NVEO
Natural Volatiles & Essential Oils

An Optimized IWOA and MECC based MapReduce Framework for Big data

Mr. Vishal Kumar[1], Prof. Kamaljit Singh Saini[2]

[1]Assistant Professor, Department of Computer Science and Engineering, Chandigarh University, Punjab.
Email: vishalkm77@gmail.com.
[2]Professor, Department of Computer Science Engineering, Chandigarh University, Punjab.
Email: kamaljit.cse@cumail.in.

Abstract

New participants in the cloud ecosystem are large MapReduce clusters, often managing petabytes of unstructured and semi-structured data. Increasing the use of these MapReduce clusters is a big problem. This work considers a subset of production workloads that consist of independent MapReduce jobs. The order in which these activities are executed has a considerable impact on overall processing time and cluster resource use, according to our findings. Our goal is to automate the creation of work plans that reduce the time it takes to complete a series of MapReduce processes (span creation).This work present Balanced Pools, a novel abstraction framework and algorithm for building an optimum task schedule based on the performance features of MapReduce processes in a given workload. Simulations of a realistic workload show that simply processing the jobs in the appropriate order can enhance make span by 15% to 38%. The data supplied to the cloud is the responsibility of the Cloud Services Provider (CSP). The main disincentive to using cloud services is the risk of strangers seeing stored data and using sensitive raw data. As a result, Data Security (DS) and privacy are the primary concerns that obstruct the adoption of the CC. There are numerous strategies for ensuring data secrecy, but none of them totally protect the data. To overcome these shortcomings, this paper provides a Modified Elliptic Curve Cryptography (MECC) technique to protect your data from hostile attacks

Keywords: Data Security, Elliptical Curve Cryptography, Cloud Services, Minimized Makespan, MapReduce.

Introduction

The term "Big Data" refers to a collection of massive datasets that can't be processed with traditional computer methods. The administration of massive amounts of varied data formats (structured and unstructured data) that are too large to process using traditional databases and software technologies is referred to as big data.

Five essential terms define the properties of big data:

a) **Variability:** Data who's meaning changes over time is referred to as "variability".

b) **Velocity:** This relates to how quickly data is created and processed in order to satisfy demand.

c) **Volume:** The storage of transactional data, live streaming data, and data acquired by sensors, among other things, all contribute to the growth of volume.

d) **Complexity:** When data is gathered from a variety of sources, the data's complexity must be taken into account.

e) **Variety:** Standard databases, text documents, emails, video, audio, transactions, and more forms are increasingly used to store data.

There is use of big data to analyze massive data sets in order to discover economic trends, determine research quality, anticipate disease transmission, and combat crime, among other things [1]. Retail, manufacturing, healthcare, pharmaceuticals, aviation, telecommunications, Banking, financial services, energy, life science, and many other industries use big data. Big data is defined by three Vs, according to industrial data analyst Doug Laney: volume, velocity, and variety. The five V's is shown in fig. 1.

Figure 1. Big data multi-V's model.

Big data analysis is divided into four parts, referred to as the four A's: action, analysis, acquisition, and assembly. By conserving time and lowering processing expenses, problems of high computational cost, speed and efficiency can be tackled. We need to lower the volume of data being processed, which we may do by minimizing the feature of big data processing. A Feature Selection (FS) strategy can be used to reduce the amount of data that is stored. FS has an impact on performance and allows for quicker decisions. FS determines which characteristics should be used to improve performance [2].

Whale Optimization Algorithm (WOA)

The Whale Optimization Algorithm (WOA) is swarm-based algorithms and this algorithm is inspired by a humpback whale's hunting method of using a bubble net. WOA is commonly divided into two categories: 1) exploration and 2) exploitation. WOA was regarded to have a lot of potential for identifying the optimum global answer rapidly while avoiding local optima. This is owing to their outstanding ability to move smoothly from exploitation to exploration. According to prior research, it can handle a variety of real-world difficulties, such as optimal scaling of renewable resources for loss reduction in feature selection, data clustering, skeletal structure sizing optimization and distribution systems,. Because WOA's advantages are clear, many academics prefer it to other algorithms when dealing with complicated optimization problems.

WOA, on the other hand, has a similar fault in that it is slow when compared to other SI algorithms in terms of convergence rate. When applied to a large-scale assignment, WOA performance suffers due to the requirement for a significant computer effort. In the problem of water resource allocation optimization, it highlighted a WOA fault. As the number of repetitions grows, WOA has a slow convergence rate. They started with a 180-time set and increased it by 2000 times, but they were unable to achieve the expected convergence rate. Algorithm 1 fully represents the WOA pseudocode.

