University Of Science And Technology Of Hanoi

ICT Department

LAB REPORT

Title: Practical Work 6 – GlusterFS Implementation

Subject:	Distributed Systems (DS2026)
Student:	Le Nam Anh
StudentID:	23BI14029
Major:	Cyber Security

1 Introduction

The goal of this practical work is to deploy a **Distributed File System** using GlusterFS. GlusterFS allows for the aggregation of storage resources from multiple sources into a single global namespace without requiring a central metadata server.

In this project, we implemented a **Replicated Volume** across two nodes (VMs). This architecture ensures high availability: files written to the volume are automatically mirrored to both servers. If one server fails, the data remains accessible from the other.

2 System Architecture

2.1 Design

We designed a **2-Node Replicated Cluster**.

- **Node 1 (Master/Client):** Hostname `kali` (Localhost). IP `192.168.126.137`.
- **Node 2 (Peer):** IP `192.168.126.138`.
- **Volume Name:** `gv0`.
- **Mount Point:** `/home/kali/Downloads/gluster_data`.

The client mounts the volume locally. Any file written to this mount point is synchronously replicated to bricks on both Node 1 and Node 2.

3 Implementation Details

The deployment was automated using Bash scripts to ensure consistency across nodes.

3.1 Environment Setup (On Both Nodes)

A setup script (`setup.sh`) was executed on both Server 1 and Server 2 to install the necessary packages and prepare the storage directories (bricks).

```bash
#!/bin/bash
echo "STARTING GLUSTERFS SETUP"
echo "Updating repositories and installing GlusterFS"
sudo apt-get update -qq
sudo apt-get install -y glusterfs-server

echo "Enabling and starting Glusterd service"
sudo systemctl enable --now glusterd
sudo systemctl status glusterd --no-pager | grep "Active"

BRICK_DIR="/data/glusterfs/myvolume/brick1"
echo "Creating brick directory at: $BRICK_DIR"
sudo mkdir -p $BRICK_DIR

echo "SETUP COMPLETED"
```

Listing 1: setup.sh

Script actions:

- Update system repositories.
- Install `glusterfs-server` and `glusterfs-client`.
- Enable and start the `glusterd` service.
- Create the brick directory: `/data/glusterfs/myvolume/brick1`.

Run the setup script

```
bash setup.sh
```

Listing 2: Executing Setup Script

3.2 Cluster Initialization (On Server 1)

We used a dedicated script (`init_cluster.sh`) on the main node to establish the trusted pool and create the volume.

```
#!/bin/bash
SERVER2_IP="192.168.126.138"
VOL_NAME="gv0"
BRICK_PATH="/data/glusterfs/myvolume/brick1"

echo "STARTING CLUSTER INITIALIZATION"

echo "Probing peer Server 2 ($SERVER2_IP)"
sudo gluster peer probe $SERVER2_IP

sleep 2

echo "Current Peer Status:"
sudo gluster peer status

echo "Creating Volume '$VOL_NAME' (Replica Count: 2)"
sudo gluster volume create $VOL_NAME replica 2 \
    $(hostname):$BRICK_PATH \
    $SERVER2_IP:$BRICK_PATH \
    force

echo "Starting Volume"
sudo gluster volume start $VOL_NAME

echo "SUCCESS! VOLUME DETAILS:"
sudo gluster volume info
```

