International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

# Enhanced Performance of Hadoop Parameters Using Hybrid Meta Heuristics Optimization Techniques

**Nandita Yambem[1], A. N. Nandakumar[2]**

**Submitted:** 27/01/2024 **Revised:** 05/03/2024 **Accepted:** 13/03/2024

**Abstract*:*** With Hadoop becoming the most popular open big data processing platform, various approaches have been proposed to achieve maximal performance gain for big data applications. But the influence of various performance tuning parameters on the overall application speedup is no-linear and it is also dependent on the application/data characteristics. This work models the problem of finding the optimal values for tuning parameters as a search optimization problem and proposes a hybrid meta heuristics solution to problem based on combining grass hopper swarm optimization with bat algorithm. The hybrid algorithm has good exploration and exploitation ability so that the optimal solution is found without getting into local minimal problem.

***Keywords:*** *grass hopper swarm optimization (GSO), Hybrid meta-heuristics, Bat algorithm, mapreduce task.*

## 1. Introduction

Big data analytics has become the engine for growth of many enterprises all over the world. Knowledge extracted from large volume of data collected from various departments of enterprises like marketing, advertisement, development etc. can be used to design the strategies for better positioning of the enterprises and achieve competitive advantage. Among multiple big data analytics platforms, Hadoop is a popular big data processing platform which is open source and base for other processing platforms like Apache Spark. Map/Reduce processing architecture is the core of the Hadoop. The applications are designed in form of Map/Reduce tasks to make maximal use of distributed and parallel computing. The input placed in Hadoop distributed file system (HDFS) is distributed to Map tasks running over multiple nodes and the intermediate data generated by them is written to disk and distributed to Reduce tasks. The results of the Reduce tasks are combined and written to HDFS as final result. Map/Reduce tasks run in multiple instances over multiple nodes to speed up data processing. The performance of Hadoop can be enhanced by fine tuning various configuration parameters like number of threads for Map/Reduce, size of buffers, size of intermediate data, sort factor, spill percent, run time memory size etc. There are multiple configuration parameters and their joint influence on application speed up is not a linear relation. Also exploring all the combination of parameters and their influence of application speed is a combinatory explosion problem. Many solutions have been proposed modeling an influence of single configuration parameter as a linear relationship. But under the constraints of multiple parameters and application/data characteristics, the linear relationship no longer holds. This creates wide negative variance between the expected and actual application Speedup. This work addresses this problem and proposes a solution to find optimal values for the configuration parameters. The problem of finding the optimal values for the configuration parameters is modeled as search optimization problem and hybrid meta-heuristics-based solution is proposed in this work.

A hybrid meta-heuristics solution combining grass hopper swarm optimization (GSO) with bat algorithm is used to find the optimal value for the configuration parameters. The configuration parameters are considered in two categories of application specific and platform specific. Parameters influencing the application speed up are identified in the two categories (application and platform) and a fitness function to maximize the application speed up is designed for it. Hybrid meta heuristics find the values of the configuration parameters with both exploration and exploitation ability so that local minima problem is avoided. Following are the contributions of this work.

1) Configuration parameters influencing the application speedup in Hadoop were identified.

2) Hybrid meta-heuristics combining GSO and bat algorithm to find the optimal value for configuration parameters with goal of application speedup.

The rest of the paper is organized as follows. Section II presents the existing works on Hadoop parameter optimization. Section III presents the proposed hybrid meta-heuristics solution to optimize Hadoop parameters. Section IV presents the results of the proposed solution and its comparison to existing works. Conclusion and future scope of work is presented in Section V.

## 2. Literature Review

Application speedup was increased through improving HDFS efficiency in works of Nicholae et al [1]. Data access concurrency induced computational slowdown was reduced using redundancy. Validating against the Grid 5000 dataset, the solution was found to increase the speed by 5% compared to default Hadoop. Compared to other methods like configuration parameter, the speed up is lower in this approach. Data compression was used as strategy for application