Algorithm 1: The WOA algorithm.

Initialize the whales population X_i (i = 1, 2,…,n)
Calculate the fitness of each search agent
X^* = the best search agent
while t < maximum number of iterations do
for each search agent do
Update a, A, C, l, and p.
if (p < 0.5) then
if (|A| < 1) then

Update the position of the current whale with a random position in the neighborhood of the best solution in the current swarm.
else if (|A| ≥ 1) then
Select a random agent (Xrand) from current swarm.
Update the position of the current whale with a random location in the neighborhood of the random agent X_{rand}.
end if
else if (p ≥ 0.5) then
Update the position of the current whale according to the bubble-net technique.
end if
end for
Check if any search agent goes beyond the search space and amend it
Calculate the fitness of each search agent
Update X^* if there is a better solution
t = t + 1
end while
Return X^*

MapReduce

In a distributed setting, MapReduce is a programming model for constructing scalable, fault-tolerant parallel programmes. In this paradigm, the Map and Reduce functions combine to provide a divide-and-conquer component. The work is distributed among a cluster of heterogeneous commodity processors in MapReduce to achieve parallelization.

On the other hand, the Apache Hadoop project is a Java-based open-source MapReduce implementation. Academics have long used Apache Hadoop because of its MapReduce capabilities, which make it simple to employ in areas like, bioinformatics, machine learning, and text mining. In addition, when it comes to spreading EAs, MapReduce gets a lot of attention. The builder of a Map and Reduce structure just needs to develop the core algorithm. This will motivate them to concentrate entirely on the algorithm rather than on the management of distributed implementations.

