Journal of Theoretical and Applied Information Technology
31st March 2022. Vol.100. No 6

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPROVING HADOOP THROUGH DATA PLACEMENT STRATEGY

MOHAMED EDDOUJAJI[1] , HASSAN SAMADI[2], MOHAMMED BOUHORMA[3]

[1]Dep. Doctoral Studies Research Center, National School of Applied Sciences, Morocco

[2]Dep. Doctoral Studies Research Center, National School of Applied Sciences, Morocco

[3]Dep. Doctoral Studies Research Center, Faculty of Sciences and Techniques, Morocco

E-mail: [1]m.eddoujaji@gmail.com, [2]hsamadi@uae.ac.ma, [2]abouhorma@uae.ac.ma,

ABSTRACT

Hadoop assumes that the computing capability of cluster's nodes is similar. In such a homogeneous environment, every node is assigned an identical load, such approach allows total use of cluster's resources and minimize multiple idle or over headed nodes. However, in real world applications, clusters are frequently deployed in a heterogeneous context [9,13–15]. In such environments, there is possibly multiple different physical servers and different virtual nodes specifications, which provide per consequence different services capabilities. Therefore, Hadoop still uses the same native strategy that distributes data blocks equally among each DataNode, similarly the load is evenly assigned between nodes, then the basic overall performance of Hadoop may also be reduced.

The main objective during the phases of this research is to find an optimal scheme and an improved architecture to optimize the classical architecture of HADOOP and MapReduce, by focusing mainly on the algorithm of locality and data distribution in a heterogeneous ecosystem composed of several nodes heterogeneous.

Despite of the native Data Placement strategy that Hadoop framework maintains by default, we present in the following a new approach that take in consideration the difference between the cluster nodes computing capabilities, with respect to the nature of tasks to adjust data blocks distribution.

The design of our solution is presented under two major phases, the first one is implemented during the HDFS input, and the second one is implemented while processing tasks are initiated.

Keywords: *Data, Big Data, Distributed Systems, Heterogeneous Systems, Hadoop Distributed File System; Distributed Storage; Distributed Computing, Sequencefile, Mapfile*

1. INTRODUCTION

The major drawbacks of such implementation are the performance degradation of MapReduce when large clusters based on heterogeneous nodes are used, and also data locality has not been taken into account for geographically dispersed environments, because HADOOP assumes all data is stored locally.

In this article, we will discuss how to improve the two fundamental aspects in a classic MapReduce architecture; the homogeneity and locality of the data! and that if they are not taken into account, they can considerably impact the performance of MapReduce.

In our approach, and in the case of critical and resource-intensive applications, a balanced distribution of tasks to be processed according to the resources allocated for each node can always improve the performance of the HADOOP platform, particularly the operating mode of MapReduce.

The most important motive is that due to different cluster's nodes capabilities, the assigned tasks are consequently achieved asynchronously. Actually, some nodes are illegible to accomplish faster execution time than others. The assignment of the remaining tasks can be improved by selecting wisely the most appropriate nodes with respect to data placement. Avoiding data transfer and wait event for data fetch can minimize the processing time and enhance the overall Hadoop performance.

Journal of Theoretical and Applied Information Technology
31st March 2022. Vol.100. No 6

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. MOTIVATION

For example, in Figure 1, nodes (A, B, C), in this order from the fastest to the slowest. Assuming that node A can perform processing 3 times faster than node C, and 2 times faster than node B. Figure 1(a), shows the distribution of data blocks that are required for a given processing job as follow, 3 data blocks on Node A, 4 data blocks on both Node B and Node C. When processing starts, Node A will accomplish processing local data blocks earlier than others.