[1] *Visvesvaraya Technological University, Belagavi, Karnataka 590018, India*
*nanditayambem@gmail.com*
[2] *City Engineering College, Bengaluru, Karnataka 560062, India*
*inandakumar53@gmail.com*

speedup in works of Verma et al [2]. The data transfer latency was reduced due to net bandwidth reduction caused by data compression. The solution could not provide more than 5% speed as validated by word counting task in XSEDE platform. Task scheduling strategy was used for application speed up in works of Zaharia et al [3]. In this strategy, the jobs which can affect the response time of other tasks are predicted and prioritized for execution against others. By prioritizing them ahead of others, application slowdown and memory contention are reduced. Lagging job prediction is difficult in heterogeneous environment. Phase level scheduling was used for application speed up in works of Zhang et al [4]. Job is split into multiple stages or phases. Each phase is a fine-grained unit. Scheduling is done phase wise, so that response time of overall job can be increased. This also facilitates introduction of more parallelism. Validating the solution in a 10 node Hadoop cluster setup, the solution is found to increase the speed up by 1.3 times. But without a generalized method for phase like application split up, the solution cannot be used for all kinds of applications. Data locality was used for speeding up the applications in works of Guo et al [5]. Task with higher inter task communication are scheduled to run in same node. By this way, message exchange across nodes is reduced and this is reflected as speedup in applications. But the communication profile of tasks must be known well in advance to achieve maximum benefit in this method. Intermediate data generated by map jobs were compressed to improve the speedup in works of Crume et al [6]. Compression of intermediate data reduced the data shuffling computation overhead. Authors also improved the compression efficiency without any compression loss. Through testing with Hadoop clusters, authors found a speedup of 6%. The scheme works well only for certain datasets. As a solution to this problem Chen et al [7] proposed an adaptive strategy for intermediate data compression. Heuristics based decision was made to decide when to compress the data based on application performance benchmark. Data shuffling strategy was used
to improve application performance in works of Yu et al [8]. This algorithm made a decision on data shuffling with the goal of minimization of extra duty cycles. Application speed up increased by 10% compared to default Hadoop in this solution. Data compression-based application speed up strategy was proposed by Ruan et al [9]. Through a novel data compression algorithm, authors compressed the intermediate data. The solution was tested against word count task in Hadoop cluster and the method was found to provide at least 5% speedup. Moise et al [10] improved the efficiency of intermediate data management through concurrency. This optimization reduced the overall data fetch time and increased the application speed up. But the method is not scalable for large clusters. By optimizing the in-memory management, Veiga et al [11] increased the application speedup.
Intermediate data were managed effectively and as result application speed up increased two times. The approach also increased the memory resource cost. Configuration parameter optimization was used for application speedup in works of Chen et al [12]. Authors identified the parameters to be fine-tuned for CPU and IO intensive tasks. But the parameter values are selected trial and error without any guideline for value selection. Application collocation was used as strategy for application speed up in works of Malik et al [13]. Collocation reduced the inter node communication and memory latency due to it. This resulted in application speed up. Authors found the proposed solution is able to increase the speed up by 8%. Application speed up for failed tasks was realized by C.K et al [14]. Authors applied check pointing to remember the failed points and execution continued from those points. Hadoop configuration parameter tuning using genetic algorithm was explored in works of Liao et al [15]. Though multiple parameters were considered, the joint influence of multiple parameters was not considered in this work. Memory management strategies for application speed up were explored in works of Bhaskar et al [16]. Based on past history of application, the memory profile for application is designed. Pre-allocation of memory is done based on the memory profile. Resources were held without utilization and this affected the overall throughput. Gradient algorithm was used for Hadoop configuration parameter tuning in works on Kumar et al [17]. Gradient algorithm fine-tuned the parameters with goal of application speedup. But the approach is specific to application and not generic. Lee et al [21] used data locality for application speedup. Two different data locality algorithms based on block and keyword was introduced to achieve maximal performance for map and reduce jobs. But the method is not scalable for large clusters. Eldouh et al [22] integrated data locality along with reduced data shuffling to increase the application speedup. Term frequency features are extracted from texts and they are grouped using K means clustering algorithms. Data belonging to same clusters are maintained in same node. Though speed
up was increased by 40%, the solution worked only for specific applications. From the survey, it could be seen that among the approaches for configuration parameter tuning, multi parameters tuning with goal of application speed up is not considered in any of existing works. Though some works on tuning the number of reduce tasks or memory was available, they are not adaptive to application/data characteristics. This work addresses this research gap and proposes a solution for application speedup in Hadoop based on optimization of multiple configuration parameters adaptive to application/ data characteristics.

