Journal of Physics: Conference Series **513** (2014) 042014 doi:10.1088/1742-6596/513/4/042014

Testing of several distributed file-systems (HDFS, Ceph and GlusterFS) for supporting the HEP experiments analysis.

Giacinto Donvito[1], **Giovanni Marzulli**[2], **Domenico Diacono**[1]

[1] INFN-Bari, via Orabona 4, 70126 Bari
[2] GARR and INFN-Bari, via Orabona 4, 70126 Bari

E-mail: giacinto.donvito@ba.infn.it, giovanni.marzulli@ba.infn.it, domenico.diacono@ba.infn.it

Abstract. The activity of testing new storage solution is of great importance in order to provide both features and performance evaluation and give few hints to small-medium sites that are interested in exploiting new storage technologies. In particular this work will cover storage solutions that provide both standard POSIX storage access and cloud technologies; we focused our attention and our test on HDFS, Ceph, and GlusterFS.

1. Introduction

In this work we will show the testing activity carried out on aforementioned distributed open source file-systems in order to check the capability of supporting HEP data analysis.

HDFS is an Apache Foundation software and is part of a more general framework, that contains a task scheduler, a NoSQL DBMS, a data warehouse system, etc. It is used by several big companies and institutions (Facebook, Yahoo, Linkedin, etc).

Ceph is a quite young file-system that has been designed in order to guarantee great scalability, performance and very good high availability features. It is also the only file-system that is able to provide three interfaces to storage: POSIX file-system, REST object storage and device storage. Native support for Ceph was introduced in the 2.6.34 Linux kernel.

GlusterFS has been recently acquired by RedHat and this will ensure the long term support of the code. It has indeed a large user base both in HPC computing farms, and in several Cloud computing facilities. It supports access to storage both in terms of POSIX file-system and via a REST gateway for object storage support.

All those file-systems are capable of supporting high availability of the data and metadata in order to build a distributed structure that could provide resilience to the hardware and/or software failure of one or more data server in the cluster.

We will describe each file-system in detail providing the technical specification and reporting about the testing of the most interesting functionalities of each one. We will focus our attention on the capabilities to recover from failures of both hardware and software and on how each software is able to provide those capabilities, and we will describe the tests carried out on to prove them.

In this work we will also present the results of tests that will highlight the scalability of each of those file-systems. We will show also the development that we have done to provide more powerful monitoring capabilities for HDFS. We have developed a web based monitoring system that is capable to show in detail the information about the status of the data nodes or the status and the historical information about the location of each block. We will also provide detailed information on automatic scripts developed in order to easily manage a big datacenter composed of hundreds of data node installed with HDFS.

2. Description and architecture GlusterFS

GlusterFS is a storage technology that permits, starting from several volumes hosted on different servers, the construction of a distributed replicated network file-system, fully POSIX compliant, also with support of new storage paradigms such as Block Storage and Object Storage. GlusterFs stores the data on stable kernel file-systems like ext4, xfs, etc.; it doesn't use an additional metadata server for the files metadata, using instead a unique hash tag for each file, stored within the file-system itself.

In the Gluster terminology a volume is the share that the servers, that host the actual kernel space file-system in which the data will be stored, expose to the clients. Each volume can be built by several subvolumes, generally hosted by different servers. A subvolume is built by a brick, the storage file-system that has been assigned to the volume, processed by at least one translator. A translator connects to one or more subvolumes, does something with them, and offers a subvolume connection. [1].

2.1. Working with GlusterFS

With these basic concepts one can build 3 types of complex volumes: distributed, replicated and striped. The most basic volume is a distribute only volume, that simply spread the data across the available bricks, so that over 100 files written on a volume built by two bricks, an average fifty will end up on one brick, and fifty on the other. If the bricks are hosted on two different servers, we have something similar to RAID0 for physical disks, with all the pros (increased velocity) and cons (increased fragility of the volume). With the replicated volume glusterFS can trasparently replicate the data with the multiplicity choosen at the volume creation, when it is possible to set the number of file replicas that the volume must contain. Obviously this setup is particularly useful if the bricks are located on different servers.

