

Question:

What distributed file system works best for small files?

1. Hadoop Filesystem (HDFS)
2. GlusterFS

Experimental setup:

I used three Ubuntu instances on Jetstream2 for the Hadoop setup. These instances served as the namenode/master node and two worker nodes. Each of these instances were m3.quad with 4 CPUs, 15 GB of RAM, and a 20 GB disk size.

For GlusterFS, I utilised my existing Google Cloud VMs, all running Ubuntu 22.04. These VMs were configured with 1 CPU, 4GB of RAM, and 10 GB of disk space. To accommodate GlusterFS, I allocated an additional 10 GB of disk space to each VM. The connectivity between the three nodes in both systems was established through SSH access.

Setting up Hadoop FS presented challenges due to outdated resources online. I encountered various issues, such as the name node failing to start correctly and difficulties identifying the worker nodes. Eventually, I configured the system to have the master node initiate the worker nodes and distribute files between them based on the specified replication factor of 2.

In contrast, setting up Gluster was relatively straightforward. Similar to Hadoop, we structured Gluster with a master node and data nodes responsible for replicating files stored in bricks. Data replication was executed to ensure no data loss, with a replication factor of three.

Side effects:

For the Hadoop setup, I have configured only a single Namenode and omitted the secondary Namenode, which is typically crucial in case the master node fails. While this omission doesn't have immediate side effects, ideally, it should be included for redundancy purposes.

The systems feature distinct configurations: the Hadoop system has 4 CPUs, 15 GB of RAM, and a 20 GB disk, whereas the Gluster system is equipped with only 1 CPU, 4 GB of RAM, and a 10 GB disk. This discrepancy may result in performance variations between the two systems.

Setup this environment:

I referred to the class documents and some online resources [3],[4],[5] to set up the environment.

Tests Procedure:

For Hadoop testing, I employed various file sizes including 10 MB, 64 MB, 128 MB, 1 GB, and 2 GB. I opted for these smaller sizes to observe how data is stored in larger chunks within HDFS. The 128 MB size mirrors the chunk size, allowing us to evaluate performance when data doesn't need to be broken down further. The larger sizes enable testing under conditions where data must be divided into smaller chunks, providing insights into performance under such circumstances. To ensure accuracy, we conducted write and read operations on the data server three times for each size.

Similarly, for GlusterFS testing, we utilized identical file sizes—10 MB, 64 MB, 128 MB, 1 GB, and 2 GB. These sizes were chosen for the same reasons as in Hadoop testing. After uploading the files, we delete them. To maintain consistency, we conducted the test three times for each size to validate the results.

Test Results / Accuracy:

The number of tests are validated by using the below performance formula and we find the margin error.

$$n = \left(\frac{z_c \sigma}{E} \right)^2$$

Running the test 3 times for each size was enough as the margin of error was less than 5% of the average of the timing.

HDFS Insert:

	Run 1	Run 2	Run 3	SD	Avg	Margin Error	5% of Avg
10 MB	1.98	1.831	1.845	0.0822820 353	1.8853333 33	0.0931136 1963	0.0942666 6667
64 MB	1.968	1.946	1.977	0.01594783 162	1.9636666 67	0.0180471 9975	0.0981833 3333
128 MB	2.346	2.29	2.158	0.09652633 492	2.2646666 67	0.1092330 349	0.1132333 333

1 GB	3.483	3.257	3.416	0.11607899 61	3.3853333 33	0.1313596 03	0.1692666 667
2 GB	5.691	6.159	6.138	0.26434636 37	5.996	0.2991448 458	0.2998

HDFS Remove:

	Run 1	Run 2	Run 3	SD	Avg	Margin Error	5% of Avg
10 MB	1.5	1.606	1.571	0.0540092 5847	1.559	0.06111902 228	0.07795
64 MB	1.516	1.515	1.585	0.04012895 879	1.5386666 67	0.0454115 238	0.0769333 3333
128 MB	1.485	1.485	1.489	0.00230940 1077	1.4863333 33	0.0026134 09994	0.0743166 6667
1 GB	1.467	1.449	1.471	0.01171893 055	1.4623333 33	0.0132616 0732	0.0731166 6667
2 GB	1.4	1.432	1.435	0.01939931 27	1.4223333 33	0.0219530 3285	0.07111666 667