## 3. Hybrid Meta-Heuristics Optimization

The architecture of the proposed solution is given in Figure 1. The application and platform specific configuration parameters with stronger correlation to application speed up are identified. The optimal values for these configuration parameters are found using hybrid meta-heuristics optimization. The optimal values are set onto application and platform to achieve higher speed-up. Following are the parameters considered for optimization in this work.

The application speedup is measured in terms of job completion time. To model the relationship between the configuration variables and the job completion time, a dry run is conducted with various configuration values and the measured job completion time. The optimal values for the configuration parameters (P1-P10) to achiever lower job completion time are found using hybrid meta-heuristics optimization combining Bat algorithm with GSO. A hybrid equilibrium is maintained between exploration (local optima) and exploitation (global optima) by adopting bat and GSO combination Bat performs well in local region search with fine exploitation and GSO offers faster convergence in terms of exploration.

The optimal values for the configuration parameters (P1-P10) to achiever lower job completion time are found using hybrid meta-heuristics optimization combining Bat algorithm with GSO. A hybrid equilibrium is maintained between exploration (local optima) and exploitation (global optima) by adopting bat and GSO combination. Bat performs well in local region search with fine exploitation and GSO offers faster convergence in terms of

exploration.

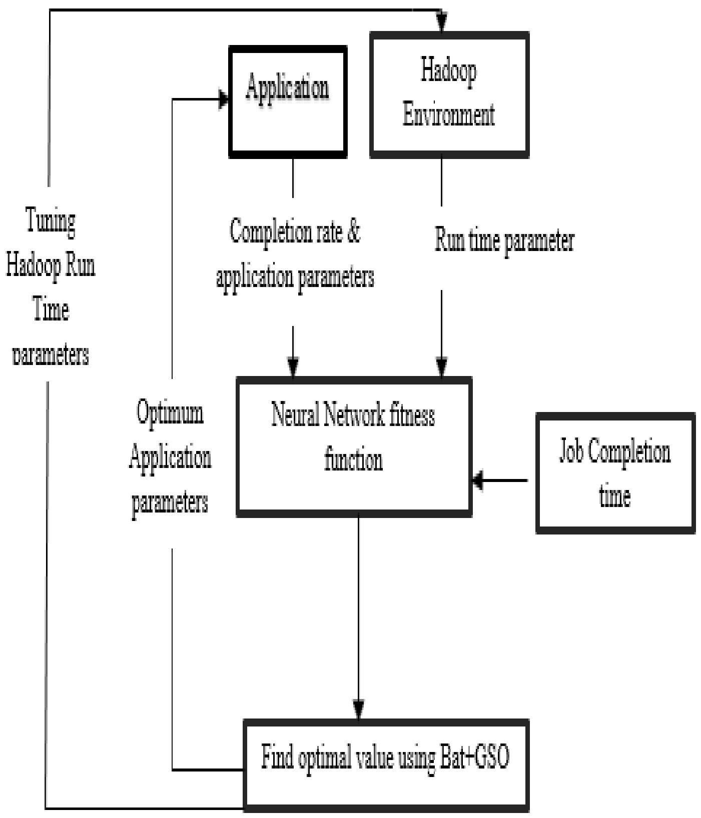
Application
Hadoop Environment
Tuning Hadoop Run Time parameters
Completion rate & application parameters
Run time parameter
Optimum Application parameters
Neural Network fitness function
Job Completion time
Find optimal value using Bat+GSO

**Fig.1.** Architecture of proposed hybrid Meta heuristics solution

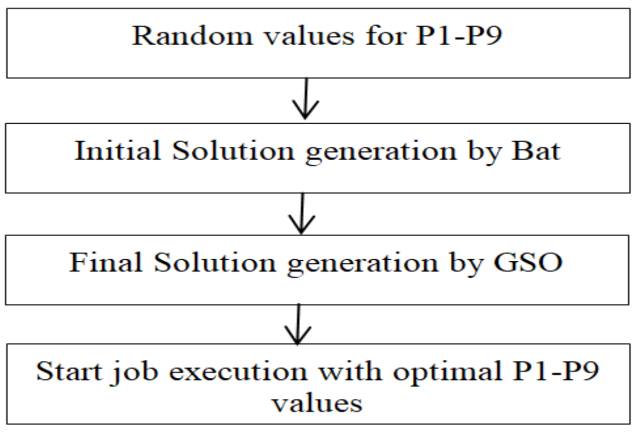
Random values for P1-P9
Initial Solution generation by Bat
Final Solution generation by GSO
Start job execution with optimal P1-P9 values

