International Journal on
"Technical and Physical Problems of Engineering"
(IJTPE)
Published by International Organization of IOTPE

ISSN 2077-3528
IJTPE Journal
www.iotpe.com
ijtpe@iotpe.com

December 2021 | Issue 49 | Volume 13 | Number 4 | Pages 194-200

DEEP LEARNING FOR OPTIMIZATION OF CHUNKS PLACEMENT ON HADOOP/HDFS

A. Elomari L. Hassouni A. Maizate

RITM-ESTC / CED-ENSEM, University of Hassan II Casablanca, Morocco
akramelomari@gmail.com, lhassouni@hotmail.com, maizate@hotmail.com

Abstract- In an era where the amounts of data generated by systems of different types have become gigantic, Big Data systems are trying to ensure proper management of this data by ensuring its security, accessibility, and optimization of its placement. This data explosion has over passed all traditional data storage infrastructures and solutions and has forced big data systems to adopt new solutions that can manage the new challenges of volume, variety, velocity, and veracity imposed by these huge amounts of data. Thus, storing data mechanisms in a big data environment like the Hadoop ecosystem is a pillar of its architecture. It comes as a sub-module called HDFS, capable of ensuring all operations related to data storage and accessibility. Among the operations that HDFS provides is the distribution of data over the Hadoop cluster, ensuring its subdivision and replication. This distribution follows a standard strategy that ensures the data durability and availability in all situations even in the event of hardware or software disasters. While effective for securing data, this strategy is not sufficient to optimize data placement in order to improve network flows and access times. Previous research has proposed to analyze the exploitation of data on HDFS and accordingly move the most requested data to nodes offering better response times. But this displacement generates additional traffic on the network, which can affect the general availability of the cluster. In this paper, we propose a mechanism for predicting the best Node in a DFS Grid that can guarantee the best response time, before placing data on the cluster. Thanks to this mechanism, we were able to reduce chunk movements by more than 30%, which led to a significant general optimization of the bandwidth occupation of the HDFS Cluster.

Keywords: Big Data, Hadoop, HDFS, Deep Learning, Placement Strategy, RNN, Performance Optimization.

1. INTRODUCTION

Market research reports show that the total amount of data consumed worldwide reached 64.2 zettabytes in 2020 and is expected to grow rapidly to 79 zettabytes in 2021. This number is expected to reach more than 180 zettabytes in 2025. Storage capacity will accordingly still be growing by an annual rate of 19.2% from 2020 to 2025 [1].

Such amounts of data can only be processed by Big Data systems, equipped with new architectures and technologies, capable of tracking the scale of this deluge of data.

Among the most important issues that such a system must deal with are those that storage poses in a big data environment. Indeed, the large amount of data to be stored requires new tools and techniques to guarantee the accessibility and sustainability of this data.

Many big data environments are based on distributed data storage systems called DFS (Distributed File Stems) [2]. DFS have a distributed architecture based on several nodes and networks, capable of guaranteeing the durability against hardware and software failures, while ensuring easy and fast access to this data.

Hadoop, one of the pioneering systems in the world of Big Data, uses HDFS which is its standard DFS. The HDFS provides Hadoop with the ability to complete complex computations by bringing computational operations closer to distributed data across multiple machines to improve processing performance. The HDFS uses several techniques to facilitate data management, such as data striping, which allows data to be subdivided into small pieces called Chunks, and replication to several nodes in the cluster to avoid total data loss in case of failure.

Replicas are usually generated when the file is created on HDFS and distributed over the cluster according to a basic strategy that has a unique objective of avoiding a total loss of data in case of a disaster affecting one or more machines or racks [3].

However, given the size that a DFS cluster can take, response times can be directly impacted by the locations assigned to the data by the placement strategy algorithm [4][5]. Therefore, the chunks placement strategy can be strongly involved in improving the overall performance of the Cluster, if it is smartly designed to be capable of choosing the best location for a chunk while respecting the constraints of "Disaster Recovery" provided for by the original HDFS placement strategy.

