester (fio). To test the use of CPU resources using the K-means, Terrasort, Pagerank, and Wordcount algorithms which are run on the server to be tested. Another test was the use of network resources running K-means and Terrasort using 10TB of data. The results indicate that Hadoop systems running on high-performance data storage are a good choice for building high-performance virtual clusters to process shared workloads. In the results of this study, high-performance storage performance makes a significant impact on HDFS-based workloads with a large number of virtual cores. Non-RAID storage performs six times better with increased volume and CPU per server compared to clusters containing a large number of low-end servers. With a model like the one developed by researchers,

we can get better scalability and efficiency on high-capacity servers and bare metal servers suitable for seeking increased performance and cost savings [10].

The research performed a statistical analysis of the variability of write and read operations on parallel file systems [11]. This study uses six factors to be tested, namely Application Programming Interface (API), I/O strategy, request size, access pattern, stripe size, and stripe count. The researcher used the ANOVA F-test method to generate a comprehensive model from the observations that had been made. The results of this study indicate that these various factors within the range of values evaluated affect performance in a way that is indistinguishable from the presence of errors in the experiments performed. It should be noted that the conclusions of this study are only statistically valid for the range of values considered. For other values and a different set of other significant factors can emerge.

One of the earliest and most successful distributed systems was developed by Sun Microsystems and is known as the Sun Network Filesystem (or NFS) [12]. To define NFS, Sun developed an open protocol that simply defines the message format used to communicate by the client and server so that other groups outside Sun Microsystems can develop their own NFS.

Gluster is an open source, software-only file-based NAS scale-out platform. This enables enterprises to combine a large number of commodity data storage devices and compute resources into a single storage pool resulting in high performance and centrally managed storage. Both capacity and performance can scale independently on demand, from a few terabytes to several petabytes, using on-premises hardware and public cloud storage infrastructure. Combining affordable hardware with a scaling approach, users get radically better price and performance, in an easy-to-manage solution that can be configured for the most demanding workloads.

GlusterFS is designed for several purposes such as elasticity, linear scaling, and scale-out. The elasticity in GlusterFS is the idea that an enterprise should be able to flexibly adapt to data growth (or reduction) and add or remove resources to the storage pool as needed, without disrupting existing systems.

Linear Scaling is that twice the amount of system storage will provide twice the performance. What is observed in this case is twice with the same average response time for each event in the external file system I/O system i.e., how long the NFS client waits for the file server to return the information associated with each NFS client request. Traditional filesystem models and architectures cannot be scalable in this way and therefore can never achieve performance at true linear scale.

Gluster is designed to provide a scale-out architecture for performance and data storage capacity requirements. This implies that the system must be able to increase (or decrease) along some dimensions. By combining the data storage, CPU, and I/O resources of a large number of systems with affordable hardware. If you want to add more capacity to your scale-out system, you can add more disks at a lower cost. In practice, both performance and data storage capacity can be linearly increased in Gluster.

As a distributed file system, Moose File System (MooseFS) has been widely used in industry. As the architecture built by MooseFS becomes more efficient to increase the level of data handling, companies such as Douban (a Chinese social networking service website) and Lenovo have benefited greatly from using MooseFS. MooseFS has just been released as an open-source project in the GitHub repository since 2016, and some research has been done on MooseFS [7].

MooseFS consists of four main parts: client, chunk server, master server and metalogger. The master server maintains all the metadata of the system files. The server chunk is the storage unit in MooseFS. The basic unit in data storage is called a chunk, and all chunks are managed by the chunk server. When a client wants to get services from MooseFS, the server and client first interact with the master server for metadata information retrieval operations, and then communicate with the chunk server to read or write data. Metaloggers are backups of the master server. Its main function is to periodically download metadata from the master server to be promoted as new in case of failure. In the general case, a single metalogger is sufficient to handle failures.

Metalogger stores metadata (such as permissions and last access of data) and file and directory hierarchies in master main memory and then performs a permanent copy of the metalogger. With this, MooseFS gives users the global namespace of the system. The MooseFS client accesses the file system by mounting the namespace in the local file system. The client can perform the operation by communicating to the master server which directs the client to the chunk server.

In MooseFS, each file has a purpose, namely a certain number of copies that must be preserved. When the client writes data, the master server sends it a list of server chunks where the data will be stored. Then, the client sends data to the first chunk server which instructs the other chunk servers to replicate the files synchronously. MooseFS uses msfmount to communicate between users and msfmount itself is based on Filesystem Userspace (FUSE) which means MooseFS can work with operating systems that use this mechanism such as Linux, FreeBSD, OSX and others.

III. Methodology

This section introduces the methods of studying network storage services. The method is developed around a test platform that allows us to run specific benchmarks. First, we aim to test client capabilities and network attaches storage design decisions. Then, use the test platform to measure the impact of these two aspects on performance under different workloads.