**Fig.2.** Flow of parameter optimization

Bat algorithm is a bio inspired search optimization algorithm based on the bio-sonar characteristics of bats. Bats use a type of sonar called echolocation to detect prey. They fly from a position $x_i$ with a random velocity $v_i$ with frequency f and loudness $A_o$ in search of pray. They adjust the wavelength of their emitted pulses and rate of pulse emission depending on their proximity to their pray. The location, velocity and pulse frequency of a bat is updated over successive iterations (t) as below

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{1}$$

$$Fv_i^t = Fv_i^{t-1} + \left(Fx_i^{t-1} - Fx_*\right)f_i \tag{2}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{3}$$

$\beta$is the random variable. Each bat is initially allocated a random wavelength between $[f_{min}, f_{max}]$. The frequency is fine tuned for a better exploration of the pray in the search space and for diversification ability.

This is achieved by combining optimization algorithms with complementary properties of strong exploitation and diversification. By this way a near optimal solution can be obtained with a faster convergence rate.

**Table 1**. OPTIMIZATION PARAMETERS

| ID | Variables | Default values |
|---|---|---|
| P1 | io. sort.factor | 10 |
| P2 | io. sort.mb | 100 |
| P3 | io.sort.spill.percent | 0.80 |
| P4 | mapred.reduce.tasks | 1 |
| P5 | mapreduce.tasktracker.map.tasks. maximum | 2 |
| P6 | mapreduce.tasktracker.reduce.tas ks.maximum | 2 |
| P7 | mapred.child.java.opts | 200 |
| P8 | mapreduce.reduce.shuffle.input.b uffer.percent | 0.70 |
| P9 | mapred.inmem.merge.threshold | 1000 |
| P1 0 | Input data size (number of samples/MB) | Application dependent parameter |

GSO is a recent swarm intelligence algorithm proposed in works of Saremi et al [19]. This algorithm is based on the grasshopper's foraging and swarming behavior. Grasshopper is an agricultural pest whose life cycle has two stage nymph and adulthood. In nymph stage, the grasshoppers move in small steps with less movement. In adulthood stage, grasshoppers make long rage movements and the movements are abrupt. GSO algorithm has two phases (i) intensification and (ii) diversification which are based on the movement pattern of grasshoppers in nymph and adulthood stage. Mathematically, GSO represents the swarming behavior of grasshoppers in terms of their social interaction ($S_i$), gravitational force ($G_i$) and wind advection ($A_i$) as

$$P_i = S_i + G_i + A_i \tag{4}$$

Where $P_i$is ith grasshopper's position. $S_i$ is calculated for N grasshoppers separated by a Euclidean distance ($d_{ij}$) with a social force s as

$$S_i=\sum_{\substack{j=1\\ j\neq i}}^{N} s(d_{ij})\widetilde{d_{ij}} \tag{5}$$

The social force is represented in terms of attraction intensity ($f$) and attration length ($l$) as

$$s(r) = f\exp^{\frac{-r}{l}} - \exp^{-r} \tag{6}$$

Attraction and repulsion are the two themes based on which social interaction is measured. For a distance in range of 0 to 15, attraction is felt in range of 2.07 to 4 and repulsion is felt in range of 0 to 2.07. At the distance of 2.07, a comfort zone is realized where there is neither attraction nor distraction.