In a previous work, we proposed to improve this placement strategy by introducing new data into the HDFS metadata schema, which allowed us to calculate the best location for a chunk, based on the previous performance of

the cluster nodes and the level of chunk solicitation. The result of analyzing this new metadata is a proposal to move chunks that are the least privileged and the most requested in reading process, to better locations if there are any. Obviously, additional data traffic will emerge because of move suggestions, and a Chunk can even be moved several times before finding an optimal node [6].

In this paper, we introduced a Deep Learning model, that can process collected performance information and requests statistics of Chunks already placed on the cluster and predicting the best location for Chunks at its first creation on the cluster. The main goal is to minimize data movement, and consequently, the excessive occupation of the cluster bandwidth [7].

We built a supervised learning method using recurrent neural networks (RNN). The training Data was based on the history of chunks' movements over the cluster and further information such as the identity of the writer and the size of the chunk.

Training data was collected using the OptorSim simulator which allowed us to perform many simulations to create a rich and reliable training Dataset.

The remainder of this paper is presented as follows: Related works in section 2 about the problematic of optimizing chunk placements. The objectives of the Deep Learning model in section 3. The Deep Learning Model we used in section 4. Discussion about results and loss measurement in section 5 and we finish this paper with a conclusion about the whole work.

2. RELATED WORK

In a previous work [8], we analyzed the architecture of HDFS, in purpose to propose an enhancement to the strategy of replica placement, which is by default basic and very limited. An improvement of more than 20% on the overall performance of the cluster is demonstrated, while respecting the basic rules of HDFS' initial placement strategy, which guarantee stability in case of failures.

Beyond this result, we have demonstrated that the distribution of data across the cluster can have a significant impact on the overall performance of a DFS system, and that it makes sense to re-evaluate this distribution over time, in order to optimize and improve it.

As a result, the strategy of placing blocks on nodes should not only take care of the initial locations, but also follow the state of response times, and make sure to move the chunks when needed [9].

One side effect of this new strategy is that the continuous movement of data can have an impact on the bandwidth of the entire cluster. Therefore, these movement operations should only be carried out during off-peak periods of bandwidth usage. Since the movement of chunks is not a primary operation, it can be postponed to periods when the operations on the cluster decreases. These periods can be determined by the Master Node on HDFS which can support the orchestration of these operations and their programming over time.

That said, finding another way to properly place chunks without having to move them around too often can be much more relevant. This can be done if we have a way to recognize the best location for a block of data before making the first placement.

In this work we have therefore turned to another alternative to mathematical calculation: Prediction through artificial intelligence. We used the previous moves of the chunks on the Grid, to predict the possible movements of a newly created chunk, and then designate the location that seems to best meet the response time requirements of that chunk. This prediction model has reduced the displacements proposed by the new chunk placement strategy based on demand and response time analyzes, by at least one displacement, while a chunk is moved three times in average to meet his optimum node. The model based on the prediction of the first placement of the chunk in the GRID has reduced traffic related to the movement of chunks by at least 33% in most cases of chunks requiring displacement to improve response times.

The prediction model we propose must work closely with the proposed algorithm for moving chunks, because the training dataset of our model comes from the database constituted by the results of data displacement proposed by the algorithm of the new strategy of moving chunks on HDFS.

In the next sections we will define a recurrent neural network (RNN), able to predict the Node that should be selected to displace a chunk to, that offers a better response time. If the response time of a node is correctly predicted, we can place the chunk at its best node at the time of its creation. And depending on the sharpness of the prediction, the work of moving chunks to better locations can be remarkably lightened.

3. OBJECTIVE OF DEEP LEARNING MODEL

As discussed in section 2, we were able to prove that setting up a system for selecting nodes to which a chunk can be moved to have improved response times (which we call chunk displacement node) is possible. By implementing an improved placement node selection strategy, significant improvements can be achieved on chunk reading times and the overall performance of the HDFS Cluster. So, we set up an algorithmic model that, based on data collected from nodes and other calculated information, can determine which of the nodes in the cluster can provide better response times for a chunk. The algorithm then orders the chunk to move and redo these calculations to see if another node can provide a better response time.