A. Overview

The environment that will be created for this research can be used as a prototype for centralized data storage and computing services. Users will be connected to the

environment via a wireless network. Docker has a non-permanent nature of data storage, so Docker needs to be integrated with external data storage methods so that data is permanently stored and accessed in a cluster. In this study, the availability of datasets prepared for computational purposes will be stored in an external file system (in this study is DFS) which is connected to the research environment created.

The data service in the environment to be built is expected to be able to produce dynamic data storage and processing. Storage and dynamic data processing in this research are that when a user accesses the service, the system will run Docker and create a directory automatically which will be tied to Docker that is running and used by the user. It is expected that by using data storage like this, the data in Docker will always be in its position when one user accesses the same data.

B. Goal

The aim of this research is to evaluate and analyse which file system is best among NFS, GlusterFS and MooseFS for big data storage and processing based on I/O performance, network transfer rate and CPU usage inside network storage cluster for data science environment.

In this study, we are using the black-box testing approach. We instrument a testbed in which one more test computers run the desired application-under-test. The Testing running several workloads inside Docker for reading and writing specific files to defined directory attached into network storage. In the meantime, testing tool writing an output file for collected data during testing execution.

We require tests to be repeatable and automated run testing scenario sequentially to get an average number of data at one time. Since we target three various network storage with write and read scenario using four different storage block, we need all scenario able to be performed without supervision and logged automatically.

C. Testbed

This study utilizes three storage clients connected for each network storage and mounted like local directory and using Docker to creating workloads. We are using three storage clients to simulate concurrent workloads running at the same time. Our test application receives benchmark test parameters, which describe the sequence of operations to be performed. Then, the test application is operated remotely through Docker to generate workloads. Network storage clusters synchronize in the background and each client detect updated file listing generated from Docker workload. Once benchmark tools running from Docker, every parameter is going generated and written in the output file and processed to measure performance metrics. The testing runs workloads thirty times for each scenario with different block size.

We plan the testbed using Linux virtual server explain in Fig. 1. The server controls the experiments by running network storage clusters and clients installed with Docker container and host benchmark tools. Both clients and network storage clusters are attached to the virtual network interface. With this setup allow the server easily to manage and observe from the virtual server host. Linux virtual servers connected with 10Gbps network link.

D. Benchmark Performance

After knowing the design choices of network storage types, we use our testbed to check their influence on performance. We plan a methodology to calculate metrics related to read and write workloads. A workload is generated by the testing application based on a benchmark definition. A variable of reading and write workload with different storage block size. Besides, workload type can be specified to random or sequential to test how service reacts to different I/O.

Performance is calculated. We calculate (i) transfer rate between clients and storage server (ii) IOPS during random and sequential workload (iii) CPU usage in operating system user-space.

E. Network Attached Storage (NAS) under Test

This study comparing three network storage and limiting the analysis to the native client using Docker workload. Table I list considered technology specification of three network storages. Each network storage using different technologies, architecture and method to work with data and metadata. In such allow us to compare the impact of the technology design and architecture to handling different data workloads.

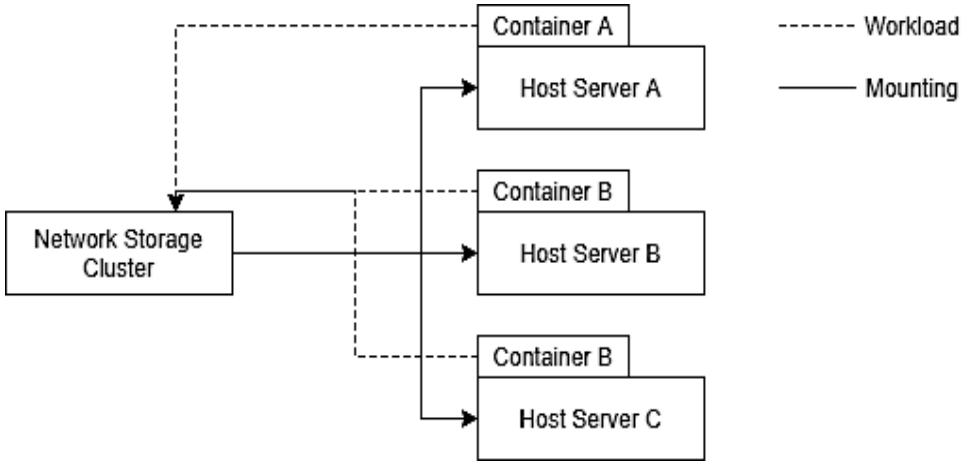


Fig. 1. Testbed Environment for Network Storage Workload Testing.

