A look into computational time of replication enabled on a distributed file system

UP896692@myport.ac.uk

Abstract — Distributed file systems are a key backbone to the ever growing big data and storage industry. Computational time of operations in distributed systems is an important factor when dealing with large data sets. This report looks into the time taken for an example normal and large load against a chosen distributed file system.

Index Terms — GlusterFS, FIO, I/O, distributed system, computational time, replication, data distribution, typology.

Introduction

Distributed file systems are an important backbone to the internet's storage solutions. A distributed file system (DFS) works by aggregating a number of nodes with connected storage into a single global namespace where all are able to communicate and work together. Whether it be localhost, local area network (LAN) or wide area network (WAN) a DFS is able to keep up high availability and scalability for clients needs with many more to offer on the table (Weka, 2021).

A focusing factor of DFS is that of replication and redundancy. If a DFS is host to a large number of nodes and for example a couple of those nodes fail then the data needs to be still accessible for use by clients. In order to keep a seamless user experience for a client both factors of replication and redundancy must be taken into consideration otherwise you may have an interrupted service and/or loss of data.

I Problem

High performance cluster computing which specifically for DFS' is more critical than ever due to the ever growing size of data is in need of the fastest time of completion possible. Due to the growing industry in big data and online storage solutions, developers and companies will be looking for this fastest time of completion (Macko & Hennessey, 2022). With further room for improvement in DFS, in larger systems a slower or faster computational time in a high workload could be the difference between weeks or hours of transfer time.

With the use of replication which is there to provide a fault tolerant system a DFS could be hindered in its computational time performance however providing an assurance for high availability and minimal data loss. In this report a look into a non replicated and replicated DFS setup will be used to compare the difference in computational time of IO operations between cluster and client.

II Aims and Objectives

This report aims to:

I Compare the computational time of a DFS with 3 nodes using:
  - i No replication
  - ii 2 replication
  - iii 3 replication

II Measure the computational time of sequential reads against each candidate

III Measure the computational time of sequential writes against each candidate

This report will have the following objectives:

I Have a clear result comparison for each test against each candidate

II Have different use cases for each random read / write for normal and high workload.

III Background Research

A Distributed Data

As mentioned in the introduction a DFS has a global namespace where a client is able to mount to a shared storage volume. Files are then uploaded to the point where they are distributed among the connected nodes in the cluster (Microsoft, 2016). This should take the network communication and the overhead time of completing the read, write and readwrite operation committed between client to cluster and internodes. When committing a further process to the mount point either the cluster will wait till the first commit is finished or thread as many as it can handle at one time.

If a node or more fails during this time of use the system will degrade and this is where the service will become unavailable and data may be lost (GlusterFS, 2022). Distributing a file system workload is very important for a seamless service, by having an extremely fast performing but high risk system the client is risking its committing data.

B Replication

Within a DFS replication is the act of making a copy of any data file that makes its way into the storage volume. Having replication makes a system much more reliable during a time of failure, for instance if a file has been replicated three times and two nodes fail the client will still have access to that third copy of the file which allows for no interruption of use but possibly degraded performance of the DFS (Kangasharju, 2010).

Consistency is a big important factor for replication to work correctly. If a replica is not up to date with the other replicas within the system then all of the replicas will or may not be identical meaning that the system could provide a client an incorrect replica of the file. Thus a degraded performance of the system. Consistency is a contributing factor to the importance of replication within a distributed system and has continued research into the specific area (Naik, 2021).

C Erasure Coding

Another type of data redundancy being erasure coding is the splitting of data into fragments (data blocks) and creating additional extra fragments (parity blocks) that can be used for data recovery in the event of a failure (Sheldon, 2021). This is great for minimising the taken file space on the system as if you have data which you replicate x times on a cluster of 10 nodes you will need x amount times of storage reserve to meet your desired total. With erasure coding the data can be divided into y pieces of equal size and applied to an example erasure coding level 3, the data will now be $y + 3$. This $y + 3$ is distributed among nodes where if 3 of the servers fail the file can be reproduced and still accessed (Weatherspoon & Kubiatowicz, 2002). This means erasure coding uses less storage space than replication.

This is not without its disadvantages as the complexity of coding will bring out more bugs, setup time and troubleshooting for the peak performance that the client may need. The performance of erasure coding can depend on the amount of divisions (y) that you specify. Since when a client is specifically reading latency is going to increase due to the y machines that are needed to fetch the data all with their own internode latency. Therefore having a negative impact on computational time.

D Distributed File System Examples

Some examples of DFS that are different in typology type, the availability (how well does the cluster handle failure) and data redundancy techniques that are on offer.

