New Data Placement Strategy in the HADOOP Framework

Akram ELomari[1], Larbi Hassouni[2], Abderrahim MAIZATE[3]
RITM-ESTC / CED-ENSEM, University Hassan II
Casablanca, Morocco

***Abstract*—Today, the data quantities generated and exchanged between information systems continues to increase. Storing and exploiting such quantities require can't be done without bigdata systems with mechanisms capable of meeting technological challenges commonly grouped under the four Vs (Volume, Velocity, Variety and Veracity). These technologies include mainly the Distributed File System (DFS). Like Hadoop, which is based on HDFS, the main Big Data systems use a data distributed storage where a subsystem is responsible for subdividing data (data striping) and replicating it on a network of nodes called Grid. In the typical case of Hadoop, a Grid generally consists of many nodes, grouped in multiple Racks. The logic of distributing the stored data through the Grid respects a simple strategy that guarantees the durability of the data and a certain speed of writing. This strategy does not take into consideration neither the technical characteristics of nodes, nor the number of requests on the data, which means a considerable loss in processing capacity of the grid. In this work we proposed a new placement strategy based on exploitation analysis of new information integrated into the HDFS metadata model. A significant 20% improvement in overall processing time was reached through the simulations we conducted on Hadoop.**

***Keywords*—Big data; data storage; Hadoop; DFS; HDFS; data striping; chunks; placement strategy; performance optimization**

I. Introduction

The amount of data generated and processed by current information systems continues to increase, and the generation of digital information has never been more abundant. While the volume of data created was estimated at 2 ZB in 2010, it is estimated that it has reached 47 ZB in 2020 and will increase to 175 ZB in 2025 and 2142 ZB in 2035 [1].

Also, taking advantage of data mines has become a complex and demanding operation. As a result, Big Data systems are positioned as effective [2], scalable, and efficient solutions to exploit this quantity of Data.

The purpose of big Data systems is therefore to store and analyze very large masses of data, while ensuring an adequate level of data security and accessibility.

In this context, most Big Data systems, such as the Apache Hadoop [3], delegate data storage management to distributed file systems (DFS).

The Hadoop Big Data system is based on HDFS (the standard DFS of HADOOP). In combination with other layers of data processing, Hadoop can achieve complex calculations on extended clusters. The main objective of HADOOP is to distribute complex operations on multiple machines while bringing those operations closer to the concerned data and thus improve the global performances.

The capacity and performance of similar platforms, such as Facebook and Google also depend on this distributed architecture based on DFS [4].

HDFS thus manage several storage nodes dedicated only to storage or to storage and computing at the same time, grouped generally in racks and interconnected by local or wide networks.

HDFS machines do not have specific characteristics; they are usually low-cost machines, easily replaceable. The protection against breakdowns is provided by a data placement strategy based on striping and replication of data blocks called chunks. This strategy is based on simple principles [5]:

Create multiple copies(replicas) of the same data.

Place replicas on several machines distributed on several racks (as much as possible).

The placement of replicas, however, does not take in consideration the nature of the demand on each data or its evolution over time [6]. The performance of machines is not considered too, even if HDFS allows integrating machines of different characteristics on the same cluster.

These shortcomings in HDFS' data placement strategy have prompted us to propose an improvement in this strategy as a novel algorithm, which takes into consideration the history of reading operations and proposes a redistribution of data over the cluster.

Our research starts with a deep understanding of the working mechanisms of HADOOP as a whole and of HDFS, to design an algorithm that is applicable and implementable.

Our test simulations were conducted on small cluster to control data transfers and thus demonstrate the efficiency of the algorithm even on a small scale.

The rest of the paper is as follows. Section 2 provides an overview of HADOOP Architecture. Section 3 presents the new proposed placements strategy of chunks. Section 4 presents testing and simulation approach. Section 5 the results and discussion and section 6 provide a conclusion of the work.

II. Hadoop Architecture Overview

Our research method was based on an in-depth analysis of how Hadoop's HDFS works, in order to better understand the

problematic raised by the chunk placement strategy. The aim of this analysis is to bring a considerable improvement to this strategy, by using new algorithms which exploit new metadata integrated into the architectural model of HDFS.

Our analysis begins with an understanding of HDFS and its logic of data stripping and replication, followed by a critical analysis of the strategy currently used by HDFS to distribute replicas across the cluster. We then proposed new metrics to be calculated and integrated with chunk metadata managed by HDFS, as well as an algorithm that is based on these metrics and offers a new and more efficient way to place chunks.

Hadoop is Apache's known big data system, based on an open-source implementation of Google's famous MapReduce. It is a complex ecosystem that aims to optimize storage and calculations made on data in large quantities.