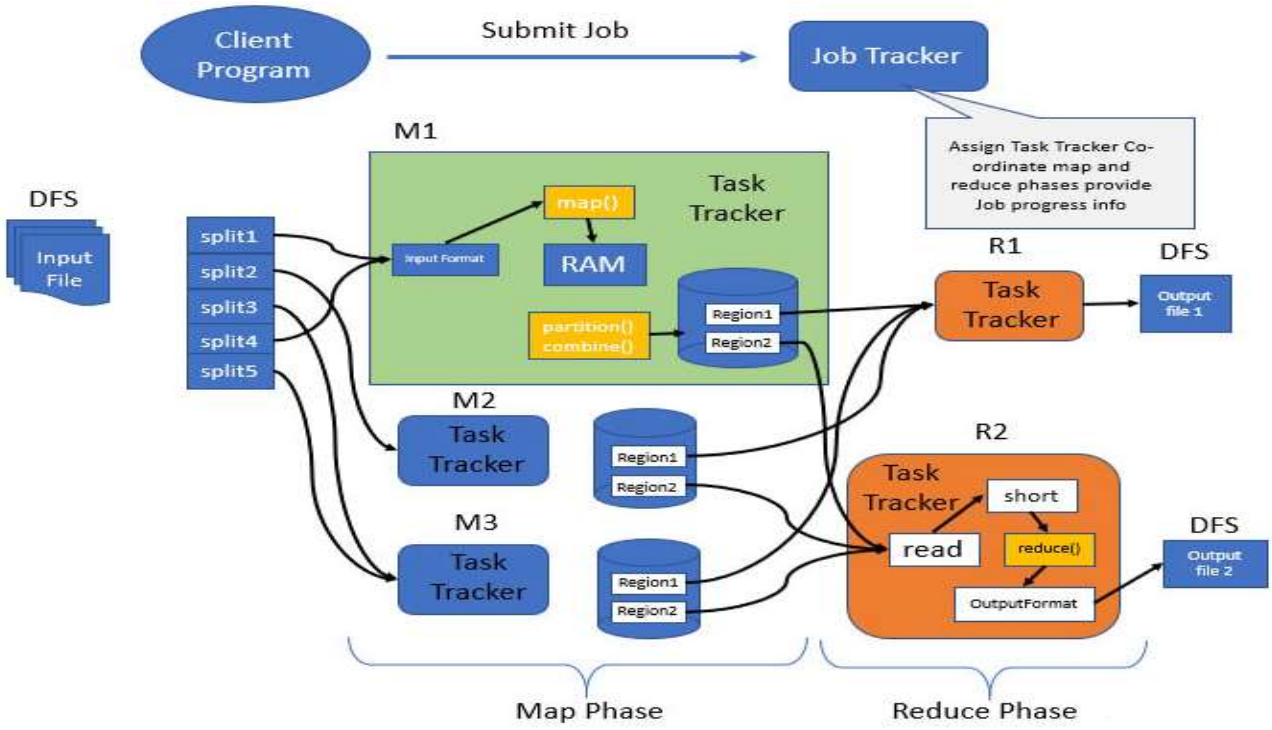


Figure 2. Hadoop Map Reduce Architecture

Figure 2 depicts the map reduction architecture, which comprises mostly of two processing phases. The Map stage is the first, and the Reduce stage is the second. The actual MR process takes happening in task tracker. Between the map and decreasing stages, there will be an intermediate phase. The intermediate phase will conduct shuffle and sort operations on the mapper output data. On the local file system, the intermediate data will be preserved.

Literature Review:

Traganitis et al. [5] presented two types of kernel-based K-means clustering for massive data clustering are sketch and validate. Batch processing was used by the former to improve calculation performance, whilst sequential processing was used by the latter. Despite the algorithm's ability to cluster data effectively, it is hampered by its failure to consider the defined MRF.

Vadivel and Raghunath [7] developed a hierarchical clustering algorithm based on the MRF to manage enormous data. The clustering task used feature selection based on co-occurrence, which was enabled by a distributed design that shuffled mapper results according to queues. Even though the approach takes less time to compute, the merging stage for hierarchical clustering takes a lengthy time to compute.

Fries et al. [6] Researchers investigated solutions for large-scale data sets in high-dimensional domains using the state-of-the-art projected clustering method P3C. The authors showed that the original design of the method was inappropriate for handling large datasets. As a result, they devised the P3C+-MR method, a MapReduce-based implementation that incorporates the necessary adjustments to the basic clustering concept. The MRF provides improved scalability, however the curse of dimensionality makes the procedure difficult.

Akthar et al. [4] The K-means clustering technique was improved for large data clustering by selecting the best centers based on the dimensionality of the data. The method was altered to include the idea of choosing the best "K" data points in highly populated areas as the starting points and omitting data points outside the selected areas from the final cluster computation. Despite the fact that the algorithm generates

better results, it has a few flaws, which include the following: I the algorithm's conclusions are only compared to one other algorithm, which is insufficient for effective performance comparison.

Hosseini & Kiani et al [8] developed an approach for evaluating huge microarray datasets using clustering methods. They proposed a Hadoop MapReduce-based Fuzzy Weighted Clustering algorithm (FWCMR). Several clustering validity indices were used to test the FWCMR approach's efficiency on numerous large microarray datasets maintained across scattered nodes.

Objectives:

- To simultaneously plan both Map and Reduce jobs while ensuring timeliness and security.
- To locate the appropriate resources to run a job on a distributed system in such a way that the total execution time, or makespan, is minimized and the work is completed within the security constraints.
- An optimization approach is proposed for minimizing total task completion time (makespan) and improving task execution security.

MapReduce and Job Profiles Model for Improving Productivity

A MapReduce job's processing phases are depicted in Figure 1. The WikiTrends application is used in this example to examine Wikipedia page traffic records that are gathered (and compressed) every hour. In the supplied input dataset, WikiTrends contains 16 map and 16 reduction slots, as shown in Figure 1. As a result, there are 5 map waves (which make up the map stage) and 4 decrease waves in the work execution (that constitute the reduce stage). A considerable percentage of the map stage may be covered by the first shuffle. We seek to reduce stage execution durations by defining task execution time as the sum of complementing, nonoverlapping maps [9].

Take, for example, job J, which is divided into N_R^J reduce and N_M^J map tasks. Let's pretend J is already installed in a Hadoop cluster. Let S_R^J and S_M^J and denote the number of map and reduce slots set aside for job J's future execution. The map stage is made up of a variety of map assignments. When the number of tasks exceeds the number of slots, the task assignment is divided into several rounds, which we refer to as waves. We calculate the average duration Mavg and the maximum time Mmax based on the distribution of map task durations in the previous run. Then, in the future execution with S_M^J map slots, the lower and upper bounds on the time of the complete map stage (denoted as T_M^{low} and T_M^{up} respectively) are approximated as follows:

$T_M^{low} = N_M^J / S_M^J * M_{avg}$

$T_M^{up} = (N_M^J - 1) / S_M^J * M_{avg} + M_{max}$

The shuffle and reduce phases make up the reduction stage, and their execution time constraints can be calculated identically.