The gravity force $G_i$in equation (4) is calculated in terms of distance unit vector to center of earth ($\hat{e}_g$) and gravitational constant (g) as

$$G_i = -g\hat{e}_g \tag{7}$$

The wind advection $A_i$ in equation (4) is calculated in terms of distance unit vector to wind direction ($\hat{e}_w$) and drift constant (u) is given by

$$A_i = u\hat{e}_w \tag{8}$$

Fitting each of the variables, the equation 1 is modified with upper bounds ($ub_d$) and lower bounds ($lb_d$) in the the d-th dimension and given as equation 9.

$$P_i^d = c\left(\sum_{\substack{j=1\\ j\neq i}}^{N} c\frac{ub_d - lb_d}{2}\right)s(|P_j^d - P_i^d|)\frac{P_j-P_i}{d_{ij}}) + \hat{T}_d$$

(9)

$\hat{T}_d$ is the best solution found so far in the d-th dimension space. The parameter c is similar to inertia weight ω in PSO. This parameter controls the grasshopper's movement around food (target) and provides a fine balance between diversification and intensification. The parameter $c$ is calculated as

$$c = c_{max} - t\frac{c_{max}-c_{min}}{t_{max}} \tag{10}$$

With the maximum value for $c$ represented as $c_{max}$ and minimum value for $c$ represented as $c_{min}$. The position is updated for every iteration ($t$) for a maximum number of iterations ($t_{max}$)

Grasshopper position is updated every iteration based on both local and global best solution. The iteration is stopped when they are no change in position of grasshopper.

Use of global best position prevents from getting trapped into local optimum.

The pseudo code of grass hopper optimization algorithm is given below

**Algorithm 1: GOA Optimization**

```
A. Random generation of initial population for n grasshoppers Pi
B. Initialize Cmin, Cmax, and a maximum number of iterationTmax
C. Evaluate the fitness f(Pi) of each grasshopper Pi
D. B= The best solution
E. While (t<tmax) do
F. Update c1 and c2
G. For i=1 to N, for all N grasshoppers in the population,
H.  do
I. Distance between grasshoppers normalized in range of 1 to 4.
J. Update position using equation (12)
K. Rectify outlier and normalize grasshoppers position
L. end for
M. Update B with best solution so far
N. t=t+1
O.  end while
P. Return B
```

Given a job completion time target ($jtct$) and input data size ($P10$), the fitness function for maximization can be framed as

$$f = \frac{1}{(jtc-jtct)+1} \tag{11}$$

Where $jtc$ is the actual execution time. Since the relation between parameters (P1-P10) and $jtc$ cannot be modeled as linear, a neural network function is used to model the relationship. The neural network function is given as

$$jtc = \frac{1}{1+e^{\sum_1^9 W_i P_i + b}} \tag{12}$$

Where the $W_i$ is the weights of the neural network and $b$ is the bias. The weights and bias are found by training a 3-layer feed forward neural network of following configuration

**Table 2**. NEURAL NETWORK CONFIGURATION

| **Parameters** | **Values** |
|---|---|
| Inputs | P1to P10 |
| Output | $jtc$ |
| Number of layers in neural network | 3 |
| Number of neurons is layer 1 | 10 |
| Number of neurons in layer 2 | 21 |
| Number of neurons in layer 3 | 1 |
| Activation function | Sigmoid |

The optimization process is given in Figure 2. The optimization process starts with initial solution generation by Bat algorithm followed by solution refinement using GSO. Bat algorithm starts with a random values for P1-P9 (P10 is input by user) and attempts to find the best values of P1-P9 by optimizing the fitness function given in Eq. 11. GSO starts with initial solution given by Bat algorithm and attempts to find the best values of P1-P9 by maximizing the fitness function given in Eq. 11.

## 4. Results

The performance of the proposed solution is tested against PUMA dataset [18] in Hadoop environment. The performance of the proposed solution is compared for word count and k-means for different volume of datasets. The performance of the proposed solution is compared against optimization scheme proposed by Chen et al [12], Enhanced Parallel Detrended Fluctuation Analysis solution proposed by Khan et al [20] and default parameters of Hadoop.

