Multicluster Hadoop Distributed File System

I. Tomašić, J. Ugovšek, A. Rashkovska and R. Trobec
Jožef Stefan Institute/Department of Communication Systems, Ljubljana, Slovenia
Ivan.Tomasic@ijs.si, janez@ugovsek.info, Aleksandra.Rashkovska@ijs.si, Roman.Trobec@ijs.si

Abstract - The Hadoop Distributed File System (HDFS) is one of the important subprojects of the Apache Hadoop project that allows the distributed processing and fast access to large data sets on distributed storage platforms. The HDFS is normally installed on a cluster of computers. When the cluster becomes undersized, one commonly used possibility is to scale the cluster by adding new computers and storage devices. Another possibility, not exploited so far, is to resort for resources on another computer cluster. In this paper we present a multicluster HDFS installation extended across two clusters, with different operating systems, connected over the Internet. The specific networking parameters and HDFS configuration parameters, needed for a multicluster installation, are presented. We have benchmarked a single and dual cluster installation with the same networking and configuration parameters. The benchmark results indicate that multicluster HDFS provide increased storage area, however, the data manipulation speed is limited by the bandwidth of communication channel that connects both clusters.

I. Introduction

The big data applications on distributed storage systems gain importance in different fields of applications [1-3]. The Apache Hadoop [4] is an open source software project sponsored by the Apache Software Foundation (http://www.apache.org). The Apache Hadoop software library allows the distributed processing of large data sets across clusters of computers using a simple programming model. It is designed to scale up from a single server to thousands of machines, each offering local computation and storage [5]. Rather than rely just on hardware to deliver high-availability, the HDFS library is designed to detect and handle failures at the application layer. As a result, the HDFS is a highly-available service that can be installed on commercially available clusters of computers, which have not been built with any special care for minimizing the failure rate.

Hadoop encompasses multiple subprojects:

- Hadoop Common: The common utilities that support the other Hadoop subprojects,
- Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data, and
- Hadoop MapReduce: A software framework for distributed processing of large data sets on compute clusters.

The HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware. There are Hadoop clusters running today that store petabytes of data [6].

Since the Hadoop is designed to run on commodity hardware, when a Hadoop cluster becomes undersized, one commonly used possibility is to scale the cluster by adding new computers and storage devices. Another possibility, not exploited so far, is to resort for resources on another computer cluster. The latter paradigm is sometimes called "scaling out".

The Hadoop cluster nodes are organized in a master-worker pattern. There is one namenode that has the master role and there is a number of datanodes, which are the slaves. The namenode manages the file system namespace. It maintains the file system tree and the metadata for all the files and directories in the tree. All this information is stored on a disk in the form of two files: the namespace image and the edit log. The namenode is configured to know the addresses of the datanodes that store the blocks for a given file.

A Hadoop client accesses the HDFS file system by communicating with the namenode and datanodes. The user code does not need to know about the namenode and datanode functionality because of the POSIX-like [7] file system interface between the Hadoop client and the system.

The datanodes store or retrieve blocks when a request comes from the Headoop client or the namenode. The datanodes also report periodically to the namenode and send it lists of blocks that they are storing. Without the namenode, the file system cannot be used. In a case of an error on the machine running the namenode, all the files on the file system could be lost because there is no alternative way to reconstruct the files from the blocks on the datanodes. For this reason, it is important to make the namenode resilient to failures.

To evaluate the feasibility and effectiveness of HDFS, we have made a test HDFS installation. The installation consists of a single namenode and five datanodes inside a cluster residing at Jožef Stefan Institute (JSI), and one additional datanode on a cluster accessible over the Internet and located at Turboinštitut d.d. (TI). Hence, the installed system spreads over two clusters and is therefore called multicluster HDFS installation. It is to be distinguished from the notion of a hybrid cloud, which represents a collaboration of a private and a public cloud.