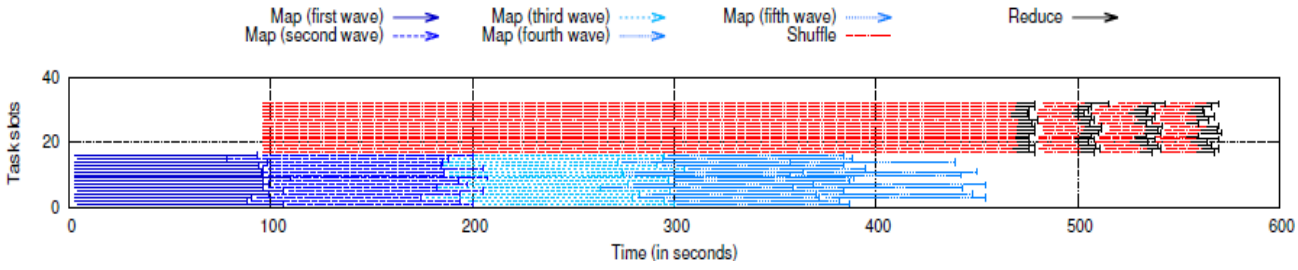


Figure 3. The WikiTrends application was developed on a Hadoop cluster with 16 map and 16 reduce slots.

Problem Definition

Each MapReduce task is assigned a specified number of map and reduce jobs. The quantity of resources (reduction and map slots) assigned to the job determines the job execution time and parameters. Figure 3 depicts the processing of the 71 map and 64 reduce processes in this application using a Hadoop cluster with 16 map and 16 reduce slots. Instead of detailed job execution at the task level, we propose a basic abstraction in which each MapReduce job J_i is described by the lengths of its map and reduction phases mi and r_i, i.e., $J_i = (m_i, r_i)$. The proposed new abstraction is derived using this paradigm $J_i = (m_i, r_i)$. The methods for lowering the total completion time of a batch of MapReduce jobs are explored in this section. The inefficiencies of this abstraction are highlighted, as well as a new heuristic for determining the best scheduling for a set of MapReduce processes. Consider the two (unrelated) MapReduce processes J_1 and J_2, which are both executing in a Hadoop cluster with a FIFO scheduler. Between these jobs, there are no data connections. As a result, once the first job finishes its map stage and begins reduce stage processing, the next job can begin running its map stage using the map resources that have been freed, as shown in Figure 4. The upcoming task's map stage and the previous job's decline stage "overlap."

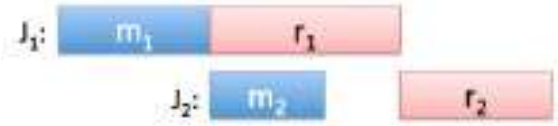


Figure 4. Execution of two MapReduce tasks, J1 and J2, in a pipeline.

We make a fascinating remark about how such jobs are completed. Some of the execution orders may result in inefficient resource utilization and longer processing times. Consider two separate MapReduce tasks that use all of the resources available in a given Hadoop cluster and produce the following reduce and map stage durations: J_2 = (2s, 20s) and J_1 = (20s, 2s). In a Hadoop cluster with a FIFO scheduler, they can be processed in one of two ways:

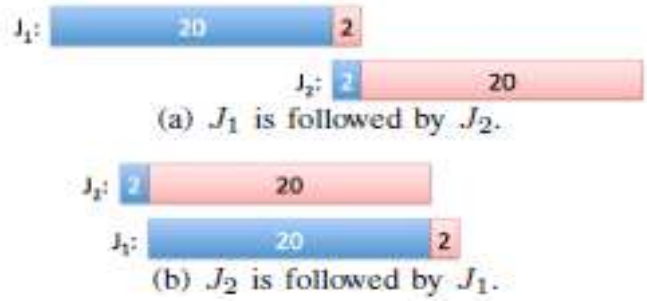


Figure 5. Different job schedules have an impact on overall completion time.

As indicated in Figure 5 (a), J_1 is followed by J_2. J_1's decrease stage overlaps with J_2's map stage, resulting in a 2s overlap. As a result, the total processing time for two jobs is 20s + 2s + 20s = 42s and J_1 appears after J_2 (as seen in Figure 5(b)). As a result, the total makespan is equal to 2s + 20s + 2s = 24s. Let's have a peek at the next issue, J = J_1, J_2, J_3,…. Jn refers to a set of n MapReduce tasks that are data independent. It is need to figure out what order (or schedule) jobs Ji ε J should be completed in so that the overall time to complete the set is as little as possible [11]. Method 1 demonstrates how Johnson's algorithm can be used to create an ideal timetable. The sorting operation dominates the complexity of Johnson's Algorithm, making it O (n log n).