**Table 3.** COMPARISION OF DIFFERENCE BETWEEN EXPECTED AND ACTUAL EXECUTION TIME FOR WORD COUNT APPLICATION

| **Data volume (MB)** | **Proposed** | **Chen et al** | **Khan et al** | **Default Hadoop** |
|---|---|---|---|---|
| 128 | 2 | 4 | 3 | 8 |
| 256 | 4 | 7 | 5 | 11 |
| 512 | 5 | 8 | 6 | 12 |
| 1024 | 5 | 8 | 7 | 14 |
| 2048 | 6 | 9 | 7 | 15 |
| Average | 4.4 | 7.2 | 5.6 | 12 |

**Table 4.** COMPARISION OF DIFFERENCE BETWEEN EXPECTED AND ACTUAL EXECUTION TIME FOR K-MEANS CLUSTERING APPLICATION

| Data volume (MB) | Proposed | Chen et al | Khan et al | Default Hadoop |
|---|---|---|---|---|
| 128 | 3 | 4.5 | 4 | 9 |
| 256 | 4.2 | 7.2 | 5.5 | 11.8 |
| 512 | 5.7 | 8.4 | 6.4 | 12.1 |
| 1024 | 6.1 | 9.3 | 7.6 | 14.7 |
| 2048 | 7 | 10.1 | 8.2 | 15.6 |
| Average | 5.2 | 7.9 | 6.34 | 12.64 |

The percentage difference between expected and actual execution time is at least 50% lower compared to Chen et al, 21% lower compared to Khan et al and more than 100% lower compared to default Hadoop parameter settings. The difference has reduced in proposed solution due to two stages of finding optimal parameters without getting into local minima problem. Initial solution is found by Bat and further refined by GSO. The refined solution provides application speed close to expected job completion time.

**Table 5.** COMPARISION OF EXECUTION TIME FOR K-MEANS CLUSTERING APPLICATION

| Data volume (MB) | Proposed | Chen et al | Khan et al | Default Hadoop |
|---|---|---|---|---|
| 128 | 37 | 60 | 46 | 100 |
| 256 | 38 | 62 | 47 | 105 |
| 512 | 39 | 63 | 49 | 107 |
| 1024 | 41 | 65 | 51 | 109 |
| 2048 | 42 | 67 | 52 | 111 |
| Average | 39.4 | 63.4 | 49 | 106.4 |

The average execution in proposed solution is at least 60% lower compared to Chen et al and 24% lower compared to
Khan et al. It is 1.7 times lower compared to Default Hadoop.

**Table 6.** COMPARISION OF EXECUTION TIME FOR K-MEANS CLUSTERING APPLICATION

| Data volume (MB) | Proposed | Chen et al | Khan et al | Default Hadoop |
|---|---|---|---|---|
| 128 | 46 | 70 | 53 | 110 |
| 256 | 49 | 72 | 55 | 115 |
| 512 | 51 | 74 | 57 | 117 |
| 1024 | 54 | 76 | 60 | 119 |
| 2048 | 56 | 77 | 61 | 121 |
| Average | 51.2 | 73.8 | 57.2 | 116.4 |

The average execution time in proposed solution is at least 44% lower compared to Chen et al and 11% lower compared to Khan et al. The execution time has reduced in proposed solution due to speedup caused by optimal configuration parameters.

## 5. Conclusion

A hybrid meta heuristics solution combing bat algorithm with GSO is proposed in this for optimizing configuration parameters. Optimal values for the parameters were found in two stages of initial solution by bat and further refinement by GSO. Combining GSO with Bat provided better exploration and exploitation ability in optimization process. Through performance analysis with two different applications for various data sizes, the proposed solution is found to provide better speed up and close to target execution time. The execution time is at least 11% lower compared to existing works and proposed solution has at least 21% reduction in deviation between actual and expected execution time.

## References

[1] B. Nicolae, D. Moise, G. Antoniu, and al. BlobSeer: Bringing high throughput under heavy concurrency to Hadoop Map/Reduce applications. In Procs of the 24th IPDPS 2010, 2010. In press

[2] A. Verma, L. Cherkasova, and R. Campbell. Resource Provisioning Framework for MapReduce Jobs with Performance Goals. ACM/IFIP/USENIX Middleware, pages 165–186, 2011.