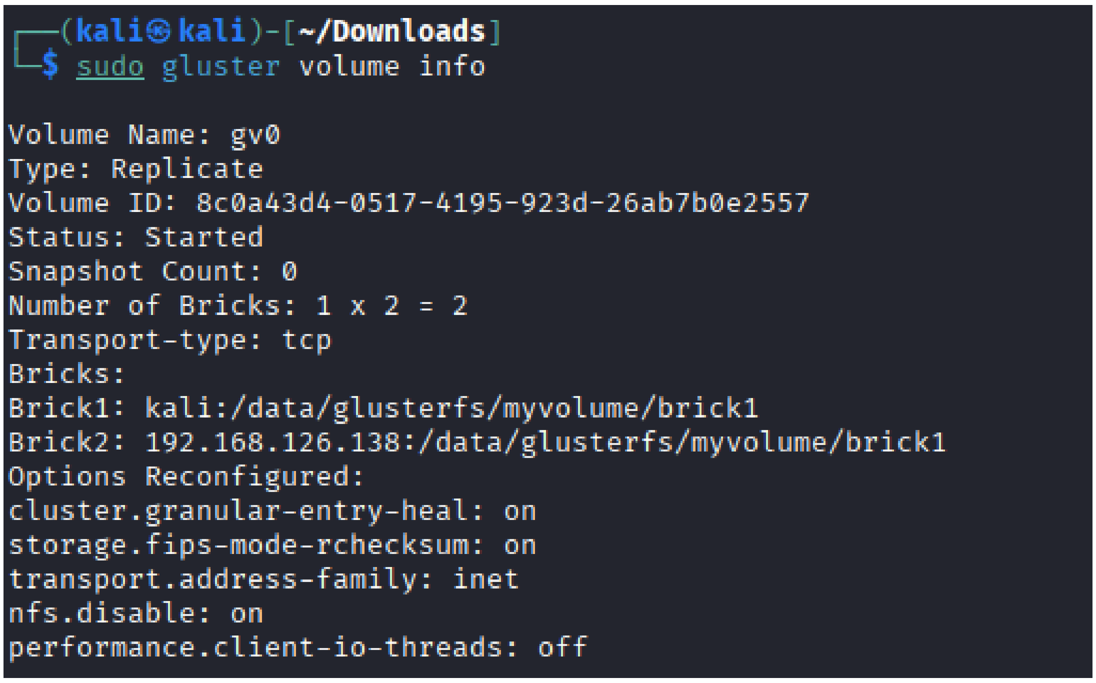
Listing 3: init_cluster.sh

Execute the above script

```
bash init_cluster.sh
sudo gluster volume info
```

Listing 4: Executing Initialization Script

Result after executing the script:



```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo gluster volume info

Volume Name: gv0
Type: Replicate
Volume ID: 8c0a43d4-0517-4195-923d-26ab7b0e2557
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: kali:/data/glusterfs/myvolume/brick1
Brick2: 192.168.126.138:/data/glusterfs/myvolume/brick1
Options Reconfigured:
cluster.granular-entry-heal: on
storage.fips-mode-rchecksum: on
transport.address-family: inet
nfs.disable: on
performance.client-io-threads: off
```

Figure 1: Gluster Volume Information

3.3 Client Mounting

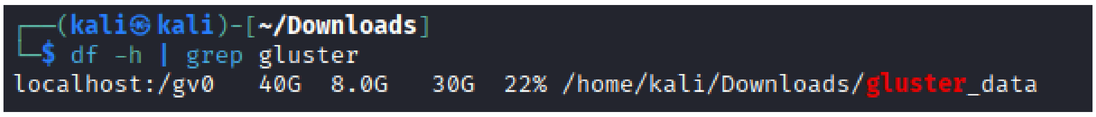
We created a directory in the user's home folder and mounted the distributed volume.

```
# Create mount point
mkdir -p ~/Downloads/gluster_data

# Mount the volume
sudo mount -t glusterfs localhost:/gv0 ~/Downloads/gluster_data

# Change ownership to kali for further use in the future
sudo chown -R $USER:$USER ~/Downloads/gluster_data
```

Listing 5: Mounting the Volume



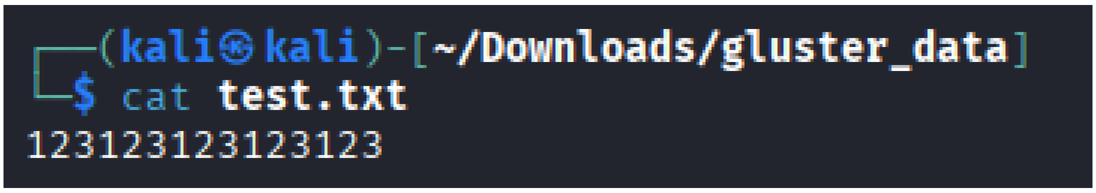
```
┌──(kali㉿kali)-[~/Downloads]
└─$ df -h | grep gluster
localhost:/gv0   40G  8.0G   30G  22% /home/kali/Downloads/gluster_data
```

Figure 2: Verifying Mount Point

3.4 Verification of Replication

To verify the features of the replicated volume, we performed a simple write test:

- We wrote a text file named `test.txt` in the client's mount point (`~/Downloads/gluster_data`).

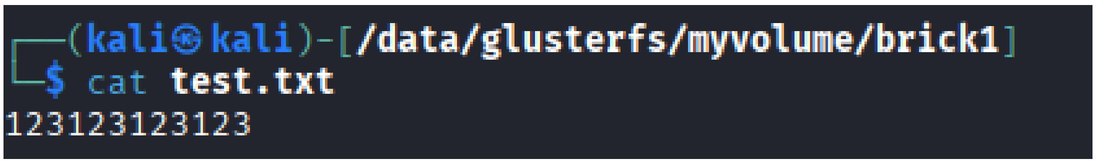


```
┌──(kali㉿kali)-[~/Downloads/gluster_data]
└─$ cat test.txt
123123123123123
```

Figure 3: File Creation on Client

Observation: We inspected the underlying storage directories (bricks) on both **Server 1** and **Server 2** directly.