The main purpose of Hadoop is to distribute very large data processing on clusters composed mainly of entry-level machines. The goal is to bring processing closer to data instead of moving data to processing, to minimize network bandwidth occupation. Hadoop's strength comes from the fact that it can manage platforms ranging from a few servers to clusters of thousands of machines, while accepting a high tolerance for hardware failures given the wide variety of machines supported.

By offering a software-managed service continuity model regardless of hardware quality, Hadoop can detect hardware failures at the application level and take the necessary steps to return to normal in case of failure in a transparent manner for system users. Hadoop is a flexible, scalable, and highly tolerant solution to hardware failures which guarantees a very good cost/effective rate.

A. *Hadoop Architecture*

There are three major implementations of Hadoop, the 1.x launched for the first time in 2012, the 2.x first appeared in 2013 and the 3.x first launched in December 2017.

Hadoop's overall architecture has not changed much since its first release; it's based on a Model of Master/Slaves/Users.

Two layers are still present:

- The storage layer of data represented by HDFS (Hadoop Distributed File System) which manages the distribution of data through the cluster and always ensures the integrity and persistence of this Data. This layer also manages storage servers (DataNode).
- The data processing layer that manages the parallelization of calculations across the cluster, based on the MapReduce model.

Therefore, two types of Master exist on Hadoop: the NameNode in charge of the HDFS component and the JobTracker (replaced in version 2.x by YARN (Yet Another Resource Negotiator)) which takes care of the implementation of MapReduce.

The NameNode is duplicated for high availability purpose (even tripled on version 3.0).

The rest of the cluster consists of nodes running the DataNode process for storage distribution (communicates exclusively with HDFS NameNode) and TaskTraker for data processing (communicates exclusively with YARN JobTracker or resource manager)

YARN uses the ResourceManager which manages several NodeManager responsible for distributing tasks and reporting the status of nodes to the ResourceManager.

The client communicates with both masters and Slaves to write or read data or give instructions on how this data should be processed.

Fig. 1 gives a basic representation of Hadoop's organization.

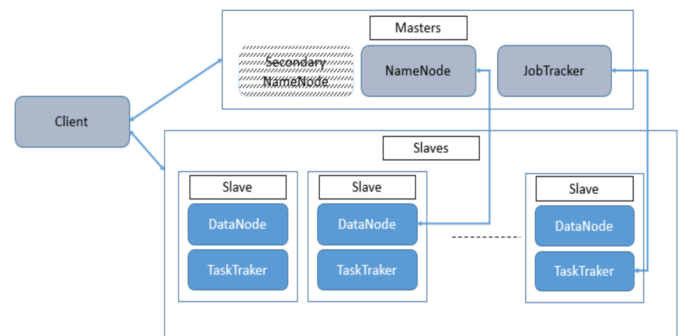

Fig. 1. Hadoop Architecture.

While reading process, the client wishing to read data contacts the NameNode to get the locations of the data blocks, and then reads the contents of the block from the nearest DataNode. When writing, the client asks NameNode to name a suite of three (by default three but can be configured) DataNodes to host replicas of the blocks. The user then writes the data in the DataNodes as a pipeline, user to node1, then node 1 to node 2 and node 2 to node 3.

The current design has only one active NameNode for each cluster. The cluster can have thousands of DataNodes and tens of thousands of HDFS clients per cluster, because each DataNode can perform multiple application tasks simultaneously.

The data processing goes through the JobTracker, the client constitutes his MapReduce job and submits it to the JobTracker which subdivides it into multiple Map/Reduce tasks and assigns each one to a TaskTracker of a DataNode containing concerned data. The user can retrieve the results through direct readings on HDFS.

B. *Hadoop Distributed File System*

HDFS is the distributed file storage system used by default by Hadoop. It has the particularity of being able to run on machines of low cost and consequently to be very tolerant to hardware failures.

For its implementation, several hypotheses have been made:

- The predisposition of equipment used to failures requires a high capacity to detect these failures and restore service.
- HDFS is better prepared for batch treatments and not for user-by-user interactions. POSIX semantics have been particularly relaxed in order to facilitate streaming and increase data flow.
- HDFS is geared to handle very large data. Files stored on HDSF can reach terabytes and therefore the size of the cluster and its architecture must support these sizes.
- Files on HDFS are usually written only once by a single owner but they are read several times. Actions that may change the files are Append or Truncate.
- HDFS allows applications to move data near processing. The other sense (bring data into processing) can be very bandwidth consuming especially when the data is very large.
- HDFS has been developed with JAVA which allows it to turn on many different platforms.

These assumptions frame the architecture of HDFS and allow the optimization of several aspects [7], especially data security and processing speed.

HDFS expose a Filesystem, called Namespace, similar to most traditional Filesystems; it can contain folder hierarchies, user can create, rename, move or delete files from the system. It can also include access rights or quotas per user.