It is also possible to mix the basic volume types, so for example one can build a distributed-replicated volume, that distributes the data across multiple servers and replicates them in order to obtain an increased availability.

We have used this type of volume, and set the replica 2 directive, so each file must be written twice. We have built a distributed replicated volume from 20 nodes: each node hosted 6 bricks, coupled with the 6 bricks on another node.

3. Description and architecture of HDFS

Apache Hadoop is an open-source software framework developed in Java that allows distributed processing of large data sets across clusters of computers using simple programming models. It is composed of several modules such as Hadoop Yarn and Hadoop MapReduce for cluster resource management and parallel processing, Hadoop Distributed File System (HDFS) that provides high-throughput access to application data and other related sub-projects such as Cassandra, HBase, Zookeeper, etc.[2]

Several big companies use Hadoop for their services; Yahoo! has by far the most number of nodes in its massive Hadoop clusters at over 42,000 nodes as of July 2011, while Facebook stores more than 100 PetaByte of data on HDFS. Many others famous companies use Hadoop such as Amazon, E-Bay, Linkedin, etc.[3]

3.1. HDFS Architecture

HDFS has a master/slave architecture. The NameNode is the master server that manages file-system namespace and regulates access to files by clients. It can be replicated in high-availability in Active/StandBy configuration sharing metadata via NFS to enable automatic or manual failover. In order to scale the name service horizontally, is possible to split namespace into multiple federations with independent namenodes and namespaces. They don't require coordination with each other but use the same datanodes as common storage.

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks on datanodes. The size of the blocks is configurable by dfs.blocksize parameter. Every block is replicated as many times as specified by replication factor parameter (dfs.replication) according to a replica placement policy managed by active namenode. To realize data reliability, namenode needs to know network topology of the cluster, and so the node-rack relationship, to place file blocks on datanodes according to replica policy that by default writes the first replica on a node of the local rack, and second and third replica on a different nodes of a remote rack, considering three as replication factor. The file-system resists the failure of a whole rack. In our activities, we developed two custom replica policies: One Replica Policy and Hierarchical Policy. The first one places a replica per rack in order to increase reliability (resisting the failure of two racks) and available bandwidth for read operation; the second one, instead, is able to exploit a geographically distributed infrastructure because it gives Hadoop the awareness of a hierarchical network topology organized in datacenters, racks and nodes. This data policy place first replica on a rack of a local site, and second and third on different racks of a remote site; in this way, the system resists failure of a whole datacenter. After a datanode failure, automatically Namenode schedules a re-replication of the blocks stored on that datanode. If it come back up, blocks are marked as over-replicated, so they will be deleted automatically in order to balance the number of replicas.

We have tested these and others type of failure such as: metadata failures, datanode failures during I/O operations, mis-replicated blocks. We always succeeded to fulfill the expected behaviour and, as expected, no data loss were registered.

Hadoop software includes the Fuse-DFS module that allows to mount HDFS in userspace; it supports many operations such as reads, writes, and directory operations (e.g., cp, ls, more, cat, find, less, rm, mkdir, mv, rmdir), however it isn't fully POSIX compliant to enable streaming access to file-system data.

3.2. Other developments

We developed parameterized and automatic script procedures to easily manage big a Hadoop cluster in order to provide:

- installation by getting software packages from an ad-hoc repository built for this specific use case; it contains Hadoop packages with policies developed;
- Configuration based on the type of node (namenode, datanode);
  - formatting, mounting and assigning unused disks/partitions to HDFS editing Hadoop file configurations;
- Process restart if node fails.

Since the namenode doesn't know what data contains a datanode when it is down, and doesn't store replica locations history, we developed a web-based monitoring system that tracks each data block and its replicas keeping a recent history of them; when a system administrator wants to know which datanode is the cause of a missing block, can easily see it by accessing the monitor. It shows also several detailed informations and attributes of nodes, files and blocks. The monitor is composed of a multi-thread Java application that is the core of the system, a Cassandra database for managing the big quantity of data and a PHP web interface. It uses

Java Hadoop API to query periodically namenode executing a file-system check and stores the informations obtained into the database; the web interface presents them.