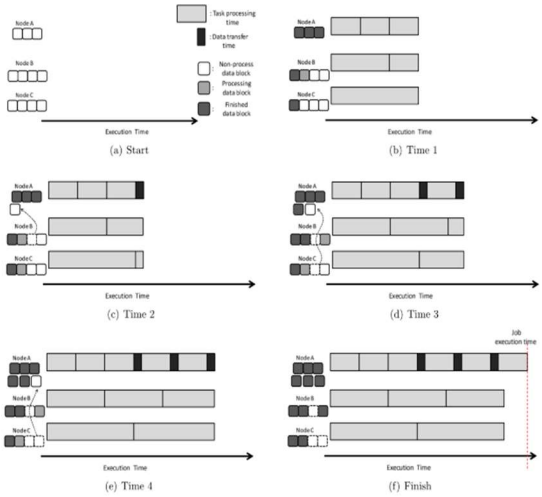

Figure 1: nodes A, B, C from the fastest to the slowest

Figure 1(b), shows that after a certain time, Node B and Node C are still processing assigned data blocks while Node A is waiting to be assigned new tasks by the NameNode. After time 2, Figure 1(c) shows that Node A have to wait for data block transfer from Node B to be completed before processing. Similarly, each time the faster node will be assigned task when others are busy, the cluster will cumulate delays to get data blocks for processing by the available node. Finally, Figure 1(f) shows 3 data blocks required by Node A from both Nodes B and C, causing delays while executing jobs. Considering the complexity and the scale of Hadoop clusters, data blocks transfer across big clusters, affects heavily the overall Hadoop performance.

3. PAPER ORGANISATION

In this paper, in section 1.1 we introduced a description on how the data placement and the nodes capabilities can be improved in Hadoop heterogenous cluster. In section 1.2, we present the design and the implementation, while describing the proposed algorithms. In section 3.3 we present our experimental implementation, results and evaluations. Finally, in Section 4.4 we conclude by a summary of the actual paper contribution.

4. DESIGN AND IMPLEMENTATION

4.1 Dynamic data placement strategy

Despite of the native Data Placement strategy that Hadoop framework maintains by default, we present in the following a new approach that take in consideration the difference between the cluster nodes computing capabilities, with respect to the nature of tasks to adjust data blocks distribution.

The design of our solution is presented under two major phases, the first one is implemented during the HDFS input, and the second one is implemented while processing tasks are initiated.

4.1.1 RatioTable

The RT -Ratio Table- is created in the NameNode, it keeps track of the data blocks when added to HDFS. The table will be used to decide if reallocation of relevant data blocks has to occur after task execution. The RT tracks task types and the computing capacity of the nodes. The computing capacity is given as a ratio, that indicate an average of how much time this node takes to execute tasks. The NameNode receives a heartbeat message from each DataNode, which include the computing capacity.

Table1: RatioTable example

	Node A	Node B	Node C
WordCount	3	1.5	1
Grep	2.5	1.5	1

For simplicity, the Table 1describes the same previous Nodes A, B and C, under the computing capacity assumption for the first job, $A(Compute\ Capacity) = 3xC(Compute\ Capacity)$ and $B(Compute\ Capacity) = 1.5xC(Compute\ Capacity)$. Then we assume that it is slightly different for the second job Grep, as $A(Compute\ Capacity) = 2.5xC(Compute\ Capacity)$ and $B(Compute\ Capacity) = 1.5xB(Compute\ Capacity)$. Then, the RT will record the following ratios respectively for WordCound and Grep as $3:1.5:1$ and is $2.5:1.5:1$

4.2 Phase 1: Initialization and Allocation

During data input through HDFS, the NameNode initiate the check into the RT. It will initially check for execution historical data about the same job. If the return is positive, then the NameNode will distribute data block to the previous DataNodes with respect to their computing ratio. Otherwise, the data blocks will be distributed equally according to the default Hadoop strategy, still the RT will record the new job mapped to the value 1 of the corresponding computing ratio of each assigned node.

According to the computing ratios reported in the Table 1, for a given WordCount job, if we assume an input data that contains 11 data blocks, the distribution of those data blocks based on our ratios will be allocated as follows:

Table 1: Allocation according to the ratio information

Assigned data blocks to Node A:	$\lvert(11 * \lceil \frac{3}{3+1.5+1} = 6)\rvert$
Assigned data blocks to Node B:	$\lvert(11 * \lceil \frac{1.5}{3+1.5+1} = 3)\rvert$
Assigned data blocks to Node C:	$\lvert(11 * \lceil \frac{1}{3+1.5+1} = 2)\rvert$

However, when no historical data are available for a given job, the NameNode can't get any records in the RT, then the data block distribution will be similarly among the 3 nodes as per the default Hadoop configuration. For example, for a new given TeraSort job, the computing capacity ratios (1: 1: 1) will be initialized in the RT for future decision (see Figure 2).