TABLE I. THE TECHNOLOGY SPECIFICATION

	NFS	GlusterFS	MooseFS
API Access	Remote Procedure Call (RPC), NFS Client	libglusterfs, Fuse, NFS, SMB, Swift, libgfapi	POSIX, FUSE MooseFS clients.
High Availability	Need additional softwear and hardware	Mirroring	Master – Slave
Architecture	Client – Server	Client – Server without metadata	Using master server to manage server
Metadata	Data and metadata are saving inside NFS	Using Elastic Hashing Algorithm instead of saving metadata	Savinng metadata inside master server
Cache	Client side	Not usning cache	Using RAM inside storage server
Fault Tolerant	Need additional softwear and hardware	Eliminates server that is in status not available	Do quarantine against the server machine who can't perform I/O operations
Replication and Synchronization	Need additional software and hardware	Not replicating directly	Keep some copy of data.

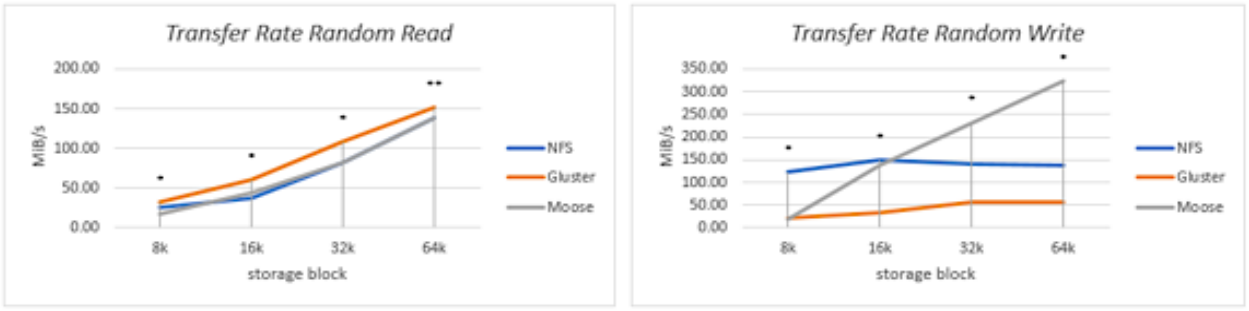
IV. RESULT AND DISCUSSION

We evaluate the testing parameter with different storage block and network storages. Workload running from Docker to read or writing test file with random or sequential to write 10GB files for each task. Our experiment checks how the network storage will handle batches workload for reading and write using four different storage block: (i) 8k, (ii) 16k, (iii) 32k, dan (iv) 64k.

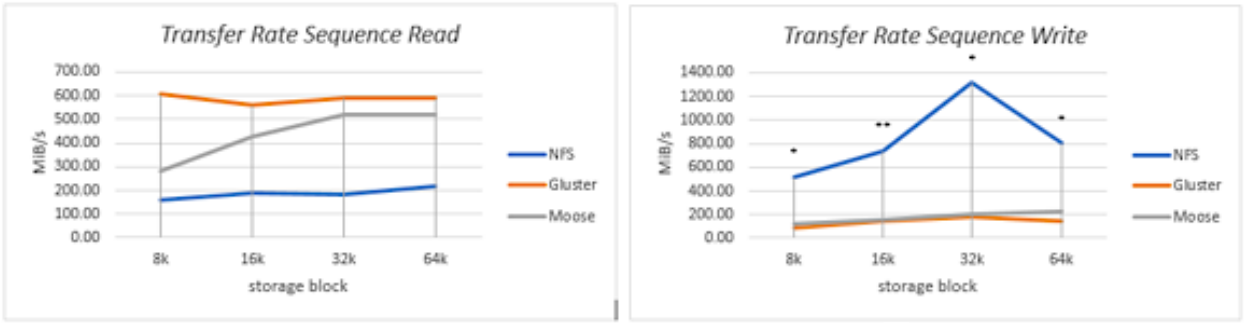
A. Transfer Rate

In data storage transfer rate can represent how many data can be transferred during the reading or writing process in one time. In general, with a higher transfer rate meaning the storage system able to process big data faster. In our methodology, we are trying to process 10 GB data with random and sequential data to be processed and transferred from the client into the storage cluster network. In Fig. 2, we can see the average result from transfer rate benchmark. In random read and sequence read, GlusterFS having better transfer rate compared to NFS and MooseFS. GlusterFS transfer rate for random read workload increase with bigger storage block size. However, GlusterFS transfer rate performance for sequential read workload not affected by the storage block size. MooseFS with sequential read workload affected with the changes in block size with 32k and the performance in 64k quite same with 32k block size.

The result for random write, MooseFS having significant performance result with 64k storage block size with reaching 300MiB/s transfer. GlusterFS and NFS transfer rate for random write performance is not affected by the block size. In Sequence read workload the NFS having better performance than GlusterFS and MooseFS. The NFS performance reaching a peak with 32k storage block size and able to be reaching 1200MiB/s. However, the performance is dropping with a 64k block size. We believe in NFS performance dropping because of the bottleneck in the networks.