1) Architecture of HDFS: The architecture of HDFS is based on a single active Master (called NameNode) and several Slaves (called DataNode).

The NameNode manages the state of DataNodes in the cluster and handle distribution of storage throughout the cluster.

The NameNode also regulates user's access to data, every write or read operation starts by requesting information from the NameNode [8].

However, the NameNode is not involved in the transfer of data between the user and dataNodes, since data blocks information (Metadata) are separated from the data itself and stored on the NameNode, it simply transfers a list of DataNodes concerned by the user's query as well as the metadata of chunks.

The user then communicates with the DataNode without overloading the NameNode. This greatly simplifies the architecture of HDFS and prevents the NameNode from becoming a bottleneck.

The persistence of the NameNode is guaranteed by two main files:

- A transaction log called EditLog that records all changes made to HDFS metadata (e.g., replica number change).
- A data file called FsImage, which contains all the information about the system as the locations of the blocks and the properties of the system.

FsImage reflects the exact state of HDFS; however it is not updated directly after data modification operations. Operations are recorded in real time on EditLog and flashed in bulk on the FsImage at the starting of HDFS or at specific times called checkpoints.

These two files constitute the heart of HDFS, and their corruption can cause the total shutdown of the service. So, the NameNode constitute a real SPOF (Single point of Failure) of the system.

To reduce the risk associated with this SPOF, the NameNode can maintain several copies of these two files and keep them meticulously synchronized, or else set up several NameNodes (possibility to put 2 on version 2 of Hadoop or more on version 3), where one is active, and the others are passive but ready to provide a fast "Failover" if needed. In this case the DataNodes are all conFig.d to recognize all NameNodes and transmit them the same data transmitted to the Active NameNode.

This can be combined with the use of a distributed EditLog called Journal which consists of the definition of several machines on the cluster (JournalNodes or JNs) that will house copies of the EditLog. The Active NameNode is responsible for transmitting the Editlog changes to the majority of the JNs. Other passive NameNodes see these changes and applicate them onto their own Namespace. In a disaster recovery situation, a passive NameNode makes sure to execute any changes made to the EditLog before turning into the active NameNode.

2) Replication in HDFS: Because HDFS often runs on convenience equipment, hardware failures can occur at any time, which can result in partial or total data loss [9]. To mitigate this risk, HDFS uses a replication technique that aims to ensure the sustainability of the data in case of hardware or network technical problems.

Fig. 2 explains the overall pattern of the HDFS replication model.

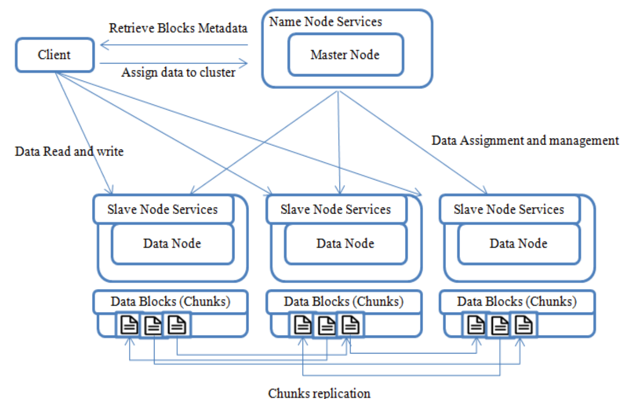

Fig. 2. Replication Process in HDFS.

In HDFS, the files are split into several Blocks of equal sizes (except the last one) called chunks. This manipulation, called Data Stripping, allows HDFS to parallelize data access by storing these blocks on different DataNodes and thus improve response times. HDFS then replicates each Block in several copies (usually three copies but it is configurable) and places each copy on a DataNode determined according to a predefined strategy.

The number of replicas, also called replication factor, is managed by the NameNode and can be changed at any time and for each file separately.

To keep the NameNode up to date, it receives regularly information from each DataNode. Two types of information are received:

- The HeartBeat that allows the NameNode to ensure that the DataNode is still up and running.
- The BlockReport through which DataNode provides NameNode with a list of all valid data blocks it contains.

When creating a file, the NameNode provides the user with a list of DataNodes that can host that data (the list contains N DataNode where N is the replication factor). The user starts transmitting the data in pieces to the first DataNode which writes it locally and begins to transmit it to the second DataNode in the list and so on until the Nth DataNode.

3) The placement of replicas on HDFS: To designate the DataNode that will house the Blocks during a write operation, HDFS applies a policy that considers both the risk of failure and the speed of access.

HDFS is often set up as a cluster consisting of several racks where several DataNodes are housed. Communication between machines of different racks necessarily passes through switches which make the exchange between machines of the same rack generally faster than the exchange between machines of different racks [10].