In the rest of the paper the architecture of the HDFS multicluster is described together with the description of HDFS configuration parameters. For the purpose of

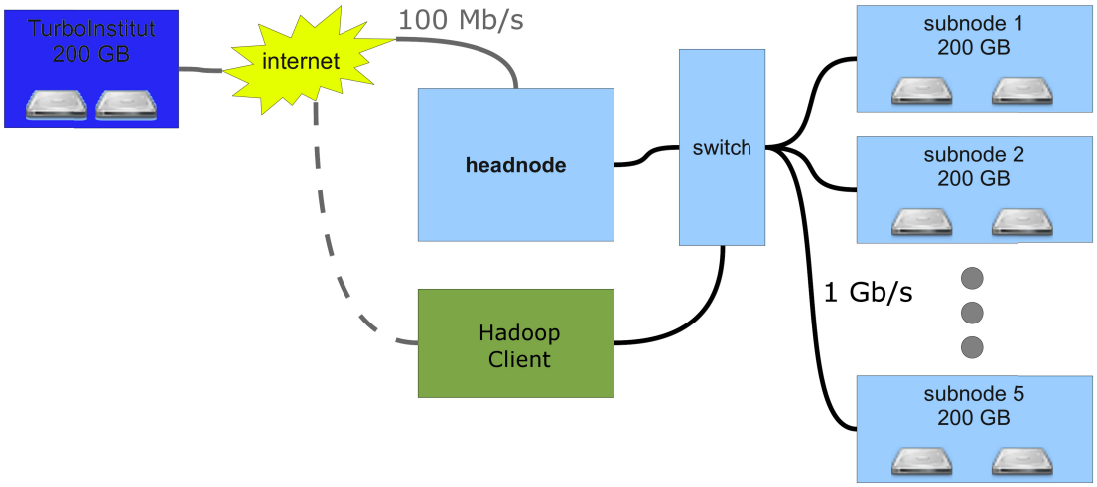

Figure 1. Multicluster storage system arhitecture

testing and analyzing the HDFS performances, we have constructed a set of benchmark programs, described in Section III. The benchmark results are presented and commented in Section IV. Our paper concludes with a summary of achieved results and some directions for the future work.

II. METHODS

A. System arhitecture

The experimental HDFS installation has the following configuration:

- Master node, the multicluster's headnode, which is the headnode of the JSI cluster, is running:
  - namenode
  - jobtracker,
- each of subnodes 1 till 5 at the JSI cluster and the subnode at the TI cluster are running:
  - datanode
  - tasktrecker.

The namenode and the jobtracker are running on the same machine, as it is usually acceptable for small clusters, i.e., on clusters with up to 10 nodes [8].

Each node on the JSI cluster has a single 500 GB data disk with the rotation speed of 7200 rpm. Only the head node is equipped with the RAID. The local network in the cluster is 1 Gb/s, while the external internet connection bandwidth is 100 Mb/s. Each JSI cluster node has 6 GB of RAM and Intel(R) Xeon(R) E5520 @ 2.27 GHz processor with four cores.

The TI cluster node has two Intel(R) Xeon(R) L5520 @ 2.27GHz processors, each with four cores, and 24 GB of RAM. The TI cluster node has a single 260 GB data disk.

On each of the available disks a partition of 200 GB is allocated for the utilization by the HDFS. The partitions are formatted with the ext4 file system.

This work was done under the competence center CLoud Assisted ServiceS (KC CLASS), founded by the Slovenian Ministry of Education Science and Technology, and the European Fund for Regional Development.

The JSI cluster nodes run under Ubuntu Server operating system, while the TI cluster node runs under SUSE Linux operating system. The described multicluster system architecture is shown in Fig. 1.

B. HDFS Configuration

Since Hadoop scripts rely on SSH to perform cluster-wide operations, SSH needs to be setup to allow password-less login for the Hadoop user from all machines in the multicluster system. This is achieved by generating one RSA [9] public/private key pair shared across the multicluster.