This algorithm has proven its effectiveness after a few iterations of displacement. But in order not to penalize the bandwidth of the cluster, data movements must be made at times of downturn in activity on the cluster network.

It should be noted that moving a chunk to a new node supposed to offer a better response time, does not necessarily guarantee the achievement of good results. Since the performance of a node is calculated based on the reading times of the chunks it contains previously, the reading times of the recently moved chunk can vary from the old averages if the operating conditions of this one is different.

Analysis of the exploitation of our new placement strategy implemented through our algorithm has shown that some chunks are moved more than once to find their best locations. Chunks are sometimes even sent back to a previous location after recalculating response times.

The deep learning model we want to implement should allow us to predict the final location of a chunks at its first creation. This would avoid unnecessary displacements that the chunks' movement algorithm will surely propose.

We therefore want to move from the schematized situation in Figure 1 to the schematized situation in Figure 2. If the algorithm of the new placement strategy will propose to move the chunk according to the following sequence (Figure 1):

1- Creation of the chunk on Node 1,
2- After calculations, the improved placement strategy, algorithm suggests moving the chunk to Node 2 expected to offer a better response time,
3- After update calculations, the algorithm suggests moving the chunk to Node 3 expected to offer a better response time than Node 2,
4- After update calculations, the algorithm suggests the chunks to Node 2 once again since the actual readings times from Node 3 are lower than those recorded on Node 2.

The algorithm moves the chunk tree times before arriving at an optimal location. The DL prediction model should suggest that we place the chunk directly on Node 2 at the moment of its creation (Figure 2).

The question our model will have to answer is: Given a node or sequence of nodes on which a chunk has been placed successively, what is the next most likely node to be a placement proposal for that chunk? This is the task for which we train the model. The input of the model will be a sequence of node IDs in the Grid, and the model will need to be able to predict the next node capable of offering better response time. The last proposition should be the best one.

The main objective is therefore to predict the best location for a chunk by trying to predict all the nodes where it can be moved by the replacement algorithm and keep the last prediction that is supposed to be the best.

The algorithm developed in our previous work performs the displacement of chunks on average two to three times before the chunk is placed on an optimal node for the response time. A good prediction model will avoid at least one unnecessary data movement on the cluster network.

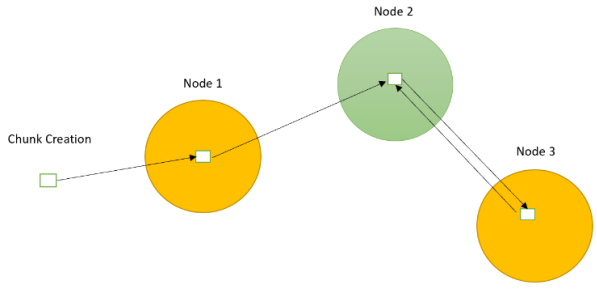

Figure 1. Moving the chunk without predicting the best location

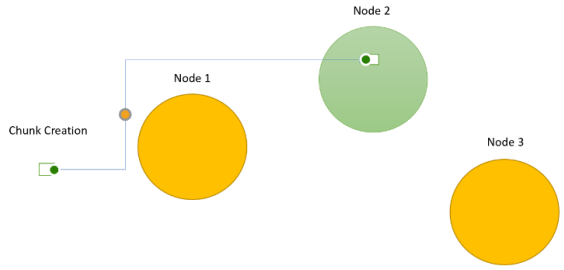

Figure 2. Placing the chunk with a good prediction of the best location

The prediction model will not replace the chunks movement algorithm based on calculations, but the two initiatives will complement each other to find the best locations for chunks, while avoiding unnecessary traffic. In addition, the algorithm based on the improved chunk placement strategy is the empirical way to be sure if a location is the best for a given chunk. The algorithm is also the provider training data for the prediction model.

4. DEEP LEARNING MODEL DESIGN

4.1. Training Data

To train our Deep Learning model and prove its effectiveness, a dataset of learning parameters was setup using a simulator (optorSim), combined with the algorithm developed in our previous works and escribed in section 3.