Name	Typology	Data redundancy available (from background section)
MooseFS	Peer-To-Peer	A/B/C
GlusterFS	Master	A/B/C
Ceph	Master	A/B/C
IPFS	Peer-To-Peer	A
Lustre	Master	A/B/C

Table 1: Distributed file system types and examples.

E Client Use Cases

DFS are used in many areas of industry and solutions for the client to interact with. It depends upon what type of work the client is doing that the workload may differ. Clients may be a role in which many small files are written to a DFS or where a small number of large files. With the type of access for the client being higher in reads or higher in writes than the other type of specific work case (Roselli et al., 2000). This can greatly affect the computational time outcome of the cluster depending on the amount of a given command that is occurring at once.

IV Experimental Design

A Discussion

The aim of the experiment is to have a replicable way of comparing the computational time of replication on a DFS. The **hypothesis** is that replication should not have a negative effect on the reads of a system however the writing process may be negatively affected. This will be compared with the collected results. The variables within this type of experiment will consist of the DFS (**controlled**), the amount of replication setup (**independent**) and the type of data operation conducted against the system (**independent**). The DFS being the same type for all tests. The replication setup will be changed for both types of data operation conducted against the system. The time taken for operations to complete (**dependent**) will be measured for all the tests.

The chosen DFS in this experiment will be GlusterFS (GlusterFS, 2022), this is due to the versatility and simplicity of setup. The operations conducted against GlusterFS will consist of sequential reads and writes, this is to provide a repeatable test that should not interfere with any randomness created during the execution of the operation. Furthermore a normal and high use case for each read and write operation will be conducted in order to compare small and large file size. This will be done with the use of FIO (Axboe, 2022) which is a flexible and highly configurable input and output tester.

A cluster of 12 GlusterFS nodes will be set up each on a Ubuntu 18.04 image where a client machine will mount to the setup volume type. Each node will have a dedicated hard drive disk for no interruptions against the disk. FIO will be deployed against the created volume to run the normal and high read and write operations. This will be repeated for each replication type and each test being conducted three times with the average being used for comparison (3x reads, 3x writes per replication type). Each read and write will have a normal single file load of 4k byte block size with 10 megabyte file size and high case will have a single file load of 4k byte block size with 1 gigabyte file size (due to limitations of the available hardware and network speeds). This has been considered with the client use cases information researched previously. FIO has a detailed result output of the conducted test, the time will be taken from this output. Results will be compared within a bar chart style output for easy comparison of the data. This procedure is outlined in figure 1.

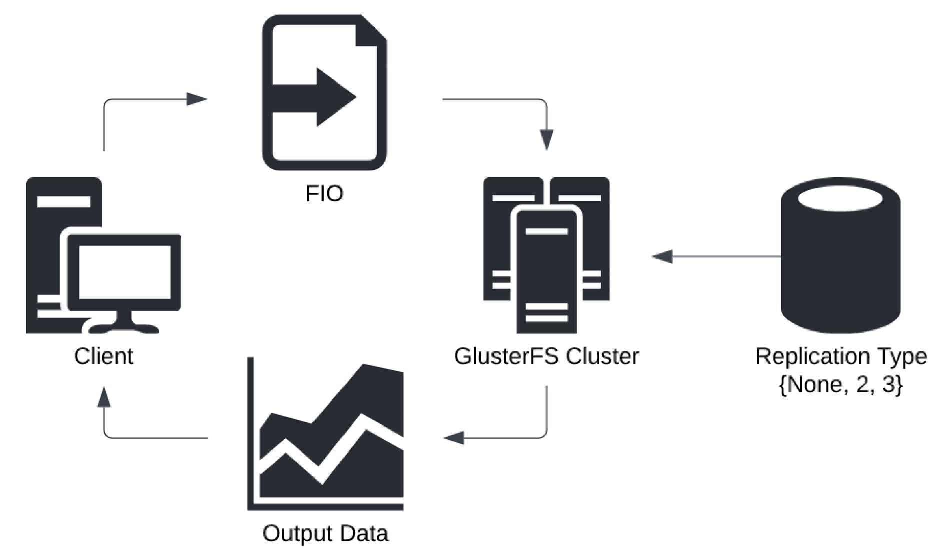

Figure 1: The method of operation generation and result collection against GlusterFS

B Method and Analysis

Simulation one is No storage volume replication, simulation two is 2 storage replication enabled and simulation three being 3 storage replication enabled:

I Normal and high workloads sequential read and write run against GlusterFS in each simulation.

Results Analysis:

I Time taken compared within a chart, average of each simulation read, write taken and plotted.