The software configuration of Hadoop cluster consists of an environment configuration for the execution of Hadoop daemons, and their parameters configuration. The Hadoop configuration parameters are stored in various configuration files located in the *conf* directory of the Hadoop distribution. Each Hadoop node in the cluster has its own set of configuration files. It is up to the administrator to ensure that they are kept synchronized across the whole system. The Hadoop provides a rudimentary facility for synchronizing configuration by using rsync [10].

The list of all datanodes is in the *slaves* configuration file containing IP addresses (on the local network) of the machines where the datanodes and tasktrackers should run. In our multicluster architecture the list also contains the IP address of the TI node, which can be accessed over the Virtual Private Network (VPN). It is obligatory to use the VPN address of machines in another cluster, since these are the only addresses visible to the namenode machine.

The parameter JAVA_HOME located in *hadoop-env.sh* configuration file specifies the location of the Java implementation to be used. Setting the value in hadoop-env.sh on all the machines in the multicluster is preferable because it ensures that the whole cluster is using the same version of Java.

The parameter fs.default.name, located in the *core-site.xml* configuration file, is normally (in plain cluster installation) set to the IP address or the name of the

namenode machine. It is an HDFS file system URI that defines the hostname and the port that the namenode's RPC server runs on. The default port is 8020. However, in the multicluster installation, the machine (or machines) located outside the namenode's cluster do not "see" the namenodes's IP on the local network. Therefore, it is necessary to use the namenodes VPN address, which is seen by all machines in the multicluster. Note that when a property is marked as final it is protected from being overridden by job configurations.

The parameters dfs.name.dir in the *hdfs-site.xml* configuration file specifies a list of directories where the namenode stores persistent files system metadata, while dfs.data.dir parameter in the same file specifies a list of directories where the datanode stores its blocks. In the multicluster installation, the nodes in the main cluster may need different dfs.data.dir parameter value than the nodes on the remote cluster. Because the remote cluster may be using different operating system and may be controlled by different administrator, the directory configured for the main cluster does not have to exist on the remote cluster. Therefore, in our case, two versions of the *hdfs-site.xml* configuration file are maintained: one for the nodes on the main JSI cluster and other for the node on the remote TI cluster. All the other parameters are left default. Note that the default replication factor is three.

The same holds for parameters mapred.local.dir and mapred.system.dir from the *mapred-site.xml* configuration file, which specify the location of local temporary storage for intermediate data and working files that are produced during a MapReduce job [11], and a directory where the shared files for the MapReduce tasks files can be stored. Hence, for our multicluster, two versions of the *mapred-site.xml* configuration file are maintained.

III. PROGRAMS FOR TESTING

We have developed scripts for benchmarking of the Hadoop deployment.

```
#!/bin/bash
if [[ "$1" == "rm" ]]; then
    /usr/local/hadoop-0.20.203.0/bin/hadoop dfs -fs
    hdfs://ninestein/ -rm /user/guest/$2
elif [[ "$1" == "ls" ]]; then
    /usr/local/hadoop-0.20.203.0/bin/hadoop dfs -fs
    hdfs://ninestein/ -ls
elif [[ "$1" == "up" ]]; then
    /usr/local/hadoop-0.20.203.0/bin/hadoop dfs -fs
    hdfs://ninestein/ -copyFromLocal $2  /user/guest
elif [[ "$1" == "down" ]]; then
    rm $2 &>>/dev/null
    /usr/local/hadoop-0.20.203.0/bin/hadoop dfs -fs
    hdfs://ninestein/ -copyToLocal /user/guest/$2 .
fi
```

File 1. hadoop.sh

The file *hadoop.sh* (File 1.) is a script that does the file operations. The first parameter of the script is therequested file operation: deletion, listing, copying to HDFS, or copying from HDFS. The second parameter is the file name.