OptorSim is a simulator of Grids, used to simulate experiences on replications strategies in distributed architectures like DFS.

We built a Grid, using OptorSim capabilities, and used the algorithm of placement optimization to perform multiple iterations of data displacement over the grid, to obtain better response times. Every displacement of chunks in the Grid is prepared and added to the training dataset.

The keys data of the training dataset are [10]:

- *P* point of creation of the chunk: it is a unique identifier that makes it possible to recognize the point of creation of the chunk. The creation point is the location of the user who create the chunk. The point of creation of chunk has a great impact on the future exploitation of this chunk and therefore on future performance related to this data.
- *T* size of the chunk: in a system like HDFS the size of the chunk is generally fixed for all files, however work has shown that manipulating the size of chunks by modifying the data striping operation, can have positive results on response times. We conducted the constitution of our training dataset using several sizes of chunks in order to enrich our training dataset
- History of the different locations of the chunks. The objective behind the inclusion of all past locations of the chunk is precisely to allow the artificial intelligence to include these locations as possible locations of the chunk.

The location history is a composed input data, which we have split to form the final labeled training dataset [11].

The location history of a chunk before reaching an optimum location can be represented by a chain of positions with several values, where each value represents an *ID* of a node, for example (10, 21, 5, 11, 128, 200).

The first ID represents the position proposed by HDFS' original placement strategy.

We constitute our learning Dataset by extracting each pair P_n, $P_n + 1$ from the chain of positions. The Target being the position $P_n + 1$. The preceding sequence will be transformed to several couples (origin node, destination node).

(10, 21), (21, 5), (5, 11), (11, 128), (128, 200)

Consequently, each line of the initial Dataset constituted by the values (P, T, H) will be subdivided into several lines (P, T, P_n, $P_n + 1$) where the column $P_n + 1$ is the target of the model.

To use the prediction model, HDFS will therefore have to query the model at the time of chunk creation with the P_n position parameter being the *ID* of the node proposed by the HDFS original placement strategy. The algorithm will request the prediction of the next node several times again until the proposed node is already in the history of predictions or reach a determined number of predictions. We set the number of predictions recovered at three successive predictions because we noticed that in most cases, moving the chunk three displacements is sufficient to place it in an optimal node. The training dataset structure is as the following Table 1, where *ID* Node $P_n + 1$ is the target of our model.

Table 1. Sample of training dataset

USER *ID* (creation point)	Size of chunk (T)	ID Node P_n	ID Node $P_n + 1$
X	2000	10	6
Z	1000	6	1
W	500	12	3
Y	4000	7	19

4.2. Recurrent Neural Networks (RNN)

The Deep Learning model we used is a Recursive Neural Network. This type of neural network, often used in natural language prediction models [12][13], is best suited to our purpose. Indeed, we can assimilate the prediction of the next placement node in a Grid to the search of the complementary letters of a word or the next word of a natural and logical sentence.

The peculiarity of RNN networks compared to other neural networks is that they use the previous outputs as additional inputs and are perfectly adapted to the processing of sequential data [14][15].

This adapts to our scenario since the previous movements of chunks through the nodes of the Grid must participate in the prediction of the nodes $N + 1$.

The architecture in Figure 3 [16] represents a standard recursive neural network.

Each time the model is executed, the previous results and an internal state of the model are transmitted. The model returns a prediction for the next node and its new state. We retransmit the prediction and the new state to continue generating positions.

We used Google's TensorFlow library and the Keras Machine Learning library, inside the Anaconda environment.

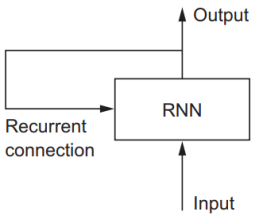

Figure 3. Traditional RNN architecture

5. MODEL ERROR ASSESSMENT

For the evaluation of the model error, we used two loss functions: Mean squared error (MSE) and mean absolute error (MAE) (Figures 4 and 5).