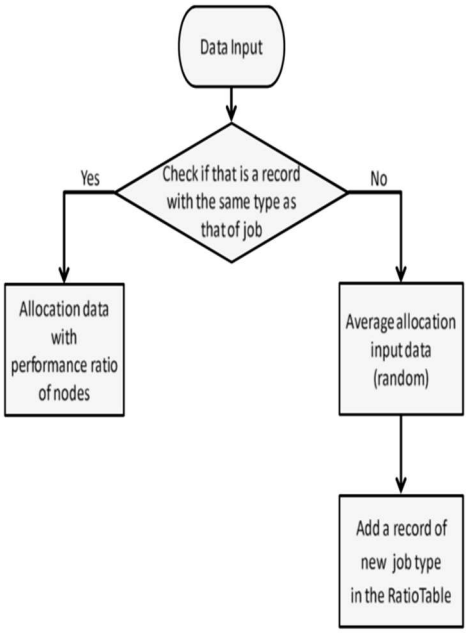

Figure2: Phase 1 process flow

4.2.1 Initial Data Allocation Algorithm

Algorithm 1: Initial_Data_Allocation.

When a *data* is written into HDFS:
JobType ← job type of the data will be performed;
DataSize ← obtain from data information;
BlockSize ← set by user;
$TotalBlockNumber = \lceil \frac{DataSize}{BlockSize} \rceil$;
set $Same = 0$;
for each *record* in the *RatioTable* **do**
  if compare *JobType* with *record* are the same **then**
    $Same = 1$;
    ComputingCapacityRatio ← obtain from *record*;
    for each *DataNode* in the cluster **do**
      NodeCapacity ← obtain from *ComputingCapacityRatio*;
      $BlockNumber = TotalBlockNumber * \lceil \frac{NodeCapacity}{\sum each\ node\ capacity} \rceil$;
      Allocate *BlockNumber* data blocks to the *DataNode*;
if $Same = 0$ **then**
  ComputingCapacityRatio ← set 1 for each node;
  Add *JobType* with *ComputingCapacityRatio* to *RatioTable*;
  for each *DataNode* in the cluster **do**
    $NodeCapacity = 1$;
    $BlockNumber = TotalBlockNumber * \lceil \frac{NodeCapacity}{\sum each\ node\ capacity} \rceil$;
    Allocate *BlockNumber* data blocks to the *DataNode*.

For simplicity in the current phase, the powerful node remains favorited to store the left-over blocks due to ponderation. In other words, when Total Blocks is different than Total assigned blocks, the powerful node A get additional blocks Diff Blocks, where Diff Blocks=Total Blocks – Total blocks assigned to A (as per the phase 1 algorithm).

Phase 2: Capacity Decision and Data Reallocation

Journal of Theoretical and Applied Information Technology
31st March 2022. Vol.100. No 6

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Considering a job J as a set of tasks, once job is started, tasks are defined, then assigned to nodes. When the tasks execution ends in each DataNode, DataNodes reports to the NameNode the statuses. NameNode will get also the ratios based on their execution time. As of the difference between their computing capacity, each node will have different availability for tasks slots. Actually, all the available task slots can be run in parallel, therefore the computing capacity must take into consideration both the number of available slots and ratios for more accuracy.

Consider Node A and Node B, while Node A is able to perform tasks two times faster than Node B. based on that, we suppose that the available task slots are also double on A compared to B. For simplicity, we will consider a number of task slots on both nodes. Assuming two on node A and four on node B. We assume that node A perform a given job tasks on the 4 task slots

respectively during 45, 43, 43 and 46 secs, as per an average of 44.25 for one task. Similarly, we assume that Node B perform a given job tasks on the 2 task slots respectively during 39 and 40 secs, as per an average of 39.5 for one task.

In term of efficiency, one can consider that Node B have better execution time for each task, compare to Node A, still node A can process 4 tasks simultaneously. Therefore, Node A can be favorited when the average compute time of a task divided by the number of slots is higher.