The script *hadoop_testing.sh* does the actual benchmarking (File 2). It is accessing files of different sizes, copying them to or from the HDFS, and measuring the time needed for the operation. It uses the script *hadoop.sh* for the actual operations with files. Each copy operation is repeated 5 times to obtain an average operation time.

```
#!/bin/bash
# writing
for i in 1 10 100 1000 10000 100000 1000000
2000000 3000000; do
    #fill the cache
    dd if=benchmark/$i.file of=/dev/null
    &>>/dev/null
    echo -n "$i, "
    for ((j=0; j<5; j++));
    do
        #writing time
        time=$(/usr/bin/time -f %e ./hadoop.sh up
        benchmark/$i.file 2>&1)
        echo -n "`echo $time | awk '{print $1}'`, "
        if [[ "$j" != "4" ]]; then
            ./hadoop.sh rm $i.file &>>/dev/null
        fi
    done
done
# reading
for i in 1 10 100 1000 10000 100000 1000000
2000000 3000000; do
    echo -n "$i, "
    for ((j=0; j<5; j++));
    do
        #reading time
        time=$(/usr/bin/time -f %e ./hadoop.sh down
        $i.file 2>&1)
        echo -n "`echo $time | awk '{print $1}'`, "
        rm $i.file
    done
done
exit 0
```

File 2. hadoop_testing.sh

IV. BENCHMARK RESULTS

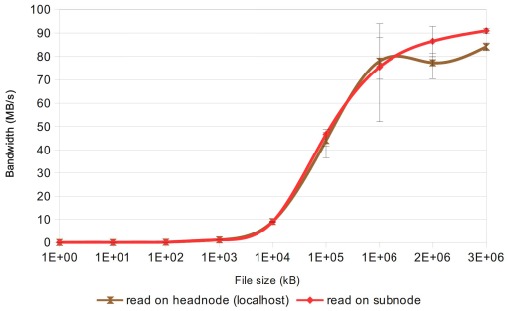

Figure 2. Read bandwidth as a function of file size (JSI cluster)

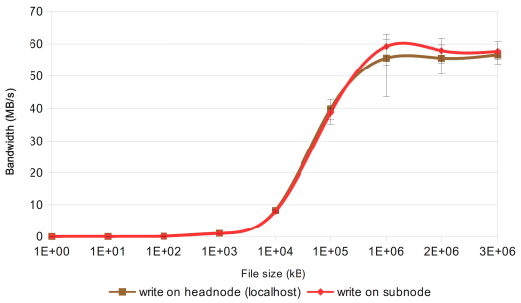

Figure 3. Write bandwidth as a function of file size (JSI cluster)

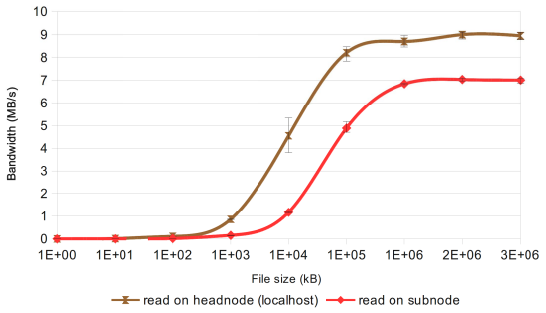

Figure 4. Read bandwidth as a function of file size (multicluster)

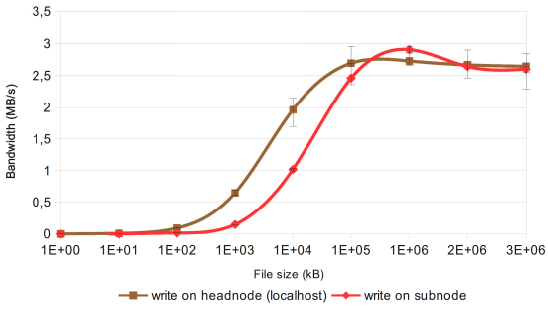

Figure 5. Write bandwidth as a function of file size (multicluster)