(a) Average Transfer Rate for Random Read and Write.



(b) Average Transfer Rate for Sequence Read and Write.

Fig. 2. Average Transfer Rate from Read and Write for Sequence and Random Workloads.

B. IOPS

IOPS refers to the maximum number of reads and writes to non-contiguous storage locations. These operations are typically dominated by seeking time, or the time it takes a disk drive to position its read/write heads to the correct location. We are trying to measure IOPS to identify IOPS between three network storages as shown in Fig. 3.

The result for this test showing IOPS affected with storage block. In random read and sequence read workload GlusterFS having the best performance with 8k storage block size. However, the IOPS performance dropping in 16k, 32k and 64k. MooseFS with random write having a better performance with 16k but it gradually decreases in 32k and 64k block size.

In random read, sequence read, and random write with MooseFS able to reach better performance. However, IOPS performance degraded with 32k and 64k block size. NFS having same degrade performance with 16k and 32k. However, in 64k block size NFS having better result than GlusterFS and MooseFS.

C. CPU Usage

The equations are an exception to the prescribed CPU usage in user space is part of this. A user-space program is any process that doesn't belong to the kernel. Shells, compilers, databases, web servers, and the programs associated with the desktop are all user space processes. If the processor isn't idle, it is quite normal that the majority of the CPU time should be spent running user space processes. This scenario able to crash the system or the environment and we need to restart the system.

Higher CPU usage causing process stuck in the system and causing a bottleneck in the data processing. n this study we measure how efficient the network-attached storage using the processing resources.

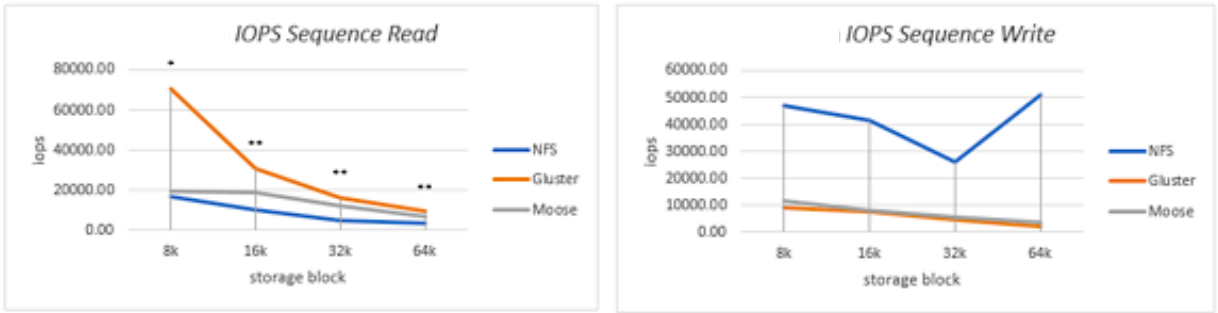
Result in Fig. 4 showing the average result of CPU usage in userspace. We found CPU usage in the three network storage affected with different storage block size. NFS showing low CPU usage in random read, random write, and sequence read workloads. However, NFS in sequence writes using high CPU resources.

GlusterFS having high usage CPU resources when running sequence read workloads with around 25% - 30% CPU usages. GlusterFS with random read, random write and sequence write are affected with storage block size as we can see in Fig. 4 GlusterFS CPU usage lower with bigger storage block size. GlusterFS and NFS having a similar pattern with CPU usage is decreasing with bigger storage block size. In the meantime, MooseFS having increase CPU usage in 32k block size workload and CPU usage dropped with bigger storage block size.

D. NFS Performance

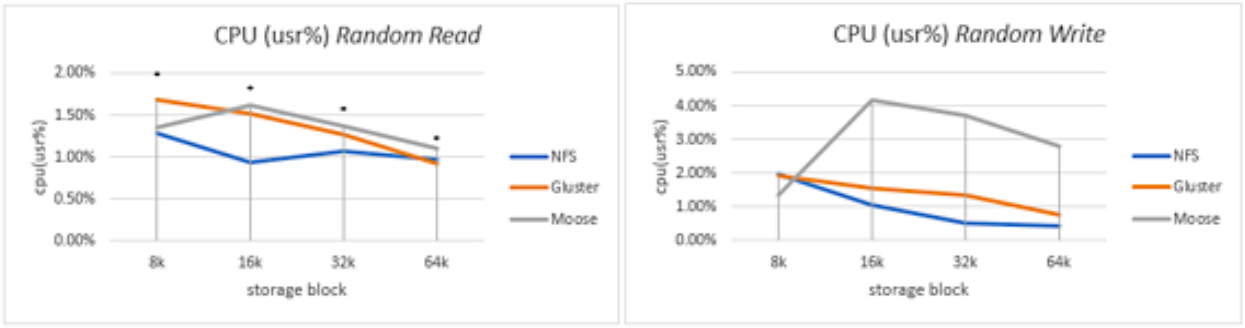
The results of the random read scenario transfer rate performance test on NFS show that 64k blocks have the highest yield, 32k in second place, then 16k and 8k in third and fourth place, as shown in Fig. 5. The results of the random write scenario transfer rate performance test on NFS show that 16k blocks have the highest yields, 32k in second place, then 64k and 8k in third and fourth place. The results of the transfer rate performance test for the sequence read scenario on NFS show that 64k blocks have the highest yield, 16k in second place, then 32k and 8k in third and fourth place. The results of the transfer rate performance test for the sequence write scenario on NFS show that 32k blocks have the highest yield, 64k in second place, then 16k and 8k in third and fourth place.