4. Description and architecture of Ceph

Ceph is an open source distributed storage system, supported by Inktank Inc., designed to provide Object, Block and file storage in order to guarantee performance, reliability without single point of failure, fault tolerance by data replication and scalability to the exabyte level.

Ceph provides seamless access to objects using native language bindings or RADOS (Reliable Autonomic Distributed Object Store) gateway, a RESTful interface that's compatible with applications written for Amazon S3 and OpenStack Swift. Ceph's RADOS Block Device (RBD) provides access to block device images that are striped and replicated across the entire storage cluster. Ceph's RBD also integrates with Kernel Virtual Machines (KVMs). Several IaaS cloud platforms (i.e.: OpenStack, CloudStack) officially supports Ceph to provide a block storage solution.

The file-system provides also a POSIX interface named CephFS using an experimental native Linux driver written by Linus Torvalds and integrated into 2.6.34 kernel or using a more stable Fuse-based solution.

A Ceph Storage Cluster is composed of three types of daemon: Ceph Object Storage Daemon (OSD) that is responsible for storing objects on a local file-system and providing access to them over the network, Ceph Monitor (Mon) that maintains maps of the cluster state, and Ceph MetaData Server (MDS) stores metadata on behalf of the Ceph file-system. In order to guarantee the absence of single point of failure and scalability, a Ceph cluster can (should) be deployed with multiples nodes of them. Monitor nodes should be in odd number in order to provide a quorum. This is important as it avoid the risk of a typical problems called split-brain that could lead to data corruption or data loss problems.

Ceph stores a client's data as objects within storage pools. Using the CRUSH (Controlled Replication Under Scalable Hashing) algorithm, Ceph calculates which placement group should contain the object, and further calculates which Ceph OSD Daemon should store the placement group. The CRUSH algorithm enables the Ceph Storage Cluster to scale, rebalance, and recover dynamically. It is possible to define failure domain at the level of: disk, server and rack defining the CRUSH map that contains a list of OSDs, a list of "buckets" for aggregating the devices into physical locations, and a list of rules that tell CRUSH how it should replicate data in a Ceph cluster's pools.

CRUSH placement policies can separate object replicas across different failure domains while still maintaining the desired distribution. For example, to address the possibility of concurrent failures, it may be desirable to ensure that data replicas are on devices using different shelves, racks, power supplies, controllers, and/or physical locations.

Ceph does striping of individual files across multiple nodes to achieve higher throughput, similarly to how RAID0 stripes partitions across multiple hard drives and to avoid inconsistencies between data and metadata, missing or mismatched objects, Ceph scrubs placement groups comparing each primary object and its replicas as a fsck on the object storage layer. Two types of data scrubbling are executed: a daily light scrubbing for object attributes check and a weekly deep scrubbling for data integrity check.[6]

In our testing activities, we verified Ceph reliability by simulating some several failures such as corrupting MDS node to test metadata service, OSD to test data service and MON to test storage availability; both using POSIX and RBD access. In all cases Ceph worked always as expected.

We tested also the possibility to export Ceph storage using standard NFS protocols: it works quite well both using RBD and POSIX interface, but it was very unstable using direct kernel access because Ceph kernel driver is in an experimental phase yet.

We also conducted two sessions of access performance comparison using dd as tool of measurement between Ceph Fuse, CephFS with kernel driver and RBD: the first one using directly the three interfaces, the second one exporting them with NFS.

5. Test description and methodology

The testbed was designed to realize a writing and reading access data performance comparison of HDFS, GlusterFS and CephFS. There are many useful benchmark tools for this aim; we used two of them: IOzone which is a very popular file-system benchmark tool that uses POSIX multithreading, and dd which isn't a real benchmark tool but it converts and copies a file by streaming blocks data and return I/O throughput; generally it is used as benchmarking tool. The environment of the testbed consisted of a cluster composed of 20 worker nodes with 24 CPU cores, 80 GBytes of RAM and 6 hard disks of 2 TBytes per machine; they are connected by a 10Gbit/s wire speed SFP+ link. All tests were performed on Scientific Linux Cern v6, except for the Ceph-dev v0.70 round that was performed on Ubuntu 12.04 LTS. Each file-system was installed in cluster-mode on every node and was mounted in userspace through relative Fuse module, so each cluster node was configured as client, too. They was tested by executing IOzone simultaneously and concurrently on each nodes of the cluster, set as following:

```
# iozone -r 128k -i 0 -i 1 -i 2 -t 24 -s 10G
```

where:

- `-r 128K`: is used to specify the record size at 128 Kbytes;
- `-i 0 -i 1 -i 2`: is used to specify which tests to run: 0=write/rewrite, 1=read/re-read, 2=random-read/write;
- `-t 24 (or 36)`: is used to specify the number of concurrent threads or processes to have active during the measurement at 24 (or 36);
- `-s 10G`: is used to specify the size of the file to test at 10GBytes.[4]

Since IOzone doesn't perform correctly with HDFS, because this file-system isn't fully POSIX compliant, and Ceph Cuttlefish, because this file-system was affected by several bugs, we decided to run another test with a different tool that can perform the whole test without errors: dd. It doesn't allow to specify multiple threads for a single execution, so we simulated 24 threads by starting 24 instances of dd set as following:

```
# dd if=/dev/zero of=zerofile bs=4M count=2560 conv=fdatasync
```

for writing measurement, and

```
# dd if=zerofile of=/dev/null bs=4M count=2560
```

for reading measurement respectively, where:

- `if` is used to specify input file;
- `of` is used to specify output file;
- `bs` is used to set the quantity of bytes that reads or writes at a time at 4MBytes;
- `conv=fdatasync` tells dd to require a complete sync once, right before it exits;[5]

6. Test results and discussion

In this paragraph, file-systems settings and results of the measurements are reported. All file-systems were configured with two replicas and flat network topology.

6.1. HDFS

For the HDFS test, the cluster was composed of 20 datanodes and 1 namenode; a release of Cloudera Hadoop 4.1.1 (based on Apache Hadoop v2) patched by the USCMS group of Nebraska was installed on them. The fuse_dfs module was configured with rdbuffer set at 128 MBytes and big_writes enabled; Hadoop block size was set at 128 MBytes. This test round was performed with 24 and 36 concurrent threads.

6.2. CephFS

Three releases of Ceph were tested: Ceph Cuttlefish (0.61) deployed on 3 Monitors nodes, 1 Metadata server and 120 Object storage daemons i.e. one OSD per hard disk, so 6 OSDs by 20 nodes; Ceph Dumpling (0.67.3) deployed on 3 Monitors, 1 Metadata server and 95 OSDs, 5 OSDs by 19 nodes; and Ceph-Dev (0.70) deployed on 3 Monitors, 1 Metadata server and 15 OSDs (5 OSDs by 3 nodes).

6.3. GlusterFS

Lastly, also two releases of Gluster we tested: Gluster v3.3 deployed with 21 nodes and v3.4 deployed with 20 nodes; both with 6 bricks per node.

Here are listed iozone findings, as throughput average per single host, so as aggregate of 24 (or 36) iozone threads:

Table 1. Test results using iozone. MB/s

	HDFS 24 Threads	HDFS 36 Threads	Ceph 0.61	Ceph 0.67.3	Ceph 0.70	Gluster 3.3	Gluster 3.4
Initial Write	239.72	N.A.	52.49	18.93	51.06	234.06	306.34
Re-Write	ERROR	ERROR	54.05	19.31	60,05	311.75	406.90
Random Write	ERROR	ERROR	ERROR	13.96	7,00	326.89	406.33
Read	155.18	193.65	95.38	53.40	101,58	621.08	688.06
Re-Read	151.33	207.43	102.04	57.29	133,61	662.92	711.46
Random Read	29.06	39.98	ERROR	5.13	12,05	242.75	284.00

Here are listed dd findings as throughput average per single host, so as aggregate of 24 dd instances:

Table 2. Test results using dd. MB/s

	Ceph CF	GLUSTER 3.3	HDFS
read	126.91	427.3	220.05
write	64.71	268.57	275.27

In dd round, we compared Cuttlefish (CF) release of Ceph and 3.3 version of Gluster because they are last stable versions released at the test moment.