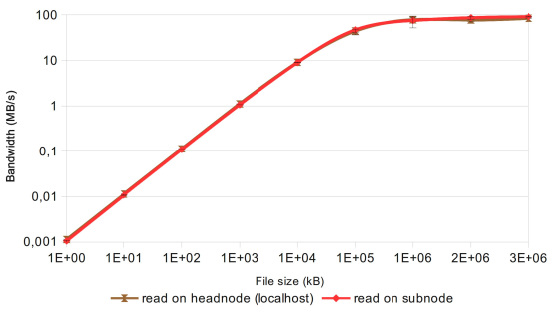

Figure 6. Read bandwidth as a function of file size in logarithmic scale (JSI cluster)

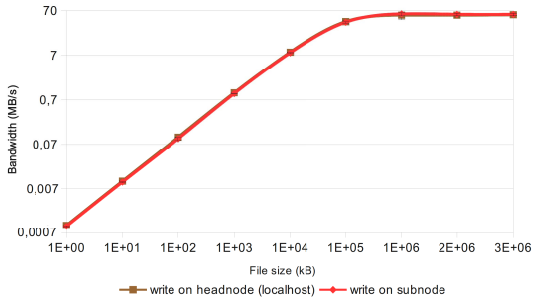

Figure 7. Write bandwidth as a function of file size in logarithmic scale (JSI cluster)

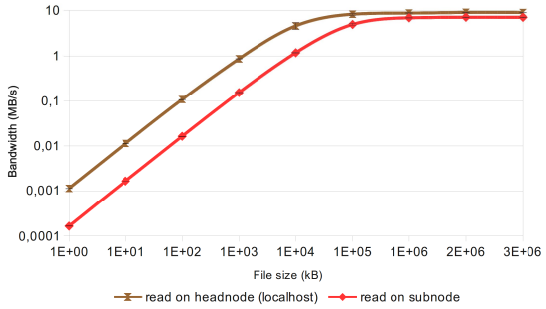

Figure 8. Read bandwidth as a function of file size in logarithmic scale (multicluster)

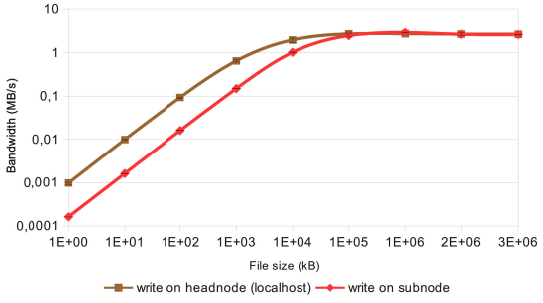

Figure 9. Writing bandwidth as a function of file size in logarithmic scale (multicluster)

The read and write operations have been conducted from the headnode, located on the JSI cluster, and from a JSI cluster's subnode.

The read and write bandwidths were tested for file sizes form 1 kB up to $3 \cdot 10^6$ kB. Any experiment has been repeated five times for each file size. The whiskers on the graphs (Figures 2 - 9) represent value ranges (max-min) for the five cases, while the mark represents the average value of each set of experiments.

Fig. 2 and 3 show the read and write performances obtained on the JSI cluster only (the TI node was not a part of the test system). Fig. 4 and Fig. 5 depict the read and write performances obtained on the interconnected JSI and TI clusters (multicluster). Figures 6 – 9 show the same results but in the logarithmic scale on the y axis to provide better inside into the performances for smaller files.

We see from Fig. 2 and Fig. 3, obtained from the JSI cluster, that the read and write bandwidths are almost the same regardless of the node from which the operations are conducted. The maximal write bandwidth is approximately 35 % lower than the maximal read bandwidth.

In the multicluster case (Fig. 4 and Fig. 5), the read bandwidth is significantly higher for the headnode. This behavior can be expected because, if the read operations are performed form a subnode, there is a possibility for a delay due to the communication with the headnode. The maximal write bandwidth is almost equal for the headnode and the subnode. However, for file sizes of 10^6 kB the write bandwidth is higher if write operations are performed from the subnode. We could explain such a behavior with eventually available storage data buffers.