(a) Average IOPS for Random Read and Write.

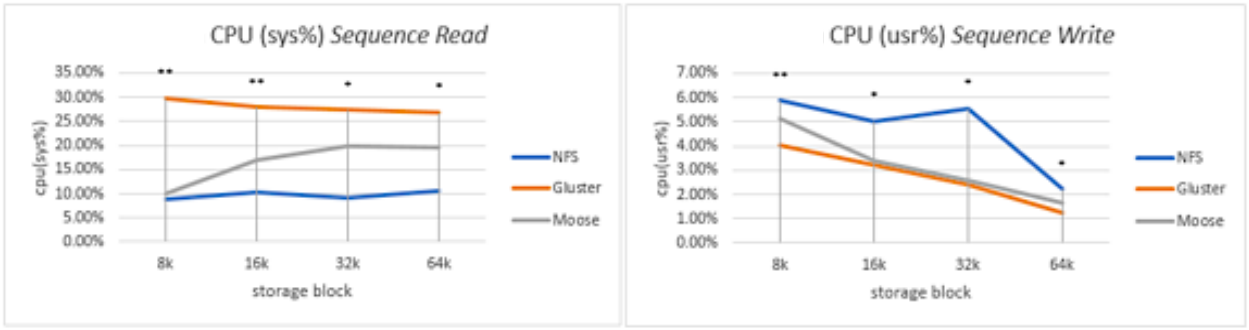


(b) Average IOPS for Sequence Read and Write.

Fig. 3. Average IOPS from Read and Write for Sequence and Random Workloads.



(a) Average CPU (usr%) for random read and write



(b) Average CPU (usr%) for sequence read and write

Fig. 4. Average CPU usage in Network Storage Performance with different Block Size.

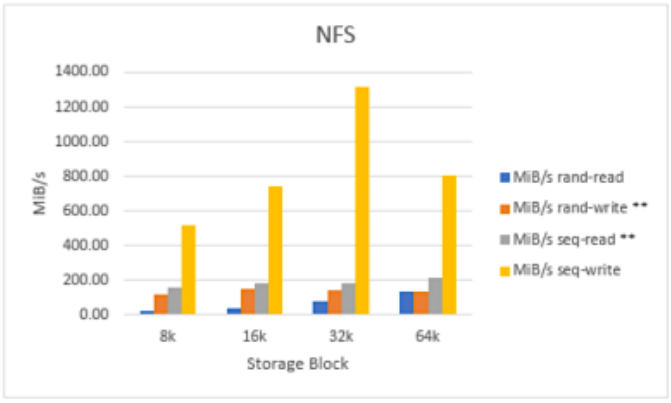


Fig. 5. Transfer Rate Performance from NFS Benchmark.

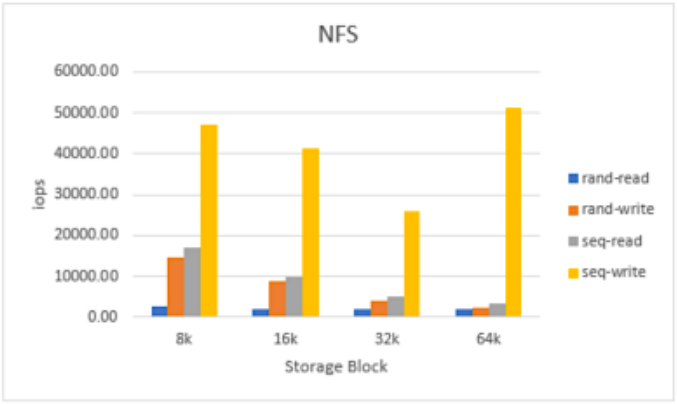


Fig. 6. IOPS Performance in NFS Workload.

The results of the random read scenario IOPS performance test on NFS show that 8k blocks have the highest results, 64k in second place, then 32k and 16k in third and fourth place. The results of the IOPS performance test in the random write scenario on NFS show that 8k blocks have the highest yield, 16k in second place, then 32k and 64k in third and fourth place, as shown in Fig. 6.