Let's consider the following

Table 3: Considerations

$Ta(X)$	Average execution time to accomplish a set of tasks on node X
$S(X)$	The number of task slots on node X

Then the average time required to complete one task on Node X is $Tt(X) = [\frac{\mathrm{Ta(X)}}{S(x)}]$

The NameNode will calculate the compute ratio for each DataNodes and compare it to the previous values if it exists in the RT. The value will be updated whenever previous value exists and different. Data blocks will be transferred based on the obtained ratios, to anticipate the execution of transferred data blocks on the faster node. (see Figure 3)

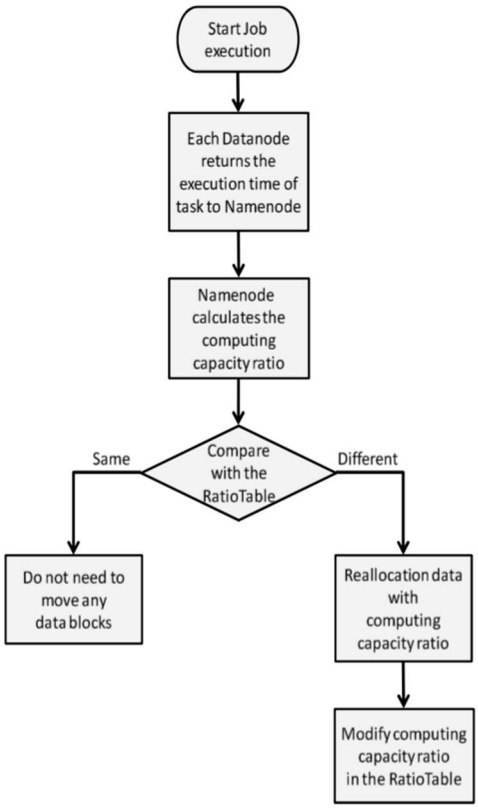

Figure3: Phase 2 process flow

4.2.2 Capacity Decision and Data Reallocation Algorithm

Algorithm 2: Capacity_Decision_and_Data_Reallocation.

When a *job* start:
NodeNumber ← obtain from *NameNode*;
CurrentNumber[*NodeNumber*] ← set 0 for all entries; // record the number of task execution time received from each node.
TotalExecutionTime[*NodeNumber*] ← set 0 for all entries;
while receive the *task execution time* from *DataNode*[*i*] **do**
  SlotNumber ← obtain from *DataNode*[*i*];
  ExecutionTime ← *task execution time*;
  TotalExecutionTime[*i*] = *TotalExecutionTime*[*i*] + *ExecutionTime*;
  CurrentNumber[*i*] = *CurrentNumber*[*i*] + 1;
  if *CurrentNumber*[*i*] = *SlotNumber* **then**
    $T_{avg} = \frac{TotalExecutionTime[i]}{SlotNumber}$;
    $T_t = \frac{T_{avg}}{SlotNumber}$;
    CurrentNumber[*i*] = 0;
    TotalExecutionTime[*i*] = 0;
  if obtain T_t for each node **then**
    PerformanceRatio ← *PerformanceRatio* and T_t are inversely proportional;
    for *record* in the *RatioTable* **do**
      if compare *PerformanceRatio* with *record* are different **then**
        Reallocation data blocks according to *PerformanceRatio*;
        Modify the *record* according to *PerformanceRatio*.

4.3 Experimental Results

In this section, we present our experimental results obtained through conducting test deployments to evaluate our proposed algorithms.

Environment

In order to conduct our experimental tests, we built a Hadoop cluster environment for test purpose. The hardware layer is abstracted by an OpenStack private cloud platform. That is used to provision compute, storage and network resources. Our cluster is simulated by one NameNode and 4 DataNodes. The NameNode VM0 has 4 CPUs, 32 GB of memory, and 500 GB of storage; The DataNode VM1 has 2 CPUs, 16 GB of memory, and a 500 GB of storage; The DataNode VM2 has 2 CPUs, 8 GB of memory and a 500 GB of storage; the DataNode VM3 and VM4 has 1 CPU, 4 GB of memory, and a 500 GB of storage. All Nodes are deployed on Ubuntu 14.04. We use Hadoop version 2.6.2 and Java version 1.8.0_65.

Our use of the virtualized resources on OpenStack provide us more flexibility to add or shrink Nodes capacities according to the purpose or each test.