Gluster Insert:

	Run 1	Run 2	Run 3	SD	Avg	Margin Error	5% of Avg
10 MB	0.182	0.172	0.173	0.0055075 70547	0.1756666 667	0.0062325 85608	0.0087833 33333
64 MB	0.759	0.726	0.731	0.01778576 21	0.7386666 667	0.0201270 7489	0.0369333 3333
128 MB	1.267	1.262	1.257	0.005	1.262	0.0056581 98614	0.0631
1 GB	17.306	17.044	18.095	0.54707799 57	17.481666 67	0.6190951 914	0.8740833 333
2 GB	36.803	33.869	34.559	1.53405736 5	35.077	1.7360002 52	1.75385

Gluster Remove:

	Run 1	Run 2	Run 3	SD	Avg	Margin Error	5% of Avg
10 MB	0.026	0.025	0.026	0.0005773 502692	0.0256666 6667	0.0006533 524986	0.0012833 33333
64 MB	0.041	0.038	0.039	0.00152752	0.0393333	0.0017286	0.0019666

				5232	3333	0823	66667
128 MB	0.049	0.05	0.05	0.00057735 02692	0.0496666 6667	0.0006533 524986	0.0024833 33333
1 GB	0.319	0.327	0.347	0.01442220 51	0.331	0.0163207 4018	0.01655
2 GB	0.574	0.602	0.587	0.01401189 97	0.5876666 667	0.0158564 223	0.0293833 3333

This table shows the overall average of all the test results between HDFS and Gluster.

	10 MB		64MB		128MB		1GB		2GB	
	Insert	Remove	Insert	Remove	Insert	Remove	Insert	Remove	Insert	Remove
HDFS	1.88	1.55	1.96	1.53	2.26	1.48	3.38	1.46	5.99	1.42
Gluster	0.17	0.02	0.73	0.03	1.26	0.04	17.48	0.33	35.07	0.58

Conclusion:


Based on the above table, we observe that Gluster outperforms HDFS for smaller file sizes (10 MB and 64 MB) in both insert and remove operations. For medium file sizes (128 MB and 1 GB), Gluster maintains an advantage over HDFS in both insert and remove operations. However, for the largest file size (2 GB), HDFS demonstrates significantly superior performance in insert operations, while Gluster remains better for remove operations. This discrepancy can be attributed to GlusterFS's utilization of a parallel I/O architecture, enabling simultaneous I/O operations across multiple nodes, thus enhancing performance, particularly for smaller file operations and scenarios with numerous concurrent read and write requests. Conversely, HDFS is engineered for handling large files, especially those associated with big data processing tasks such as Hadoop MapReduce jobs, prioritizing high throughput for sequential access patterns rather than random access, which is more prevalent with smaller files.

In conclusion, the performance contrast between HDFS and Gluster appears contingent on the file size. Gluster proves more efficient for smaller file sizes, especially in insert operations, whereas HDFS becomes increasingly efficient with larger file sizes, particularly for insert operations. These findings imply that for big data workloads involving large files, HDFS may be the preferred choice, whereas Gluster can be used for workloads which generally have smaller file sizes.

Learning:

From this experiment I learned how to set up HDFS and Gluster on a three node system. I understand when selecting a distributed file system, it's crucial to consider the specific characteristics of the workload, including file size distribution, access patterns, and concurrency requirements. Tailoring the choice of file system to match the workload can lead to optimized performance and resource utilization. Also overall, GlusterFS demonstrates superiority for smaller file sizes due to its parallel I/O architecture, while HDFS excels with larger files and sequential access patterns.

References:

- [1] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf
- [2] <https://docs.gluster.org/en/main/Quick-Start-Guide/Architecture/>
- [3] <https://sparkbyexamples.com/hadoop/apache-hadoop-installation/>
- [4] <https://www.youtube.com/watch?v=iP2Em-5Abw>
- [5] <https://ruan.dev/blog/2019/03/05/setup-a-3-node-replicated-storage-volume-with-glusterfs>
- [6]  Distributed Storage