The IOPS performance test results in the sequence read scenario on NFS show that 8k blocks have the highest yields, 16k in second place, then 32k and 64k in third and fourth place. The IOPS performance test results in the sequence write scenario on NFS show that 64k blocks have the highest yield, 8k is in second place, then 16k and 32k are in third and fourth place.

The results of the CPU performance test (usr%) for the random read scenario on NFS show that the 16k block has the lowest results, 64k is in second place, then 32k and 8k are in the third and fourth place. The results of the cpu performance test (usr%) for the random write scenario on NFS show that the 64k block has the lowest yield, 32k is in second place, then 16k and 8k are in the third and fourth place. The results of the CPU performance test (usr%) in the sequence read scenario on NFS show that 64k blocks have the lowest results, 32k are in second place, then 16k and 8k are in the third and fourth places. The results of the CPU performance test (usr%) in the sequence write scenario on NFS show that 64k blocks have the lowest results, 16k are in second place, then 32k and 8k are in the third and fourth places, as shown in Fig. 7.

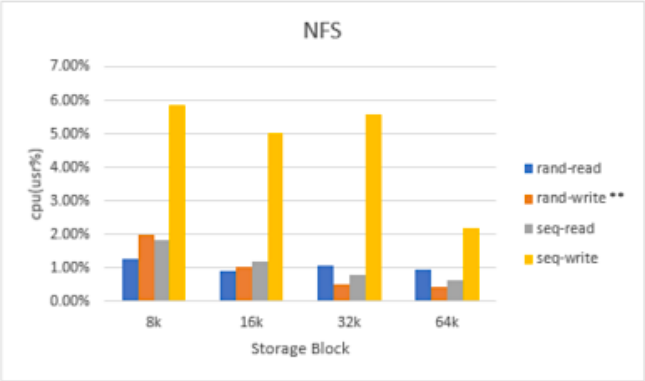


Fig. 7. CPU (usr%) Performance in NFS Benchmark.

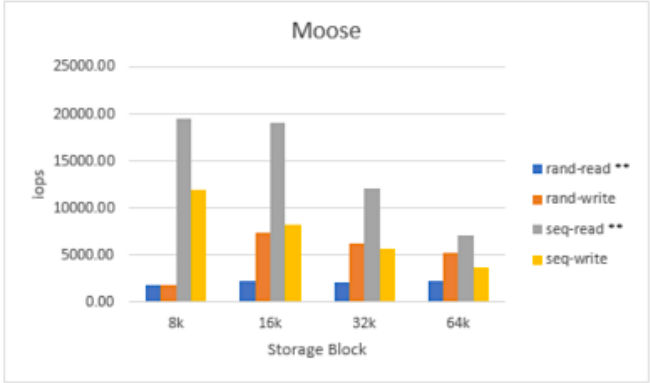


Fig. 9. Average IOPS in MooseFS with different Block Size and Workload.

E. MooseFS

Benchmark of the random read scenario transfer rate performance test on MooseFS show that the 64k block has the highest yield, 32k is in second place, then 16k and 8k are in the third and fourth place. The results of the random write scenario transfer rate performance test on MooseFS show that 64k blocks have the highest yield, 32k in second place, then 16k and 8k in third and fourth place, as shown in Fig. 8.

The results of the transfer rate performance test for the sequence read scenario on MooseFS show that 32k blocks have the highest yield, 64k in second place, then 16k and 8k in third and fourth place. The results of the transfer rate performance test for the sequence write scenario on MooseFS show that 64k blocks have the highest yield, 32k in second place, then 16k and 8k in third and fourth place.

The results of the random read scenario IOPS performance test on MooseFS show that the 16k block has the highest yield, 64k in second place, then 32k and 8k in third and fourth place. The results of the IOPS performance test in the random write scenario on MooseFS show that 16k blocks have the highest yield, 32k in second place, then 64k and 8k in third and fourth place. The IOPS performance test results in the sequence read scenario on MooseFS show that 8k blocks have the highest yields, 16k in second place, then 32k and 64k in third and fourth place. The IOPS performance test results in the sequence write scenario on MooseFS show that 8k blocks have the highest results, 16k are in second place, then 32k and 64k are in third and fourth places, as shown in Fig. 9.

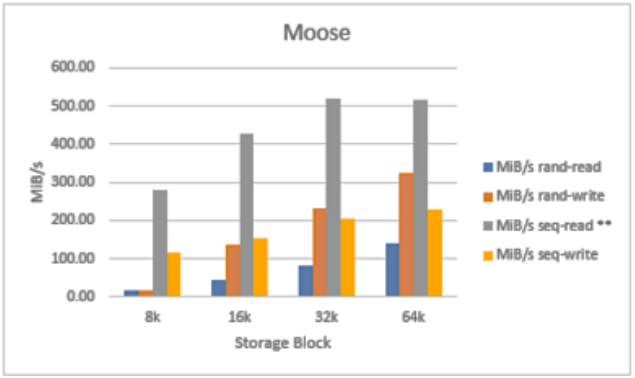


Fig. 8. Average Performance Transfer Rate in MooseFS with different Blocksize and Workload.