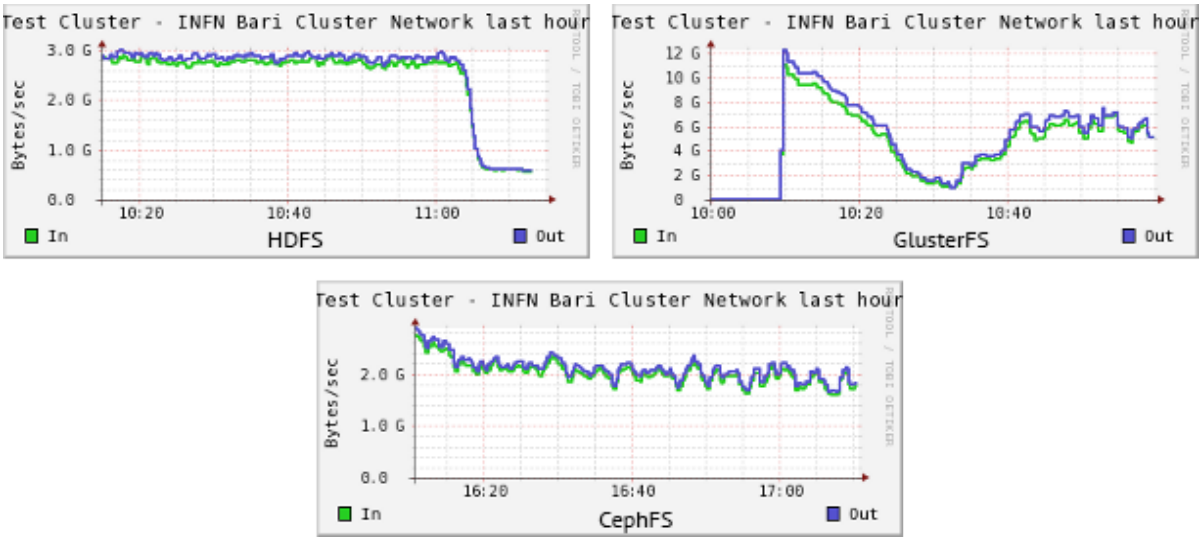

Figure 1. HDFS, GlusterFS, CephFS aggregate network performance.

By results HDFS appears faster in writing than in reading and this is due to the Fuse parameter big_writes enabled. It's pretty noticeable that Gluster performs better than others in all cases; it reached over 12 GByte/s as aggregate bandwidth throughput in reading compared to 3 GByte/s of HDFS and about 2 GBytes/s of Ceph Cuttlefish. Trying to justify large variations in the results, we can consider that: all file-systems were configured on a flat network topology, so no ones have data placement informations; they were tested on the same hardware, then we can suppose that results difference may be imputable to a simpler implementation of GlusterFS and less performance overhead then HDFS and CephFS. Infact HDFS is developed in Java that involves an additional software layer: JVM, Ceph instead is a quite young product, so it needs performance improvements by its developers.

7. Future works and conclusions

We will continue this activity of testing storage solution in order to follow the quite fast evolution in this field. In particular Ceph looks quite promising if/when stability and performance issues will be solved, but currently Gluster remains the best system in all performance test we conducted, although it presents some instabilities. In fact in two different setups, we were able to completely trash a working file-system simply adding a new volume to the pre-esistent ones and launching the "rebalance" command. HDFS instead appears the more stable and reliable storage system, performs quite well but can't support cloud technologies. The increasing interest in cloud storage solution are forcing the developers to put effort in providing both block and object storage solutions together with the standard POSIX.

References

[1] http://gluster.org/community/documentation/index.php/GlusterFS_Concepts, August 2013
[2] http://hadoop.apache.org
[3] http://www.hadoopwizard.com/whichbigdatacompanyhastheworldsbiggesthadoopcluster, February 2013
[4] http://linux.die.net/man/1/iozone
[5] http://linux.die.net/man/1/dd
[6] http://ceph.com