Table 4: Initial resource allocation

Node	CPU	RAM	Disk
VM0 (Master)	8	64 GB	500 GB
VM1 (Slave)	4	32 GB	500 GB
VM2 (Slave)	4	16 GB	500 GB
VM3 (Slave)	2	8 GB	500 GB
VM4 (Slave)	2	8 GB	500 GB

5. RESULTS

In this section, we use WordCount and Grep, two micro benchmarks abstracted from big data Hadoop services over various implementations infrastructure. The WordCount program WordCount counts the occurrences of all words in the files. It takes as input any document format, text files are the most common input, and provide an output format $< word >< count >$. The Grep program searches for regular expressions from the input files then it counts the occurrences of the strings. It takes as input format: $< key >< value >$, and provide an output format $< key >< value >$. Both benchmarks are provided through Hadoop Libraries.

To evaluate our proposed algorithm performance, in a Hadoop heterogeneous cluster, we made several rounds of executions. Each round contains 10 jobs to be run simultaneously. The jobs input is all 1Gb, but not necessarily using the same datasets. The average execution time is related to each round of execution.

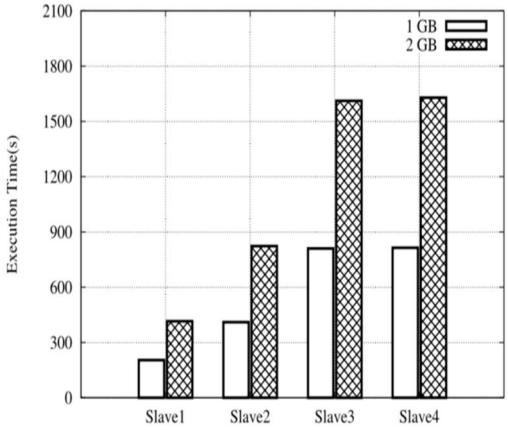

Figure 4: Execution time - WordCount job on each node

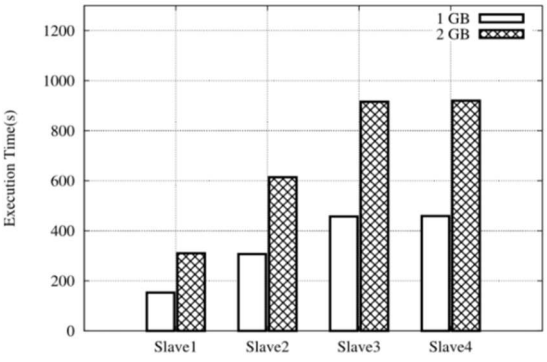

Figure5: Execution time - Grep job on each node

Initially, we execute both benchmarks using different size of data files 1Gb and 2Gb, Figure 4 shows the average execution for WordCount related to both data files sizes. Figure 5 shows the average execution time for Grep related to both data files sizes. Based on Figure 4 and Figure 5 we observed that the execution time is proportional to data sizes on each node. The computing capacity ratio between nodes is not affected by the processed data sizes.

According to Figure 4 and Figure 5 we calculated the compute ratio of each node based on the two types of jobs, as obtained in Table 5.

Journal of Theoretical and Applied Information Technology
31st March 2022. Vol.100. No 6

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

According to the first results, the size of the processed data is not heavily impacting the computing capacity ratio. For a WordCount job execution, we observed that VM1 node is four times faster than VM3 node, and VM2 node is two times faster than VM3 node 3.

Table 5: The job computing capacity ratio of each node

Node	WordCount	Grep
VM1	4	3
VM2	2	1.5
VM3	1	1
VM4	1	1

However, the proposed algorithm is principally designed to optimize map tasks, while execution time includes also reduce tasks. Moreover, there is slightly delays to consider during the calculation of the computing ratios, which might impact scalability performance. Therefore, we continue to execute different jobs to calculate the average time required to complete one single task.