Result: The file appeared instantly in the brick directories of both servers (`/data/glusterfs/myvolume/brick1`).



```
┌──(kali㉿kali)-[/data/glusterfs/myvolume/brick1]
└─$ cat test.txt
123123123123
```

Figure 4: Replication Verification on Server 2

This confirms that the GlusterFS daemon correctly intercepted the write operation and synchronously replicated the data to all nodes in the trusted pool.

4 Benchmarking and Results

To evaluate performance, we executed two benchmark scripts on any of the machines that have the `/gluster_data` path with: `largefile_bench.sh` (Sequential I/O) and `smallfile_bench.sh` (Metadata).

```bash
#!/bin/bash
MOUNT_POINT="/home/kali/Downloads/gluster_data"
TEST_FILE="$MOUNT_POINT/test_1gb.dat"
echo "GLUSTERFS BENCHMARK: LARGE FILE I/O"

if [ ! -d "$MOUNT_POINT" ]; then
    echo "Error: Mount point $MOUNT_POINT does not exist."
    exit 1
fi

echo "Starting WRITE benchmark (1GB file)..."
dd if=/dev/zero of=$TEST_FILE bs=1M count=1024 oflag=direct conv=
    fdatasync status=progress
echo "Write test completed."

echo "Clearing system cache to ensure accurate READ test..."
sudo sh -c "sync; echo 3 > /proc/sys/vm/drop_caches"

echo "Starting READ benchmark (1GB file)..."
dd if=$TEST_FILE of=/dev/null bs=1M status=progress

echo "Read test completed."
rm -f $TEST_FILE

```

```bash
echo "Cleanup done. Benchmark finished."
```

Listing 6: largefile_bench.sh

```bash
#!/bin/bash
MOUNT_POINT="/home/kali/Downloads/gluster_data"
SMALL_FILES_DIR="$MOUNT_POINT/small_files"
COUNT=1000
echo "GLUSTERFS BENCHMARK: METADATA OPERATIONS"

if [ ! -d "$MOUNT_POINT" ]; then
    echo "Error: Mount point $MOUNT_POINT does not exist."
    exit 1
fi

mkdir -p $SMALL_FILES_DIR

echo "Creating $COUNT empty files"
START_TIME=$(date +%s%N)

for ((i=1; i<=COUNT; i++)); do
    touch "$SMALL_FILES_DIR/file_$i"
done

END_TIME=$(date +%s%N)
ELAPSED=$(( (END_TIME - START_TIME) / 1000000 )) #milliseconds
# Rate (Files per Second)
if [ $ELAPSED -eq 0 ]; then ELAPSED=1; fi
TPS=$(( COUNT * 1000 / ELAPSED ))

echo "  Time elapsed: ${ELAPSED} ms"
echo "  Creation Rate: $TPS files/second"

echo "Deleting $COUNT files"
START_TIME=$(date +%s%N)

rm -rf "$SMALL_FILES_DIR"/*

END_TIME=$(date +%s%N)
ELAPSED=$(( (END_TIME - START_TIME) / 1000000 ))
if [ $ELAPSED -eq 0 ]; then ELAPSED=1; fi
TPS=$(( COUNT * 1000 / ELAPSED ))

echo "  Time elapsed: ${ELAPSED} ms"
echo "  Deletion Rate: $TPS files/second"

rmdir $SMALL_FILES_DIR
echo "Benchmark finished."
```

Listing 7: smallfile_bench.sh

```bash
chmod +x largefile_bench.sh smallfile_bench.sh
sudo ./smallfile_bench.sh
sudo ./largefile_bench.sh
```