V Results

A Runtime Data

	mode	low min	low avg	low max
no replication	read	342.0 msec	355.3 msec	373.0 msec
2 replication	read	257.0 msec	293.7 msec	312.0 msec
3 replication	read	138.0 msec	324.3 msec	587.0 msec
no replication	write	804.0 msec	804.7 msec	805.0 msec
2 replication	write	1597.0 msec	1598.3 msec	1599.0 msec
3 replication	write	2387.0 msec	2388.7 msec	2390.0 msec

Table 2: Raw data of low workload time taken (msec) against GlusterFS.

	mode	high min	high avg	high max
no replication	read	439012.0 msec	439820.0 msec	440518.0 msec
2 replication	read	430869.0 msec	432008.3 msec	433064.0 msec
3 replication	read	430490.0 msec	460182.7 msec	501331.0 msec
no replication	write	87349.0 msec	87350.0 msec	87351.0 msec
2 replication	write	178904.0 msec	178921.7 msec	178957.0 msec
3 replication	write	270339.0 msec	270442.0 msec	270514.0 msec

Table 3: Raw data of high workload time taken (msec) against GlusterFS.

B Graphical Data

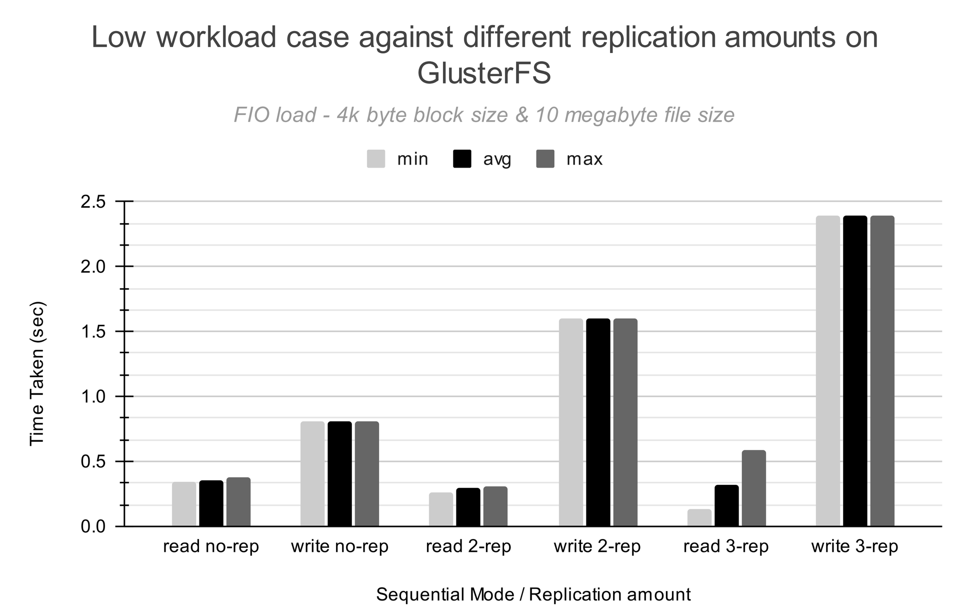

Figure 2: Converted and plotted low workload time taken (sec) against the different types of setup replication on GlusterFS

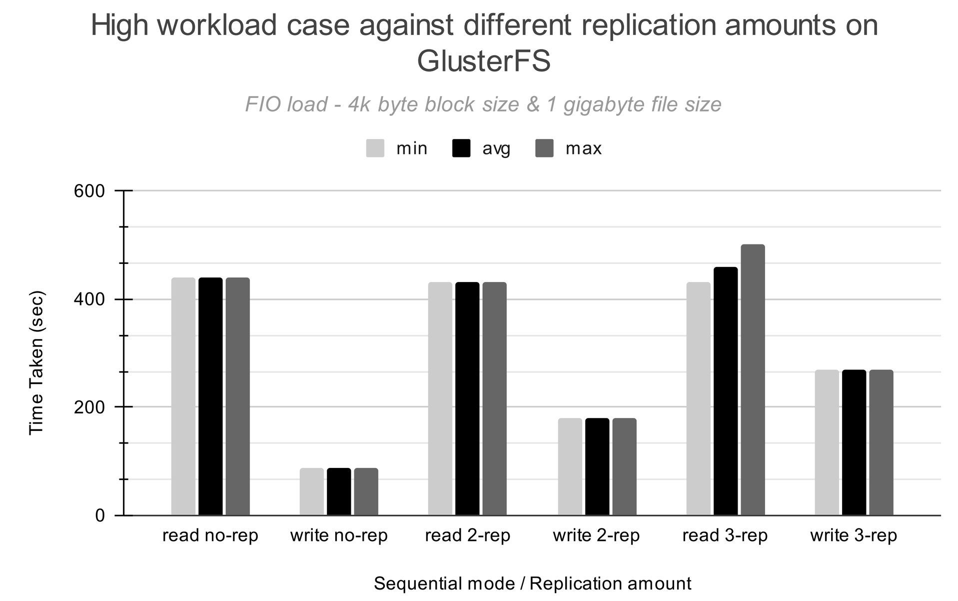

Figure 3: Converted and plotted high workload time taken (sec) against the different types of setup replication on GlusterFS