It will therefore make sense to select DataNode from the same rack to place all the replicas of a block, given the saving of time and bandwidth that this can provide. However, in this way it will increase the fragility of the system in case of a failure affecting an entire rack.

That is why HDFS has a strategy that takes these two aspects in consideration:

As illustrated in Fig. 3, in the common case where the replication factor is 3, two of the replicas are placed on two DataNodes of the same rack while the third is placed on a DataNode of a different rack, depending on the availability of storage space.

If the replication factor is greater than three then these conditions are supplemented by the condition of not putting more than two replicas on the same Rack or the same Data Node as far as possible.

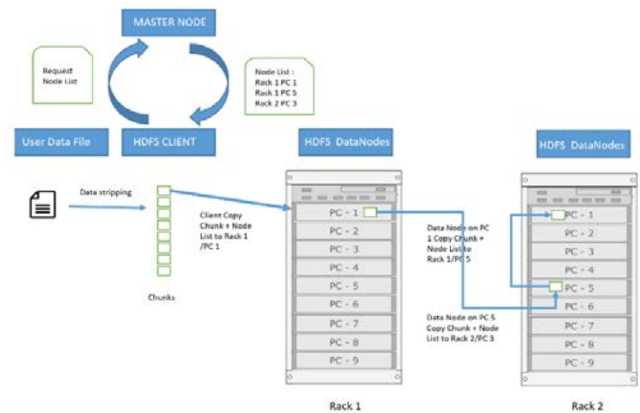

Fig. 3. Chunk's Placement Strategy in HDFS.

The problem statements:

This strategy allows better data availability in case of a network, rack or switch failure, and also optimizes writing times since one-third of the data is exchanged in a single rack.

However, this strategy may reduce the overall parallel playback bandwidth since the data is available only on two racks and not three (where 3 is the replication factor).

This strategy also shows another more important disadvantage [11]; it determines the DataNode where replicas are placed at creation of the chunk and never takes in consideration the evolution of traffic or demand on the replicas, nor the capacity of the nodes chosen for the hosting of the data.

Indeed, at the time of the creation of the chunks, the first replica is created on a DataNode close to the "writer", but our simulations have shown that the demand on the replicas can evolve to another location. In addition, since Hadoop cluster is built using convenience hardware that can fail or simply become obsolete [12], the structure and composition of the cluster may evolve rapidly and radically.

This suggests moving most requested Data (according to a cluster operations analysis) to machines most willing to process them as quickly as possible [13].

These points make the placement strategy used by default by HDFS, a possible source of loss of processing performance due to the poor distribution of data on the cluster.

Our work consists of proposing a method for analyzing the demand on replicas and develops an algorithm for replacing replicas based on the recurrence of demand and machines performances [14], [15].

4) Related work: Many works have tried to improve the strategy of placing chunks in DFS. Some works have proposed to change the storage method by reducing the number of small files by grouping them into larger files [5], or to optimize the replication factor and the size of chunks to improve internal exchanges inside the cluster [6], [7]. Another approach is to try to assign the data to the best node according to the known physical capacities of each node. The limitation of these methods is that it is based on data inputs that never updates,

even in evolving environments like HADOOP where the hardware is constantly changing because of its possibly low-cost nature.

Our approach allows building a dynamic database based on collected response times and not the hardware characteristics of the Grid, then use this data to move the chunks to better locations and improve the overall response time of the grid.

III. Improved Placement of Chunks

A. Evaluation of Demand on Chunks

To assess the need to change the location of a chunk in the grid we propose to add two metadata to the chunks metadata managed by HDFS. Those metadata will be calculated by the DataNode and transmitted in the block report to the MasterNode. They will be updated every time a chunk is consulted.

These two metadata are:

- C: The chunk consultation rate, which is the number of times the chunks has been downloaded during a configurable period (such as one month)
- Tc: The mean read time of the chunk during the same period as defined in (1).

$$Tc = \frac{\sum_{i=1}^{C} t_i}{C} \quad (1)$$

C : The number of consultations.

t_i : Read time at Nth consultation.

Mean read time is calculated by the HDFS daemon running on the DataNode after each read operation.

The NameNode will have a consolidated and sorted table of all the chunks as the example Table 1:

TABLE I. Sample of Chunks Consultation Statistics

Id chunks	C	Tc (ms)
A	200	10
B	100	6
C	50	12
D	40	7

A perfectly ordered table should contain the chunks at the beginning with a maximum C corresponding to a minimum Tc.

However, analysis of the data recovered from a simulation cluster shows cases where highly requested chunks have relatively high response times.

Our goal is to be able to move the most requested chunks to the Nodes that give the best reading times (Pn).