5.1. Mean Squared Error

Mean Squared Error (MSE) is the most known loss function. However, it remains relevant to assess the gap between the performance of the learning model and the real expected values (labels).

The MSE is computed by calculating the difference between the predictions and the expected values [17]. This difference is squared to avoid negative values.

The mathematical Equation (1) for the Mean Squared Error is therefore.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - y'_i\right)^2 \quad (1)$$

where, y_i is the empirically measured value and y'_i is the corresponding prediction calculated by our prediction model.

The MSE is ideal to ensure that the trained model does not have aberrant predictions with huge errors, since it gives greater importance to these kinds of errors because of the squared part of the function that amplifies the glaring deviations.

On the other hand, if model makes a wrong prediction quite far from the actual value, the squared part of the function will amplify this error and will affect the whole prediction results. However, in many practical cases, those bad predictions are ignored and the most importance is given to a model that works well in most cases.

The analysis of the evolution of the mean squared error of our model (Figure 4) shows that the value of the MSE stops evolving substantially after the Epoch 1,000. This indicates that the model has converged well.

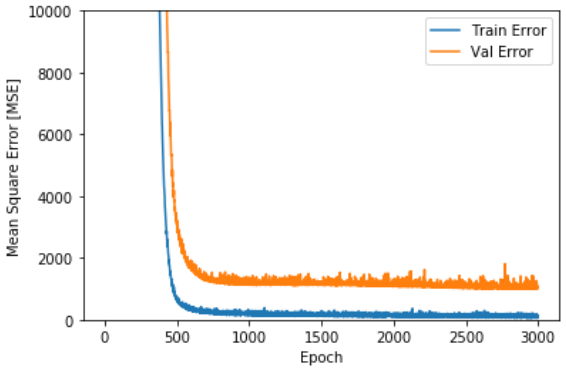

Figure 4. Evaluation of Mean Squared Error

The curve of the loss function stabilizes at its lowest level, which confirms that the model has converged and that it can be used in the desired prediction.

The training and validation loss decrease exponentially as the number of epochs increase, meaning that the model become more accurate ate the end of training.

5.2. The Average Absolute Error

Mean Absolute Error (MAE) is slightly different from the MSE, but it provides very a different assessments of loss gaps. The MAE is the sum of the absolute values of the difference between the model's predictions and the expected value, averaged against all the data.

All calculated values in the Mean Absolute Error have the same weights. Therefore, large errors value does not have an exaggerated weight. The Mean Absolute Error function provides a uniformed measure of the model performance [18] [19].

The mathematical Equation (2) for the Mean Absolute Error is:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - y'_i\right| \quad (2)$$

where, y_i is the empirically measured value and y'_i is the corresponding prediction calculated by our prediction model.

Analysis of the Mean Absolute Error curve shows that the model converged.

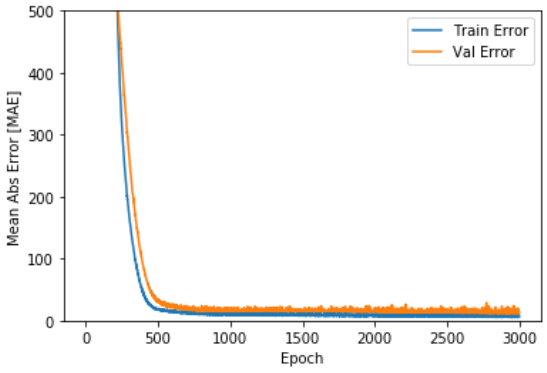

Figure 5. Evaluation of the Mean Absolute Error

We have arrived at a stable and acceptable average (Figure 5) of errors in our context as the validation loss is stable at his nearest values to 0. We thus confirm the convergence of the model and the values collected during the evaluation of the mean squared error.

5.3. Prediction Test

The ultimate test for a deep learning model is the confrontation with new labeled data never introduced during the learning phase.

That's why we divided our Dataset into two parts, one that was used for learning and another that we used to test our model with.