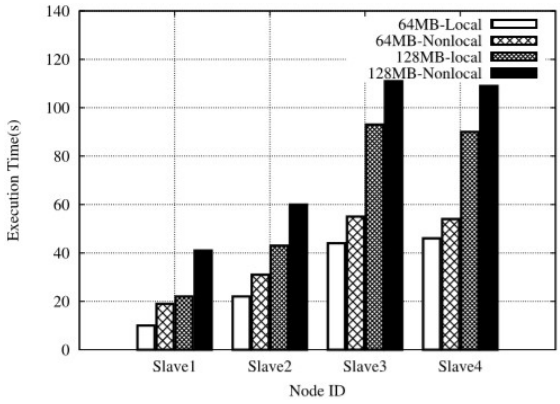

Figure 6: Average time - WordCount task on each node

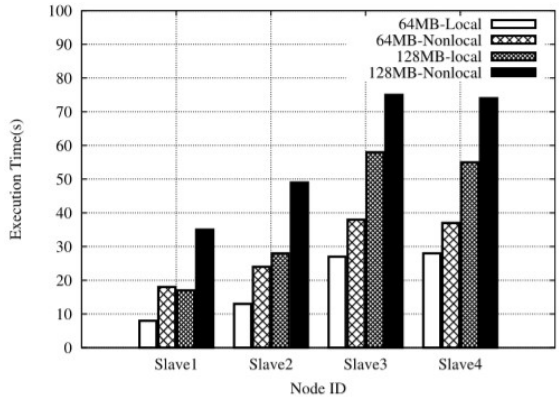

Figure 7: Average time - Grep task on each node

Figure 7 shows the average execution time of single WordCount task, on each node with respect to data locality. Figure 7 shows the average execution time of a single Grep task, similarly with respect to data locality.

We observe, that the executing time in both cases, locally and non-locally depends on the size of the processed a data block as shown in both figures. According to both figures 6 and 7, the execution time is proportional between the executed task and the data block size, regardless of the computing ratio on each node, therefore we adopted the same Hadoop default block size 64Mb.

According to Figure 6 and Figure 7 we calculated the compute ratio of each node for a single task, as obtained in Table 8. According to those results, for one single task of the WordCount job, we observed that VM1 is 4.5 times faster than VM3, VM2 is twice time faster than VM3. While for one single task of Grep job, VM1 is 3.5 times VM3, VM2 is twice faster than VM3. We conclude that those ratios are approximately close to the previous calculated ratios, only the job execution time is more accurate as it takes into consideration the reduce phases. Therefore, we update the ratios as in the Table 7.

Table 6: The task computing capacity ratio of each node

Node	WordCount	Grep
VM1	4.5	3.5
VM2	2	2
VM3	1	1
VM4	1	1

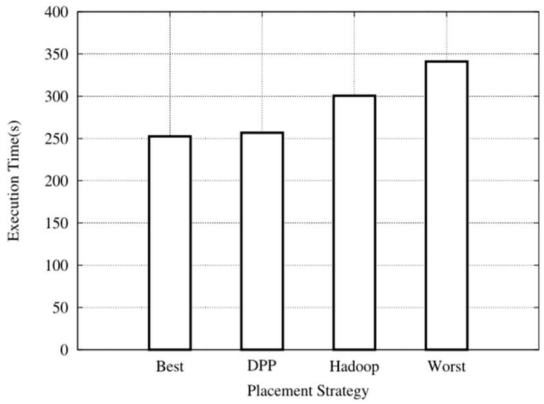

Figure 8: Data placement strategy impact on average execution time - WordCount

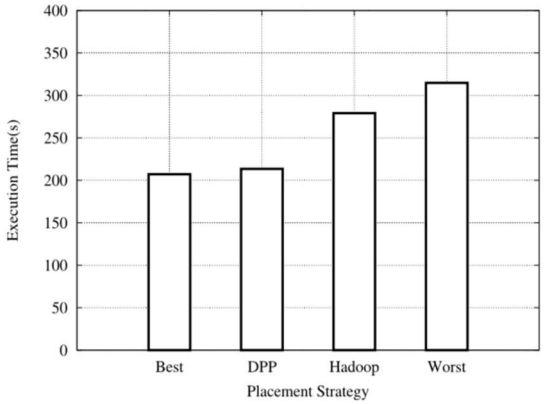

Figure 9: Data placement strategy impact on average execution time - Grep

Figures 8 and 9 shows the execution time for the same benchmark jobs Word-Count and Grep while different data allocation strategies are used. In this experiment, we had used 2GB as an input of data file size, and 64 MB as the default data block size. We performed a total of 5 execution cycles, each execution cycle contains 10 jobs, the data files input is different across the executed jobs. Finally, the average execution time of each job is reported for evaluation.

As shown in Figure 8 and Figure 9, our DPP algorithm is compared to three data placement cases, the native Hadoop strategy, all data blocks are local which is the best case, and all data blocks are not local as the worst case.