Through metadata we can calculate the mean performance of each node through the formula (2):

$$Pn = \frac{\sum_{i=1}^{k} T_{ci} C_i}{\sum_{i=1}^{k} C_i} \quad (2)$$

Pn: Mean performance of the node n

k : Number of chunks in the node n

Ci: Number of consultations of Chunk i placed in the node n.

Tci : Mean read time of the Chunk i placed in the node n.

Moving a chunk to a location with better reading times does not necessarily mean that response times on this chunk will improve, as the mean response times also depend on the nature of the request (location of reading requests, processing on the data...), but Tc's evaluation algorithm will continue to collect response times for the moved chunks on its new location and will allow to reassess the need or not to move it to a better location.

After one to several iterations of chunks moves, the improvement on the mean consultation time will eventually be observable or at least stagnate in its minimum value, meaning that the chunks are on the best location in terms of response time.

The number of chunks moves must be limited so as not to saturate the network more than necessary.

Movements of Data should be made at a time when data access is low or absent, so that the bandwidth will not be over occupied by this operation.

B. Algorithms

<u>Optimizer</u>

The Optimizer function (Fig. 4) scans the table of statistics of chunks consultation starting from the greatest number of consultations, which corresponds to the most requested chunks. This chunk should be placed in the best Node in terms of response time. In the best case the chunks with highest C will have to be moved to the Node whose have lowest Pn. The algorithm will try to retrieve an available Node with the best Pn.

```
Function Optimizer(Table chunks_Table)
{
--Table ordered by decreasing number of consultations –

Chunks_OrdredDescByC = OrderDescByC(chunks_Table);
        Foreach (chunk in Chunks_OrdredDescByC)
        {
                DestinationNode=
                GetBestNodeForchunks(Chunk chunk);

                MoveChunkToNode(chunk,
                DestinationNode)

        }
}
```

<u>GetBestNodeForchunks</u>

The GetBestNodeForchunks (Fig. 5) function allows retrieving the Node which offers the best average consultation time (Pn) which suggests it is a better location for a given chunk.

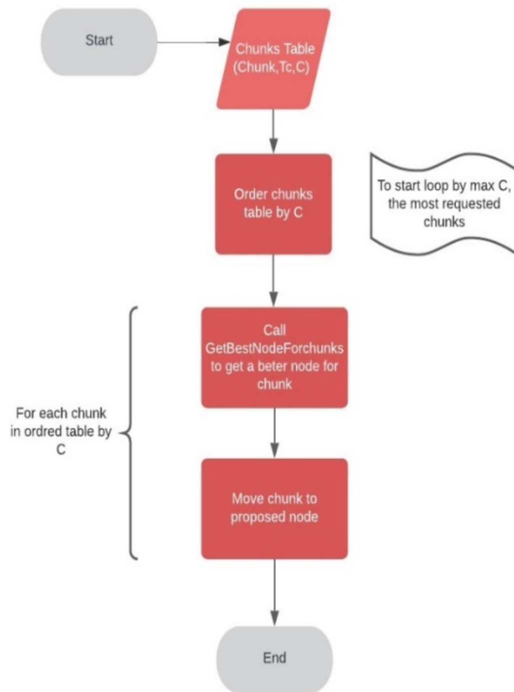

Fig. 4. Replacement Algorithm.

To do this, we calculate Pn for each node in the table and we go through the table of means of consultation time of the chunks, ordered up wards according to Pn, we check for each node its eligibility to receive the chunk passed in parameter.

Eligibility is based on four criteria:

1) The node is different from the one where the chunk is already placed.

2) The availability of space on the node.

3) The Node does not already contain a chunk replica.

4) The Node is not in the same rack as two other replicas.

Those criteria comply with Hadoop's basic policy, i.e., no more than two replicas in the same rack.

```
Function Node GetBestNodeForchunks(Chunk chunk)
{

--Table ordered by decreasing number of consultations –

Chunks_OrdredAscByPn = OrderDescByC(chunks_Table);
Foreach (betterPnChunk in Chunks_OrdredAscByPn)
        {
          If (betterPnChunk.Pn< chunk.Tc
                  {
               Node node = getChunkNode(betterPnChunk);
              If (Eligible(node,chunk)
                              {Return node;}
                  }
          }
        Return Null;
}
```