The logarithmic scale of y-axis in Fig. 6 to Fig. 9. discovers that the bandwidths grow linearly up to file sizes of 10^5 kB. We can observe that the manipulation with the smaller files is constantly 10 times slower on the multicluster system. For files larger than 10^5 kB the bandwidths reach their maximal values and remain quite stable. Again, there is approximately tenfold difference in the bandwidth in favor of the system with IJS cluster only. This can be expected because the data from the TI cluster are accessed through the 100 Mb/s Internet connection, which is approximately ten times slower than the JSI cluster network (see Fig 1.).

V. CONLUSION

We have presented the steps and configurations needed for deploying the Hadoop on a multicluster. We further tested the installation with a series of benchmarks, developed for this purpose. The multicluster paradigm enables a HDFS cluster to be scaled out to physical machines that are accessible over the internet. Such an approach may decrease the investments if an existing storage system becomes occasionally undersized.

We have performed a series of benchmarks on the JSI Hadoop cluster and on the multicluster constituted by joining a storage node from the TI cluster. Benchmark results indicate that the read and write bandwidths drop significantly with joining the TI node. One obvious reason for such a behavior is slow communication line between the clusters, but this drawback can be easily improved by introducing a faster interconnection channel. Further investigation is needed to identify the Hadoop implementtation properties that influence the fall of performance. We suspect that a smarter organization of data items on a higher abstraction level, e.g., user location aware approach, could, in most cases, alleviate this problem.

However, the drop in read and write bandwidths does not necessary imply the same drop in performances of MapReduce jobs, because Hadoop does its best to run the Map tasks on nodes where input data reside in the HDFS, because of build-in "data locality optimization" principle. The Reduce tasks, usually, don't possess the advantage of data locality as the input to a single reduce task is most often the output from all mappers. Further tests with MapReduce jobs are therefore needed to investigate the influence of the multicluster architecture on the Hadoop MapReduce performances.

The multicluster storage, presented in this paper, was built from computing nodes of two clusters on different geographical positions, with different hardware platforms and different operating systems, which confirms that a multicluster Hadoop on heterogeneous clusters is a viable option. The read and write performances of HDFS, if critical for a multicluster application, have to be improved, e.g., by increasing the communication speed of the inter-cluster connection.

ACKNOWLEDGMENT

The authors are grateful to Turboinštitut d.d. for granting a full access to their storage and computational resources at Ljubljana Supercomputing Center - LSC ADRIA.

REFERENCES

[1] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox, "Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 17, pp. 2338–2354, 2011.

[2] Yunhong Gu and Robert L. Grossman, "Sector and sphere: The design and of a high-performance data cloud," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1897, pp. 2429-2445 , 2009.

[3] E. Afgan et al., "Harnessing cloud computing with Galaxy Cloud," *Nature Biotechnology*, vol. 29, no. 11, pp. 972-974, 2011.

[4] Welcome to Apache Hadoop. [Online]. http://hadoop.apache.org/

[5] S. Yu, X. Gui, R. Huang, and W. Zhuang, "Improving the storage efficiency of small files in cloud storage," *Hsi-An Chiao Tung Ta Hsueh/Journal of Xi'an Jiaotong University*, vol. 45, no. 6, 2011.

[6] Yahoo. [Online]. (2008, September) http://developer.yahoo.com/blogs/hadoop/posts/2008/09/scaling_hadoop_to_4000_nodes_a/

[7] IEEE Standard Association. [Online]. http://standards.ieee.org/findstds/standard/1003.1-2008.html

[8] Tom White, "The Hadoop Distributed Filesystem," in *Hadoop: The Definitive Guide*. Sebastopol,: O'Reilly Media, Inc., 2011, ch. 3.

[9] Wikipedia. [Online]. http://en.wikipedia.org/wiki/RSA_%28algorithm%29

[10] Wikipedia. [Online] http://en.wikipedia.org/wiki/Rsync

[11] S.N. Srirama, P. Jakovits, and E. Vainikko, "Adapting scientific computing problems to clouds using MapReduce," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 184-192, 2012.