Algorithm 1: Johnson's Procedure

Input: A collection of n MapReduce jobs, J. As previously stated, D_i is an attribute of job J_i.

Output: Schedule σ (order of jobs execution)

1: Sort the original set J of jobs into the ordered list l using their stage duration attribute D_i^1
2: head ← 1, tail ← n
3: for each job Ji in L do
4: if D_i^2 = m then
5: // Put job Ji from the front
6: σ_{head} ← Ji, head ← head + 1
7: else
8: //Put job Ji from the end
9: σ_{tail} ← Ji, tail ← tail - 1
10: end if
11: end for

Table 1. MapReduce jobs considering 5 samples.

J_i	r_i	D_i	m_i
J_1	5	(4, m)	4
J_2	4	(1, m)	1
J_3	4	(4, r)	30
J_4	30	(6, m)	6
J_5	3	(2, m)	2

Table 2. The ordered list L of five MapReduce jobs.

J_i	r_i	D_i	m_i
J_2	4	(1, m)	1
J_5	3	(2, m)	2
J_1	5	(4, r)	4
J_3	4	(4, m)	30
J_4	30	(6, m)	6

Let's utilize Johnson's method to plan a job for the five MapReduce jobs provided in Table 1 as an example. The computed attribute D_i, which is displayed in the last column, is used to complement these tasks. A sorted list of job vacancies may be found in Table 2. For work execution with the lowest total makespan, job ordering is equal to Johnson's schedule (J_2, J_5, J_1, J_4, J_3). In our case, the ideal schedule has a makespan of 47 days.

Security Issues in HDFS

HDFS, the Hadoop Architecture's foundation layer, contains a large number of data types and is particularly vulnerable to security flaws. There is a risk of data access, theft, and unauthorised disclosure when data is combined in a single Hadoop system. Additionally, the copied data is unsafe, demanding additional security to prevent breaches and vulnerabilities. Most government sectors and companies never employ Hadoop Technology to store valuable data because of the absence of security considerations. They offer security services such as firewalls and intrusion detection systems outside of the Hadoop ecosystem. Hadoop

ecosystem is secured to avoid theft and vulnerabilities by encrypting block levels and individual file systems using the Modified Elliptical Curve Cryptography (MECC) encryption method.

Proposed Methodology

The original data is first obfuscated and encrypted using the modified ECC (MECC) approach before being sent to the cloud. To offer a higher level of data safety, the obfuscated data is then encrypted using the Modified ECC algorithm. The ECC algorithm is a form of public-key cryptography implementation technology (PKC). This method is based on using a prime number function to construct a curve with certain base points. A public key, private key, and a secret key are all created in this proposed system. The secret key "Sc_K" was introduced to the encryption algorithm during encryption, and the encrypted material was then uploaded to the cloud. This encrypted material must first be decoded before it can be accessed. The MECC algorithm is inverted for data decryption, which means Sc_K is subtracted from the decryption calculation. This suggested effort is depicted in full in Figure 6.