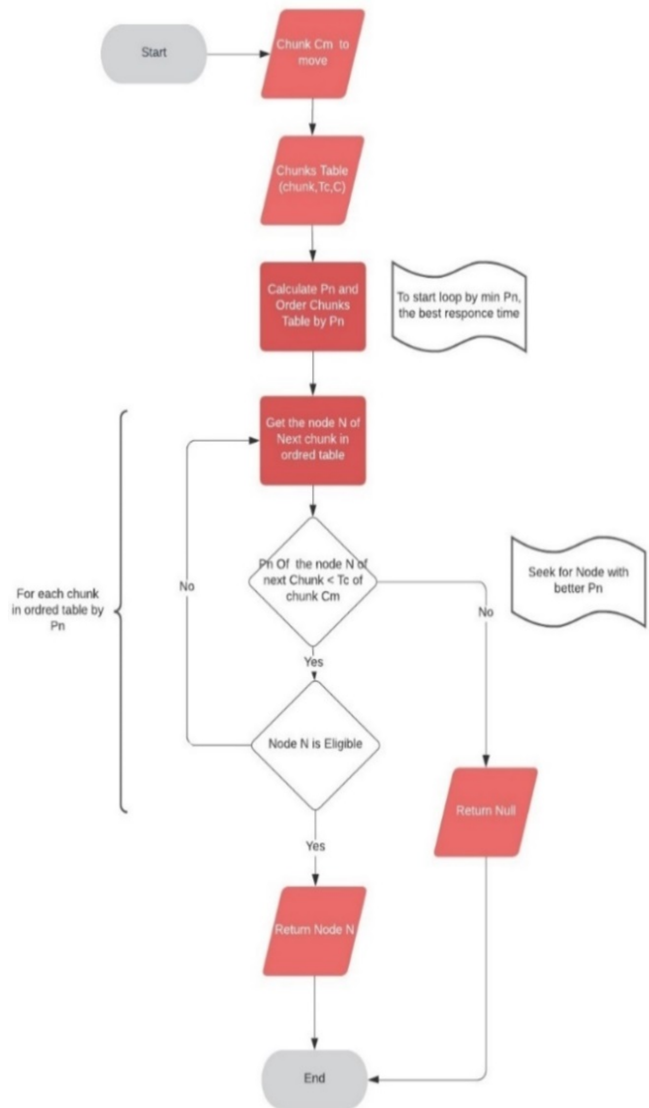

Fig. 5. Best Node Algorithm Selection.

IV. Testing Protocol

To make the performance measurements, we conducted simulations on a test grid consisting of 12 nodes.

The testing environment of the response time improvement algorithm consists of three racks (Fig. 6), each composed of four nodes of identical storage capacity and equal to three times the size of one chunk size (3x64MO).

Choosing this size allows us to limit the storage capacity of a node to a maximum of three chunks, in this way we will quickly saturate the nodes and the algorithm's ability to take this into consideration will be tested.

The performance characteristics of the nodes are heterogeneous; each of the three racks consists of the nodes of Table II.

TABLE II. GRID'S NODES CONFIGURATION

Node ID	RAM (GO)	Processor (Cores * speed)	Storage capacity
1	2	1 * 2GHz	192 MO
2	4	1 * 2GHz	192 MO
3	6	2 * 2GHz	192 MO
4	8	4 * 2GHz	192 MO

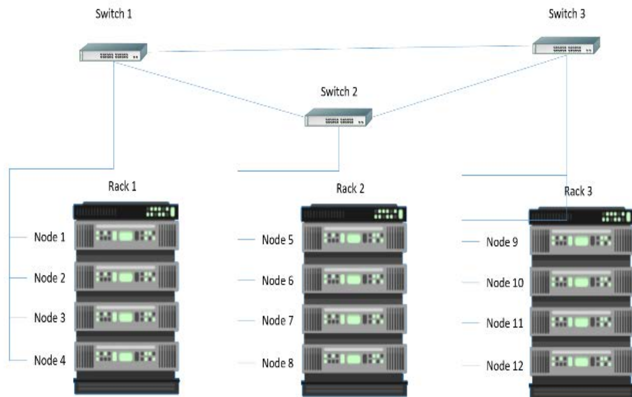

Fig. 6. Test Grid Architecture.

The bandwidth inside the same rack is 1 GB/s.

The direct bandwidth between the racks is set up according to the matrix in Table III (in MO/s).

TABLE III. GRID'S BANDWIDTH CONFIGURATION

	Rack 1	Rack 2	Rack 3
Rack 1	1000	100	10
Rack 2	100	1000	10
Rack 3	10	10	1000

The version of Hadoop used in the simulation is 2.5.0.

The simulations were conducted using 12 files with a unit size of 64MB.

The Size block configured on Hadoop is 64MO and the replication factor is 3.

Our TestJob consists of performing a variable number of jobs (map tasks) from the 12 nodes of the Grid on each of the 12 files distributed evenly on the Grid.

Each job is calling only one file, son the consultation rate C is equal to the number of executed jobs.