All results were gathered against the different replicated volumes in a timely manner without any issues. Each operation with workload type was run 3x times with the minimum, average and maximum calculated from this. Due to the FIO output being in milliseconds this is what the raw output data is inside of the tables 2 and 3, this has been converted into seconds within graphs 2 and 3.

With the graphs in a bar chart style approach allows for easy interpretation of results, the time taken is in seconds to make the results readable and simplified for understanding. Each is titled separately and a small description of load ran against each relocation volume. Finally all of the collected results are not in one large bar chart as this would cause confusion when interpreting with a large number of bars with a completely different scale of time taken.

VI Result Discussion

The results show that the reads of each workload is not impacted very much by the amount of replication.The hypothesis mentioned in the design discussion (IV.A) has been met which for the assumption is promising, however this may not meet other DFS typologies. It is clear that writes of each case cause time taken to increase with the amount of replication enabled on the DFS. In both read workload cases of the 3 replication it is seen that the min and max values are varying compared to the other results. This could be due to the latency between nodes and placement of data as each node will be communicating differently in each test to deliver the content of the file to the client. It may be a further trend when looking into increased amounts of replication on the cluster eg 4 or 5. It should be noted that the results presented represent the replication on GlusterFS which is a peer-to-peer typology and may not represent the master typology set of DFS.

VII Conclusion

It is clear from both low and high workload operations that the amount of replication does have an impact on computational time. The differences in typology may cause results on master typology to differ vastly. This is due to the communication mostly being done from client to master, where the metadata about file locations are stored and managed. This of course would then have another level of network latency and time taken to fetch then retrieve. This work has found that computational time on a peer-to-peer example DFS does increase computational time with increased replication. The trade off between total redundancy and fastest computational time is ultimately a key deciding factor for the administrator of the system to decide.

If conducting this test again I would compare a master and peer-to-peer topology DFS, whilst also comparing the random I/O operations which could be further analysed. The methodology used was solid and very repeatable with the process being automated for each test (via a python script).

VIII Future Work

Distributed systems is a very large and expandable topic, DFS is a small fragment of this. The method layed out in this research could be applied against other areas of distributed systems not in the form of read and write but for example packets in and out of a network with the computational time measured. A look into erasure coding which discussed in the background (III.C) could be used as a measuring point of computational time. Where the results could then be compared against this study. Otherwise an overall method of rating DFS for consumers to choose upon could be built with this method as part of a larger picture.

References

Axboe, J. (2022). *Flexible I/O Tester*. https://github.com/axboe/fio
GlusterFS. (2022). *Architecture - gluster docs*. https://docs.gluster.org/en/main/Quick-Start-Guide/Architecture/#types-of-volumes
Kangasharju, J. (2010). *Replication and consistency fall 2010 jussi kangasharju*. https://www.cs.helsinki.fi/u/jakangas/Teaching/DistSys10/DistSys-10f-4.pdf
Macko, P. & Hennessey, J. (2022). Survey of distributed file system design choices. *ACM Transactions on Storage (TOS)*, *18*(1), 1–34.
Microsoft. (2016). *Dfs namespaces and dfs replication overview*. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/jj127250(v=ws.11)
Naik, N. (2021). Comprehending concurrency and consistency in distributed systems. *2021 IEEE International Symposium on Systems Engineering (ISSE)*, 1–6.
Roselli, D., Lorch, J. R. & Anderson, T. E. (2000). A comparison of file system workloads. *2000 USENIX Annual Technical Conference (USENIX ATC 00)*.
Sheldon, R. (2021). *Erasure coding*. https://www.techtarget.com/searchstorage/definition/erasure-coding
Weatherspoon, H. & Kubiatowicz, J. D. (2002). Erasure coding vs. replication: A quantitative comparison. *International Workshop on Peer-to-Peer Systems*, 328–337.
Weka. (2021). *Distributed file system explained — features and advantages — wekaio*. https://www.weka.io/learn/distributed-file-system/