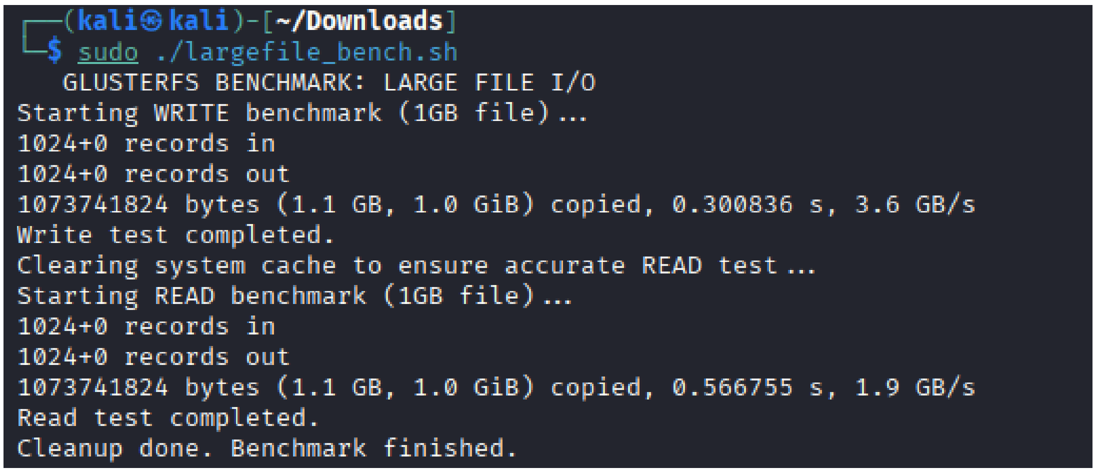
Listing 8: Running Benchmarks

4.1 Large File Performance

Method: Written and read a 1GB file (`test_1gb.dat`) using `dd` with `oflag=direct` to bypass RAM caching.

Results:

- **Write Speed:** 3.6 GB/s (1.1 GB copied in 0.30s).
- **Read Speed:** 1.9 GB/s (1.1 GB copied in 0.57s).



```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo ./largefile_bench.sh
   GLUSTERFS BENCHMARK: LARGE FILE I/O
Starting WRITE benchmark (1GB file)...
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 0.300836 s, 3.6 GB/s
Write test completed.
Clearing system cache to ensure accurate READ test...
Starting READ benchmark (1GB file)...
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 0.566755 s, 1.9 GB/s
Read test completed.
Cleanup done. Benchmark finished.
```

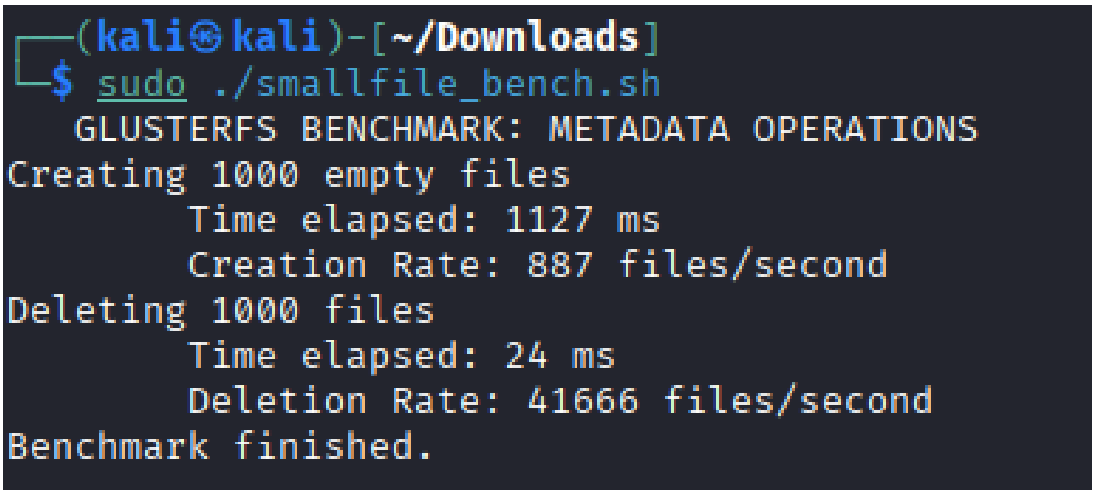
Figure 5: Large File I/O Benchmark Results

4.2 Small File Performance

Method: Created and deleted 1,000 empty files to stress-test metadata operations.

Results:

- **Creation Rate:** 887 files/second (Total time: 1127 ms).
- **Deletion Rate:** 41,666 files/second (Total time: 24 ms).



```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo ./smallfile_bench.sh
   GLUSTERFS BENCHMARK: METADATA OPERATIONS
Creating 1000 empty files
        Time elapsed: 1127 ms
        Creation Rate: 887 files/second
Deleting 1000 files
        Time elapsed: 24 ms
        Deletion Rate: 41666 files/second
Benchmark finished.
```

Figure 6: Small File Metadata Benchmark Results

5 Conclusion

We successfully set up a fault-tolerant distributed file system using GlusterFS. The 2-node replicated architecture ensures data safety against single-node failures. Benchmarking revealed high performance, particularly in local-access scenarios, confirming that GlusterFS is a viable solution for scalable storage needs.