The results of the CPU performance test (usr%) for the random read scenario on MooseFS show that 64k blocks have the lowest results, 8k is in second place, then 32k and 16k are in third and fourth place. The results of the cpu performance test (usr%) of the random write scenario on MooseFS show that the 8k block has the lowest result, 64k is in second place, then 32k and 16k are in the third and fourth place. The results of the CPU performance test (usr%) in the sequence read scenario on MooseFS show that the 64k block has the lowest results, 32k is in second place, then 16k and 8k are in the third and fourth place. The results of the CPU performance test (usr%) in the sequence write scenario on MooseFS show that 64k blocks have the lowest results, 32k are in second place, then 16k and 8k are in the third and fourth places, as shown in Fig. 10.

F. Random Read Workload in NFS, GlusterFS and MooseFS

In the random read test, GlusterFS has good performance at data transfer speed and bandwidth, resulting in higher iops than NFS and MooseFS. The high performance of iops causes the runtime value to be smaller, which means that in jobs that require random data reading, GlusterFS can complete it quickly. GlusterFS's speed in handling reading work can be influenced by the GlusterFS architecture that propagates metadata and breaks data files into smaller data on all machines in one cluster of distributed storage systems [6]. The performance results from NFS and MooseFS in the random data reading test produce almost the same values in blocks of 16k, 32k, and 64k except for CPU usage.

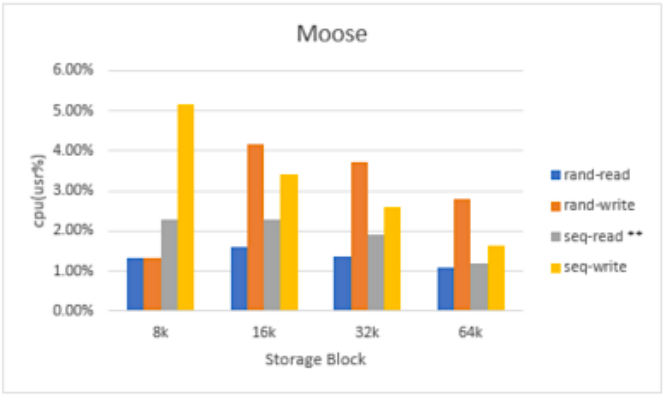


Fig. 10. Average CPU (usr%) in MooseFS with different Block Size and Workload.

The use of CPU resources in the random read test shows that NFS uses the least amount of CPU resources on both the user CPU and the CPU system. GlusterFS experienced a decrease in user and system CPU resource usage according to the larger block being tested. MooseFS in this study experienced an increase in CPU user usage in 16k blocks and after that experienced a decrease in CPU user usage on 32k and 64k blocks. The use of the CPU system on NFS, GlusterFS and MooseFS tends to increase in proportion to the larger the data storage block.

G. Random Write Workload in NFS, GlusterFS and MooseFS

In the random write test on MooseFS, the transfer rate and bandwidth performance always increase for each block tested, resulting in a small runtime. The lowest performance on the random write test is GlusterFS. On the performance of NFS IOPS, the performance decreases on each block.

MooseFS on random write testing with high bandwidth and transfer rate performance results in high CPU system and user usage. In the random write test on MooseFS, it is carried out in parallel which is synchronized by the master server and therefore must be sequential so that writing data requires a lot of processing. So the CPU load depends on the number of operations and RAM on the total number of files and folders, not the total size of the files themselves. RAM usage is proportional to the number of entries in the file system because the master server process stores all metadata in memory [7].

H. Sequence Read in NFS, GlusterFS and MooseFS

Testing on a distributed file system on sequence read performance resulted in different performance from the random read. The results of this test show that GlusterFS has the best performance for sequential file reading. The value of GlusterFS bandwidth and transfer rate in the sequence read test results in high-performance values. GlusterFS has decreased performance on IOPS with each increase in the block size of data storage so that GlusterFS tends to experience a decrease in the system and user CPU resource usage.

In MooseFS, the transfer rate and bandwidth performance increase with each increase in the block being tested so that it affects the CPU resource usage which also increases for each block tested. In the NFS test results, the performance is below GlusterFS and MooseFS. The bandwidth performance and transfer rate appear to decrease in each block. The decline in performance on NFS is caused by a bottleneck that occurs in each block. When multiple NFS clients read data from an NFS server, there may be a winner-lose pattern in which the network bandwidth is unfairly distributed among clients. This winner-lose pattern is included in an unexpected scenario because in this experiment using the same tools and operations in running the test.