[3] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In USENIX Symposium on Operating Systems Design and Implementation (OSDI), volume 8, page 7, 2008

[4] Qi Zhang, "PRISM: Fine-Grained Resource-Aware Scheduling for MapReduce", 2015 IEEE

[5] ZhenhuaGuo, Geoffrey Fox, Mo Zhou, Investigation of Data Locality in MapReduce, Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), p.419-426, May 13-16, 2012 [doi>10.1109/CCGrid.2012.42]

[6] Adam Crume, Joe Buck, Carlos Maltzahn, Scott Brandt, Compressing Intermediate Keys between Mappers and Reducers in SciHadoop, Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, p.7-12, November 10-16, 2012 [doi>10.1109/SC.Companion.2012.12]

[7] Y. Chen, A. Ganapathi, and R. H. Katz, "To compress or not to compress-compute vs. io tradeoffs for mapreduce energy efficiency," in Proceedings of the first ACM SIGCOMM workshop on Green networking. ACM, 2010, pp. 23–28.

[8] W. Yu, Y. Wang, X. Que, and C. Xu, "Virtual shuffling for efficient data movement in mapreduce," IEEE Transactions on Computers, vol. 64, no. 2, pp. 556–568, 2015

[9] G. Ruan, H. Zhang, and B. Plale, "Exploiting mapreduce and data compression for data-intensive applications," in Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery. ACM, 2013, pp. 1–8.

[10] D. Moise, T.-T.-L. Trieu, L. Boug´e, and G. Antoniu, "Optimizing intermediate data management in mapreduce computations," in Proceedings of the first international workshop on cloud computing platforms. ACM, 2011, pp. 1–7.

[11] Veiga, Jorge & Expósito, Roberto &Taboada, Guillermo & Touriño, Juan. (2018). Enhancing in-memory efficiency for MapReduce-based data processing. Journal of Parallel and Distributed Computing. 120. 10.1016/j.jpdc.2018.04.001.

[12] Chen, Xiang & Liang, Yi & Li, Guang-Rui& Chen, Cheng & Liu, Si-Yu. (2017). Optimizing Performance of Hadoop with Parameter Tuning. ITM Web of Conferences. 12. 03040. 10.1051/itmconf/20171203040.

[13] Maria Malik, Hassan Ghasemzadeh, Tinoosh Mohsenin, Rosario Cammarota, Liang Zhao, AvestaSasan, Houman Homayoun, and SetarehRafatirad. 2019. ECoST: Energy-Efficient Co-Locating and Self-Tuning MapReduce Applications. In Proceedings of the 48th International Conference on Parallel Processing (ICPP 2019).

[14] C, K. and X, A. (2020), Task failure resilience technique for improving the performance of MapReduce in Hadoop. ETRI Journal, 42: 748-760.

[15] Liao G., Datta K., Willke T.L. (2013) Gunther: Search-Based Auto-Tuning of MapReduce. In: Wolf F., Mohr B., an Mey D. (eds) Euro-Par 2013 Parallel Processing. Euro-Par 2013. Lecture Notes in Computer Science, vol 8097. Springer, Berlin, Heidelberg.

[16] Bhaskar, Archana&Ranjan, Rajeev. (2019). Optimized memory model for hadoop map reduce framework. International Journal of Electrical and Computer Engineering (IJECE). 9. 4396. 10.11591/ijece.v9i5.pp4396-4407.

[17] S. Kumar, S. Padakandla, L. Chandrashekar, P. Parihar, K. Gopinath and S. Bhatnagar, "Scalable Performance Tuning of Hadoop MapReduce: A Noisy Gradient Approach," 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, 2017, pp. 375-382, doi: 10.1109/CLOUD.2017.55

[18] Engineering.purdue.edu/~puma/datasets.htm

[19] S. Saremi, S. Mirjalili, and A. Lewis, ``Grasshopper optimisation algorithm: Theory and application," Adv. Eng. Softw., vol. 105, pp. 30_47, Mar. 2017

[20] Mukhtaj Khan, Zhengwen Huang, Maozhen Li, Gareth A. Taylor, Phillip M. Ashton, Mushtaq Khan, "Optimizing Hadoop Performance for Big Data Analytics in Smart Grid", Mathematical Problems in Engineering, vol. 2017, Article ID 2198262, 11 pages, 2017.

[21] S. Lee, J.-Y. Jo, and Y. Kim, ''Hadoop performance analysis model with deep data locality,'' Information, vol. 10, no. 7, p. 222, Jun. 2019

[22] A. Eldouh, H. Elkadi, and M. Khafagy, ''Reducing data shuffling and improving MapReduce performance using enhanced data locality,'' in Proc. IASTEM Int. Conf., 2019, pp. 5