The curve in Figure 6 illustrates the difference between the predictions on the Test Dataset and the labels on the same Dataset.

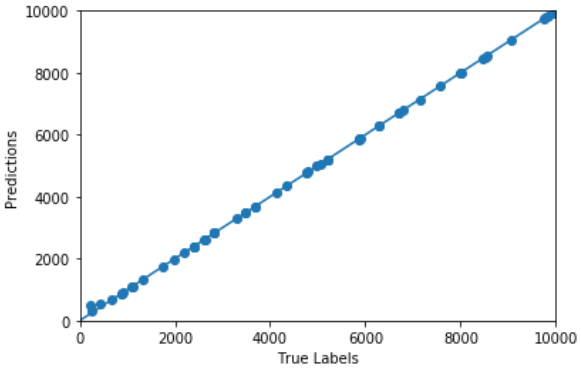

Figure 6. Prediction test

The predicted values are quite similar to the actual values. The results of the model prediction test are satisfactory so the model can be considered as accurate.

6. CONCLUSION

In a previous works we demonstrated the response time savings that can be generated by placing data on appropriate nodes on a DFS cluster. We calculated the best location for a Chunk and moved it to that location.

We also discussed the cost of moving chunks over the cluster on the network bandwidth and proposed that it must be done outside the period of intense operation of the cluster.

However, having the ability to determine the best location for a Chunk without having to move it after its first placement will certainly reduce the impact of our algorithm on bandwidth.

Determining the appropriate nodes in terms of response time for a Chunk means determining the nodes through which the chunk can transit during these movements, proposed by the chunk replacement algorithm, and selecting the last proposed node, assuming that it is this node that offers the best response times.

Given that the chunk placement algorithm proposes each time to displace the chunk to a single node, we built a deep learning model that can predict the next node that the algorithm will propose to host the chunk. Running the model recursively, we were able to define a set of transit nodes and select the last node in the transition chain to be the first placement of the chunk.

To establish the Training Dataset, we adapted the list of all previous movements of Chunks -provided by the replacement algorithm- by defining the destination node of each movement as the target of the DL model.

The main objective of this work is to prove the possibility of using Deep Learning to predict an optimized location of a chunk on a network of nodes of a DFS Cluster. Then place the chunk in the predicted placement at his creation.

Given that the replacement algorithm proposes to move the disadvantaged chunks (in terms of response time) to other nodes able to ensure a better response time, it was judicious to use AI to reduce these movements at most, by predicting the destination to which the displacement algorithm will lead.

In the case of HDFS for example, when creating the three replicas of a chunk, the Master offers three slots according to the standard HDFS strategy, namely two nodes on the rack of the user who takes care of the writing and a third on another rack. By including the location prediction model in this phase, the client code can write the chunk directly to the most optimized location.

The algorithm of replacement will then confirm this proposition based on statistics or propose a new node that will be taken to consideration by the deep learning model, for the next predictions.

NOMENCLATURES

1. Acronyms

DFS	Distributed File System
HDFS	Hadoop File System
DL	Deep Learning
RNN	Recursive Neural Network

REFERENCES

[1] Statista Digital Economy Compass, www.statista.com, 2021.

[2] A. Elomari, A. Maizate, L. Hassouni, "Data storage in big data context: A survey", The 3rd International Conference on Systems of Collaboration (SysCo), 2016.

[3] G. Bhatt, M. Bhavsar, "Performance analysis of local, network and distributed file systems running inside user's virtual machines in cloud environment", Advances in Modelling and Analysis B, vol. 61, no. 1, pp. 48-55, Mar. 2018.

[4] N. Mostafa, I. Al Ridhawi, A. Hamza, "An intelligent dynamic replica selection model within grid systems", IEEE 8th GCC Conference and Exhibition, 2015.

[5] M. Nithya and N.U. Maheshwari, "Load rebalancing for Hadoop Distributed File System using distributed hash table", International Conference on Intelligent Sustainable Systems (ICISS), 2017.

[6] Y. Wu, F. Ye, K. Chen, 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.