In the best case, no data blocks are transferred, inversely the Hadoop worst case in term of data distribution occurs when all the data block have to be transferred to be processed. As explained in our algorithm the data blocks must be allocated based on the computing ratio of each node. Therefore, Table 6 values are considered in our experiment. We consider data input for the WordCount job, as 17 data blocks in HDFS, ratios of Table 6 are 4.5, 2, 1, 1. Then, the allocation through our DPP algorithm, is as follow, VM1 is assigned 9 data blocks, VM2 is assigned 4 data blocks, VM3 and VM4 are both assigned 2 data blocks. The worst case is when all of the data blocks are on the same slowest node, then other nodes executing tasks have to wait until data blocks are completely transferred from the slowest node, that is handling all data blocks, in our case VM3 or VM4.

Figure 8 shows how the WordCount job execution time is impacted by different type of data placement. We found that the proposed algorithm of data placement is operating optimally compared to the worst case. As our algorithm is based on previous job execution to evaluate the computing ratio, the first job execution might result in slightly delays as of using the native Hadoop strategy. This can be easily adjusted once the job execution starts on the cluster. Therefore, we are getting nearest average execution time based on the implemented algorithm compared to the best case. Figure 10 shows that our algorithm is able to reduce execution time compared to the default Hadoop strategy by 14.33%. Figure 10 shows that our algorithm can improve the execution time compared to the worst case by approximately 23.8%.

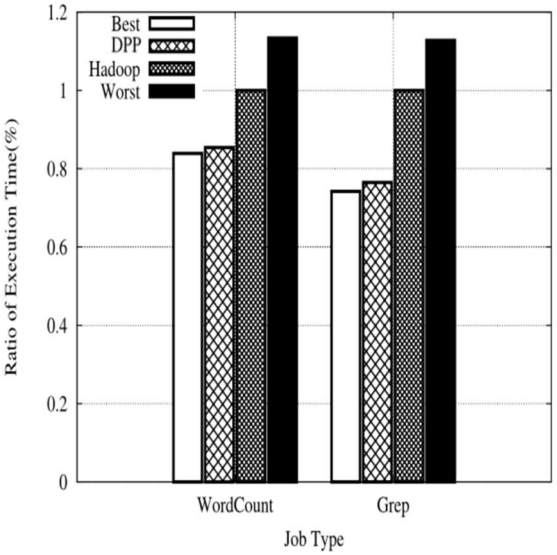

Figure 10: Comparison with Hadoop default strategy - Job average execution time Ratio

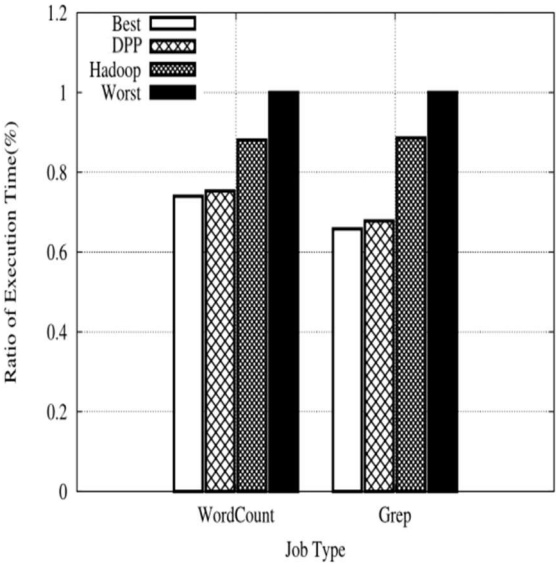

Figure 21: Comparison with worst case - Job average execution time Ratio

For the Grep job evaluation, Figure 11 shows that our algorithm is able to reduce execution time compared to the default Hadoop strategy by 23%. Figure 11 shows that our algorithm can improve the execution time compared to the worst case by approximately 32%.

6. SUMMARY

In this paper, we proposed a Data Placement Policy (DPP) for map tasks of data locality to allocate data blocks. The Hadoop default data placement strategy is assumed to be applied in a homogeneous environment. In a homogeneous cluster, the Hadoop strategy can make full use of the resources of each node. However, in a heterogeneous environment, a produces load imbalance creates the necessity to spend additional overhead. The proposed DPP algorithm is based on the different computing capacities of nodes to allocate data blocks, thereby improving data locality and reducing the additional overhead to enhance Hadoop performance. Finally, in the experiments, for two types of applications, WordCount and Grep, the execution time of the DPP compared with the Hadoop default policy was improved. Regarding WordCount, the DPP is improved by up to 24.7%, with an average improvement of 14.5%. Regarding Grep, the DPP is improved by up to 32.1%, with an average improvement of 23.5%. In the future, we will focus on other types of jobs to improve Hadoop performance such Terasoft and Pi benchmark.