I. Sequence Write in NFS, GlusterFS and MooseFS

In the sequence write test, NFS has better performance when compared to MooseFS and GlusterFS which produce almost the same test value. Based on the results of the NFS test data, performance has decreased when writing data to data storage with 64k blocks. Average CPU usage across all distributed file systems tends to decrease as the data storage block under test gets larger. The use of the CPU system on NFS has increased along with the increase in bandwidth and data transfer performance.

In MooseFS and GlusterFS, the performance of almost all parameters is below that of NFS. In this experiment, the performance of GlusterFS and MooseFS is below NFS possible because of the metadata function that needs to be processed and distributed in GlusterFS and MooseFS to all servers so that there is a possibility of increasing execution on the server to write data.

V. Conclusion

In this paper, a studied network attaches storage for web-scale infrastructure proposed for data science environment. Distributed data storage systems can assist in centralized data storage for data science needs with a computing environment with webscale technology. The distributed data storage method has proposed by analyzing several distributed data storage models, namely, NFS, GlusterFS, and MooseFS. The parameter used in this study is the transfer rate, IOPS, and CPU resource usage.

By testing the work of reading and writing data sequentially and randomly, it found that GlusterFS's performance was faster in reading data both sequentially and randomly with the best performance using 64k block data storage. MooseFS achieves the best performance on random data read jobs using 64k blocks of power storage. NFS gets the best results in random writing using 32k blocks of data storage.

Performance of the distributed data storage system can be affected by each size of the data storage block. With larger data storage block used, faster the performance can be in terms of data transfer and execution of operations on the data. However, this also implies greater use of resources.

Based on the test results, it can be concluded that each distributed data storage system has its advantages in every job. GlusterFS can achieve good performance on data reading jobs, NFS can obtain the best performance on data writing jobs in order, and MooseFS achieves the best performance at reading data sequentially.

Acknowledgment

The research has supported from Magister Program of Computer Science, Department of Computer Science and Electronics, Faculty of Mathematics and Natural Sciences, Universitas Gadjah Mada. The work one part of the research on GamaCloud infrastructures in the thesis work.

References

[1] P. China Venkanna Varma, K. Venkata Kalyan Chakravarthy, V. Valli Kumari, and S. Viswanadha Raju, “Analysis of a Network IO Bottleneck in Big Data Environments Based on Docker Containers,” Big Data Res., vol. 3, pp. 24–28, 2016, doi: 10.1016/j.bdr.2015.12.002.

[2] P. Zikopoulos and C. Eaton, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, 1 st. McGraw-Hill Osborne Media ©2011.

[3] Mishra and A. K. Somani, “Host managed storage solutions for Big Data,” no. February, 2018, doi: 10.13140/RG.2.2.19115.90406.

[4] A. Leibovici, “Understanding Web-Scale Properties,” 2014.

[5] J. Wang, J. Yin, D. Han, X. Zhou, and C. Jiang, “ODDS- Optimizing Data-locality Access for Scientific Data Analysis,” IEEE Trans. Cloud Comput., 2017, doi: 10.1109/TCC.2017.2754484.

[6] B. Depardon, L. Mahec, S. Cyril, B. Depardon, L. Mahec, and S. Cyril, “Analysis of Six Distributed File Systems,” 2013.

[7] Y. Fang, H. Zhu, and G. Lu, “Modeling and Verifying MooseFS in CSP,” 2018 IEEE 42nd Annu. Comput. Softw. Appl. Conf., pp. 270–275, 2018, doi: 10.1109/COMPSAC.2018.00043.

[8] E. Mohammed Mahmoud Nasef, and N Azaliah Abu Bakar, “Enterprise Architecture “As-Is” Analysis for Competitive Advantage”, International Journal of Advanced Computer Science and Application (IJACSA), vol 11, issue 7, 2020.

[9] I. Shabani, E. Meziu, B. Berisha, and T. Biba, “Design of Modern Distributed Systems based on Microservoces Architecture”, International Journal of Advanced Computer Science and Application (IJACSA), vol 12, issue 2, 2021.

[10] Lee, H. & Fox, G.C., 2019, Big Data Benchmarks of High-Performance Storage Systems on Commercial Bare Metal Clouds, 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), 1–8.

[11] Inacio, E.C., Barbetta, P.A., Systems, A., Inacio, C., Barbetta, P.A. & Dantas, A.R., 2017, A Statistical Analysis of the Performance Variability of Statistical Analysis of the on Performance Variability of Read / Write Operations Parallel File Systems, Procedia Computer Science, 108, 2393–2397. http://dx.doi.org/10.1016/j.procs.2017.05.026.

[12] Sandberg, R., 2000, The Sun Network Filesystem : Design , Implementation and Experience, , 1–16.