[7] M. Hajeer, D. Dasgupta, "Handling Big Data Using a Data-Aware HDFS and Evolutionary Clustering Technique", Transactions on Big Data, vol. 5, no. 2, pp. 134-147, 1 June 2019.

[8] A. Elomari, L. Hassouni, A. Maizate, "New Data Placement Strategy in the HADOOP Framework", International Journal of Advanced Computer Science and Applications (IJACSA), 2021.

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

[10] E. Yazan, M.F. Talu, "Comparison of the stochastic gradient descent-based optimization techniques", International Artificial Intelligence and Data Processing Symposium (IDAP), 2017.

[11] C. Guobei, Y. Wenfu, L. Wei, Z. Xuan, Y. Jian, Z. Xiaoning, "Method for generating infrared big data for deep learning algorithm training by using small sample data", IEEE International Conference on Signal, Information and Data Processing (ICSIDP), pp. 1-5, 2019.

[12] J. Klejsa, P. Hedelin, C. Zhou, R. Fejgin, L. Villemoes, "High-quality Speech Coding with Sample RNN", ICASSP - IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 7155-7159, 2019.

[13] N. Atashafrazeh, A. Farzan, "A Review of Using Machine Learning Algorithms for Image Retrieval Words", International Journal on Technical and Physical Problems of Engineering (IJTPE), Issue 20, Vol. 6, No. 3, pp. 139-144, September 2014.

[14] Y. Denny Prabowo, H.L.H.S. Warnars, W. Budiharto, A.I. Kistijantoro, Y. Heryadi, et al., "LSTM and Simple RNN Comparison in the Problem of Sequence to Sequence on Conversation Data Using Bahasa Indonesia", Indonesian Association for Pattern Recognition International Conference (INAPR), pp. 51-56, 2018.

[15] D. Choi, J. Han, S. Park, S. Hong, "Comparative Study of CNN and RNN for Motor fault Diagnosis Using Deep Learning", IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA), pp. 693-696, 2020.

[16] F. Chollet, "Deep learning with python", O'reilly Media, p. 196, 2021.

[17] Y. Sai, R. Jinxia, L. Zhongxia, "Learning of Neural Networks Based on Weighted Mean Squares Error Function", Second International Symposium on Computational Intelligence and Design, 2009.

[18] L. Mendo, "Estimation of a probability with guaranteed normalized mean absolute error", IEEE Communications Letters, vol. 13, no. 11, pp. 817-819, Nov. 2009.

[19] S. Kheiri, V. Yousefi, S. Rajebi, "Evaluation of K-Nearest Neighbor, Bayesian, Perceptron, RBF and SVM Neural Networks in Diagnosis of Dermatology Disease", International Journal on Technical and Physical Problems of Engineering (IJTPE), Issue 42, Vol. 12, No. 1, pp. 114-120, March 2020.

BIOGRAPHIES

Akram Elomari was born in Meknes Morocco, 1981. He received his Engineering Diploma in Computer Science from Hassania School of Public Works, Morocco in 2004 and a Specialized Master on Project and Program Management from SKEMA Business School, France in 2011. He is currently an ECM and BPM specialist and have a various experience in output management and EDITIC field. In addition, he is preparing a Ph.D. in Distributed Data Storage system in University of Hassan II Casablanca, Morocco. His research interests are focalized on Big Data storage and analysis systems.

Larbi Hassouni has a Ph.D. in Computing Science from Aix Marseille University, France. He was an Engineer of Marseille Central School, Marseille, France. His research topics are big data, machine learning, deep learning, E-learning, IA.

Abderrahim Maizate was born in Casablanca, Morocco in 1979. He received his Engineering Diploma in Computer Science from the Hassania School of Public Works, Morocco in 2004 and DESA degree from ENSIAS, Morocco in 2007. Then, he received his Ph.D. degree from Chouaib Doukkali University, El Jadida, Morocco. He is currently, a Professor Researcher at Hassan II University, Casablanca, Morocco. His research interests are wireless communication, mobile communication, wireless sensor networks, quality of service (QoS) guarantees and big-data.