Also, with the importance of AI and machine learning, we will improve this work with advanced algorithms and provide some answers, focusing on the following points:

Which of the machine learning algorithms: logistic regression, RNN, decision tree or clustering is better suited to address the improve Data Placement?

Which of the machine learning algorithms: logistic regression, RNN, decision tree or clustering is better suited to address the problem of Data Placement,

REFERENCES:

[1] Y. Gao and S. Zheng, "A Metadata Access Strategy of Learning Resources Based on HDFS," in proceeding International Conference on Image Analysis and Signal Processing (IASP), pp. 620—622, 2011.

[2] J. Xie, S. Yin, et al. “Improving MapReduce performance through data placement in heterogeneous Hadoop clusters”, In 2010 IEEE International Symposium on Parallel & Distributed

[3] G. Mackey; S. Sehrish; J. Wang. Improving metadata management for small files in HDFS. IEEE International Conference on Cluster Computing and Workshops (CLUSTR). 2009. pp.1-4.

[4] C. Vorapongkitipun; N. Nupairoj. Improving performance of small-file accessing in Hadoop. IEEE International Conference on Computer Science and Software Engineering (JCSSE). 2014. pp.200-205.

[5] A. Patel, Mayuri A. Mehta “A Novel Approach for Efficient Handling of Small Files in HDFS”. 2015. Pp.1-14

[6] T. White, Hadoop: The Definitive Guide, 4th ed. O’Reilly, 2015.

[7] C. Jin, R. Kang, and R. Li, “VTB-RTRRP: Variable Threshold Based Response Time Reliability Real-Time Prediction,” IEEE Access, vol. 6, pp. 60–71, 2018.

[8] C.-H. Chen, J.-W. Lin, and S.-Y. Kuo, “MapReduce Scheduling for Deadline Constrained Jobs in Heterogeneous Cloud Computing Systems,” IEEE Trans. Cloud Comput., vol. 6, no. 1, pp. 127–140, Jan 2018.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” in Proc.IEEE 26th Symp. MSST, Washing-ton, DC, USA, 2010, pp. 1–10.

[10] R. M. Karp, “Reducibility Among Combinatorial Problems,” in Proc. Symp. the Complexity of Computer Computations. Springer US, 1972, pp. 85–103.

[11] K. Zhang, “A Constrained Edit Distance between Unordered Labeled Trees,” Algorithmica, vol. 15, no. 3, pp. 205–222, Mar. 1996.

[12] C.-X. Xu, “A Simple Solution to Maximum Flow at Minimum Cost,” in Proc. 2nd ICIECS, Wuhan, China, 2010, pp. 1–4.

[13] Z. Han, H. Tan, Y. Wang, and J. Zhou, “Channel Selection for Rendezvous with High Link Stability in Cognitive Radio Network,” in Proc. 9th Int. Conf. WASA, Harbin, China, 2014, pp. 494–506.

[14] XenServer — Open Source Server Virtualization. [Online]. Available at:

[15] https://xenserver.org/

[16] D. Cheng, J. Rao, Y. Guo, C. Jiang, and X. Zhou, "Improving Performance of Heterogeneous MapReduce Clusters with Adaptive Task Tuning," IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 3, pp. 774–786, March 2017.

[17] J. A. Issa, "Performance Evaluation and Estimation Model Using Regression Method for Hadoop WordCount," IEEE Access, vol. 3, 2784–2793, 2015.

[18] Tom White. Hadoop The Definitive Guide. O'Reilly, 2009.

[19] P. Dreher, C. Byun, C. Hill, V. Gadepally, B. Kuszmaul, and J. Kepner, "PageRank Pipeline Benchmark Proposal for a Holistic System Benchmark for Big-Data Platforms," in IEEE IPDPSW, May 2016, pp. 929–937.

[20] S. E. Mendili, Y. E. B. E. Idrissi, and N. Hmina, "Benchmarking Study on Smart City Data Analytics," in IEEE Int. CiSt, Oct 2016,841–846.

[21] MathWorks R . [Online]. Available at: https://www.mathworks.com/