Testing protocol:

We followed a test protocol as shown in Fig.7:

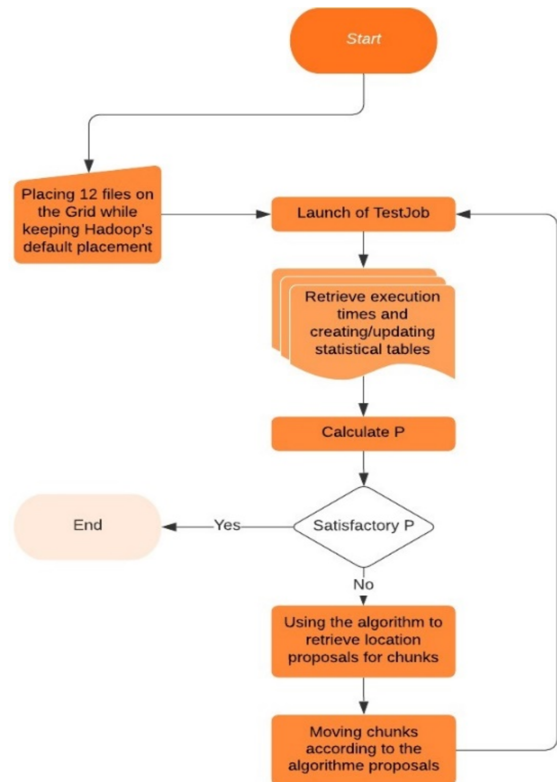

Fig. 7. Testing Protocol.

The phases were repeated over several iterations and after each iteration, we calculated the overall performance P of the cluster.

V. RESULTS AND EVALUATION

The main objective of running the test simulation is to calculate P before and after using the proposed algorithm. Thus, we can observe the evolution of P and assess the relevance of our approach.

The results collected on the first iteration of jobs are represented in the Table IV:

TABLE IV. INITIAL CHUNKS STATISTICS TABLE

Chunk Id	Id Node	C	Tc (ms)
0	0	12	20000,33
1	1	11	21626,64
2	2	10	23117,40
3	3	9	20143,00
4	4	8	16487,00
5	5	7	9967,00
6	6	8	26891,00
7	7	9	19819,44
8	8	10	31952,60
9	9	11	38492,09
10	10	12	35360,33
11	11	9	35135,00

C is the number of executed Jobs witch is equal to the consultation rate because each job is calling only one file.

The file is corresponding to only one chunk because of the chosen size of file and chunk configuration.

Tc is the mean job time of each file (MJT)

The overall performance of the Simulation Grid is then:

P1 = 25594.86 ms

The Tc calculated at this step is reflecting the default situation using the initial Hadoop placement strategy.

Running the relocation algorithm gives the locations suggestions in Table V:

TABLE V. CHUNK'S RELOCATION SUGGESTIONS TABLE

Id chunks	Id Node	Suggested Destination Node Id
0	0	5
10	10	5
1	1	4
9	9	4
2	2	7
8	8	7
3	3	None
7	7	None
11	11	0
4	4	None
6	6	3
5	5	None

The Algorithm proposes to relocate eight files to a new location.

Four files were not relocated because no improvement in the executions times is supposed to happen with any of available locations.

TABLE VI. CHUNKS STATISTICS TABLE AFTER FIRST RELOCATION

Id Chunks	Id Node	C	Tc (ms)
0	5	12	20000,33
1	4	11	21207,73
2	7	10	23117,40
3	3	9	20143,00
4	4	8	16487,00
5	5	7	9967,00
6	3	8	29259,00
7	7	9	19819,44
8	7	10	22736,60
9	4	11	17127,73
10	5	12	20000,33
11	0	9	19876,33

After moving the chunks to the suggested nodes and running the same set of jobs, the collection of response times gives the Table VI where Tc was recalculated:

The overall new performance of the grid is:

P2 =20125.21 ms

This value represents an improvement of 21.3% compared to P1.

Fig. 8 shows the evolution of the mean response time up to the third iteration.

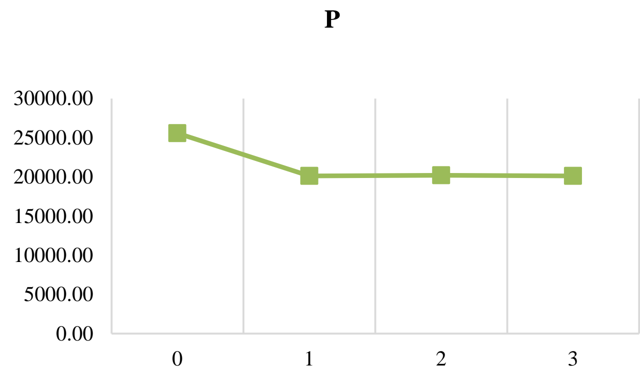

Fig. 8. Overall Performance Evolution.

After the first iteration P stabilizes. Changing the locations of the chunks according to the algorithm's proposals hardly affects the overall performance of the Grid.

We can conclude that the chunks are in an optimal node from the first iteration of replacement of chunks.

The overall performance improvement of the Cluster is nearly 20%. This improvement has been achieved after a single chunk replacement operation.

VI. CONCLUSION

The HADOOP system is a powerful and flexible big data system. Through its architecture it makes it possible to carry out complex operations on mass data distributed on grids which can range to thousands of machines whose characteristics can be very different.

Data striping and replication are among its strong features, thanks to them it manages to maintain the integrity of the system. But its basic chunk location strategy does not take advantage of all the capacity that the characteristics of the grid can offer.

In this research we conducted a deep analysis of how HDFS works to propose an improvement in the replica location strategy, and we demonstrated through our simulation that a substantial gain of more than 20% on the overall performance of the cluster is possible, while respecting the basic rules of HDFS initial chunk placement strategy.

In our future works we plan to integrate a module determining the best locations for data from the first writing, based on an artificial intelligence model capable of predicting the response time of a node for a given chunks. This will

reduce data movements in the clusters and avoid unwanted bandwidth usage.

REFERENCES

[1] Statista. "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025" Statista Digital Economy Compass https://www.statista.com/ (accessed Feb. 2019)

[2] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya, "Big data computing and clouds: Trends and future directions," J. Parallel Distrib. Comput., vols. 79-80, pp. 315, May 2015.

[3] Y.Wu, F.Ye, K. Chen, and W. Zheng, "Modeling of distributed file systems for practical performance analysis," IEEE Trans. Parallel Distrib. Syst.,vol. 25, no. 1, pp. 156-166, Jan. 2014.

[4] C. Verma and R. Pandey, "Comparative Analysis of GFS and HDFS: Technology and Architectural landscape," 2018 10th International Conference on Computational Intelligence and Communication Networks (CICN), 2018, pp. 54-58, doi: 10.1109/CICN.2018.8864934.

[5] W. Dai, I. Ibrahim and M. Bassiouni, "A New Replica Placement Policy for Hadoop Distributed File System," 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016, pp. 262-267, doi: 10.1109/BigDataSecurity-HPSC-IDS.2016.30.

[6] V. Venkataramanachary, E. Reveron and W. Shi, "Storage and Rack Sensitive Replica Placement Algorithm for Distributed Platform with Data as Files," 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), 2020, pp. 535-538, doi: 10.1109/COMSNETS48256.2020.9027494.

[7] T. Ma, F. Tian and B. Dong, "Ordinal Optimization-Based Performance Model Estimation Method for HDFS," in IEEE Access, vol. 8, pp. 889-899, 2020, doi: 10.1109/ACCESS.2019.2962724.

[8] The Apache Software Foudation. "Apache Hadoop 3.3.1" Apache Hadoop. https://hadoop.apache.org/docs/current/ (accessed Mars 2021).

[9] J. Liu et al., "A Low-Cost Multi-Failure Resilient Replication Scheme for High-Data Availability in Cloud Storage," in IEEE/ACM Transactions on Networking, doi: 10.1109/TNET.2020.3027814.

[10] V. Venkataramanachary, E. Reveron and W. Shi, "Storage and Rack Sensitive Replica Placement Algorithm for Distributed Platform with Data as Files," 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), 2020, pp. 535-538, doi: 10.1109/COMSNETS48256.2020.9027494.

[11] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," in Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, pp. 1-9, Atlanta, USA, 2010.

[12] Alshammari, Hamoud & Lee, Jeongkyu & Bajwa, Hassan. (2016). H2Hadoop: Improving Hadoop Performance using the Metadata of Related Jobs. IEEE Transactions on Cloud Computing. PP. 1-1. 10.1109/TCC.2016.2535261.

[13] Alanazi, Rayan & Alhazmi, Fawaz & Chung, Haejin & Nah, Yunmook. (2020). A Multi-Optimization Technique for Improvement of Hadoop Performance with a Dynamic Job Execution Method Based on Artificial Neural Network. SN Computer Science. 1. 10.1007/s42979-020-00182-3.

[14] Z. Li, H. Shen, W. Ligon and J. Denton, "An Exploration of Designing a Hybrid Scale-Up/Out Hadoop Architecture Based on Performance Measurements," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 2, pp. 386-400, 1 Feb. 2017, doi: 10.1109/TPDS.2016.2573820.

[15] M. Hajeer and D. Dasgupta, "Handling Big Data Using a Data-Aware HDFS and Evolutionary Clustering Technique," in IEEE Transactions on Big Data, vol. 5, no. 2, pp. 134-147, 1 June 2019, doi: 10.1109/TBDATA.2017.2782785.