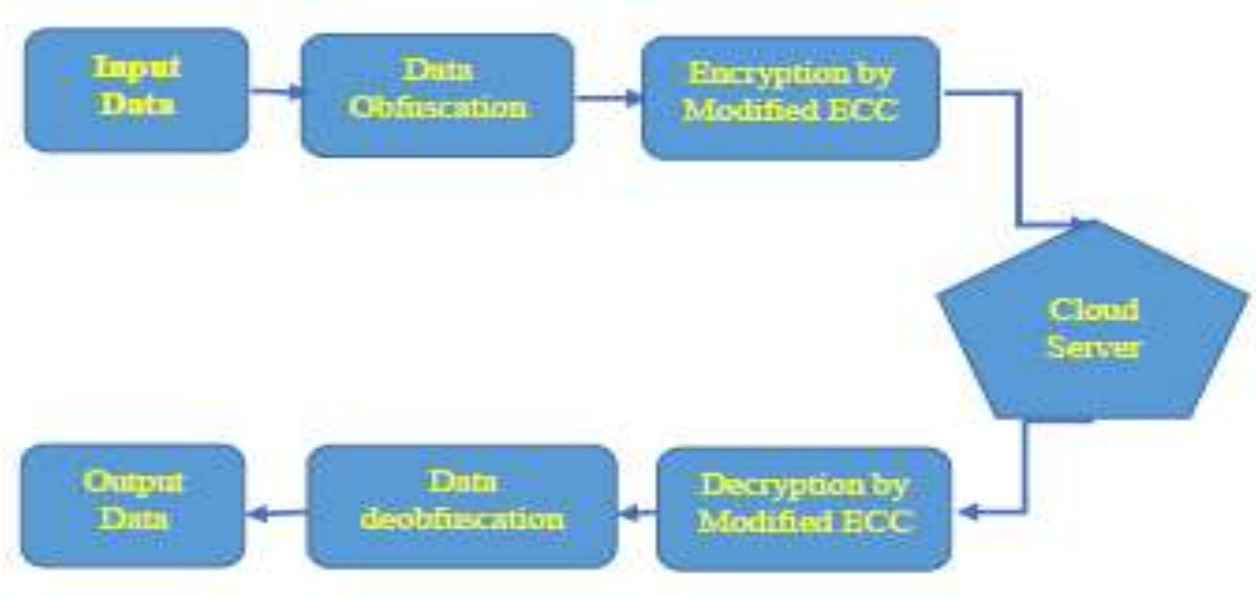


Figure 6. Proposed MECC architecture

Modified ECC Algorithm

ECC is utilized for implementing PKC. A prime number function and a curve with defined base points are used in this method. In terms of this function, it functions as a maximum limit. The ECC is theoretically expressed as,

$$g^2 = x^3 + ax + b \quad (1)$$

The integers are denoted by a & b. The encryption approach's strength is totally dependent on the key generation process employed throughout the cryptography procedure. In this suggested task, three keys must be constructed. For encrypting the message, the server generates the public key "Pb_K" the server generates the private key "Pr_K" and the Sc_K is made up of the Pb_K, Pr_K, and point on curve "H_i". During the key generation process, both Pb_K and Pr_K are created. The data may be encrypted by the sender using the recipient's Pb_K, and the data could be decrypted by the receiver using the Pr_K [11].

Algorithm 2: Modified ECC algorithm

Select a number ‘d’ within the range of ‘n’

//Key Generation

Generate public key using

$Pb_K = Pr_K * H_i$

Generate secret key using

$Sc_K = Pb_K * Pr_K * H_i$

//Data Encyption

Add secret key using

$C_1 = (K * H_i) + Sc_K$

$C_2 = (M + (K * Pb_K)) + Sc_K$

//Data Decryption

$M = (((C_2 - P_K r) * C_1) - Sc_K)$

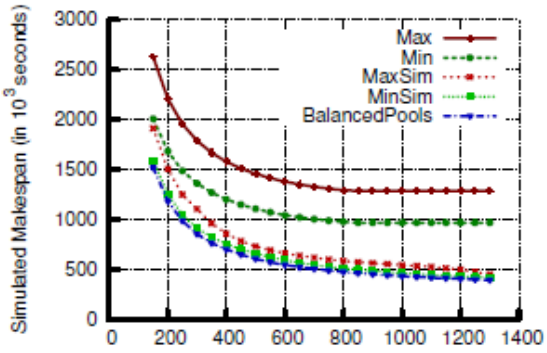
Figure 8. Pseudocode for Modified ECC algorithm

Where, Message is original (M) and in the range of 1 to n - 1, a random number is generated (K). This encrypted data is sent to the user, along with a new source IP address. Pseudocode, which is evinced using Algorithm 2, is used to explain the proposed MECC algorithm. The MECC technique is used to obscure and encrypt the data sent to the cloud.

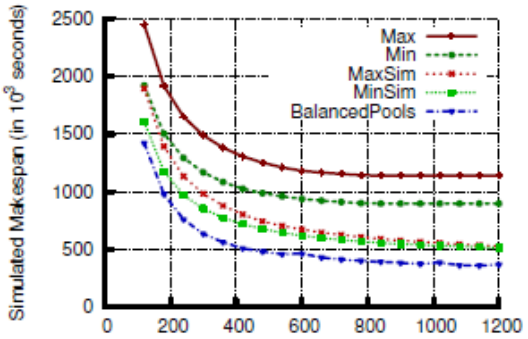
Simulation Results

The proposed work schedule algorithms and their performance are tested and evaluated using the SimMR simulation site. SimMR can play and use performance-based tracking activity data obtained from Hadoop collections. Performance simulation performance is attractive because it enables the sensitivity analysis of MapReduce wide range of planning policies. The consequences of the fake workload with Unimodal and Bimodal distributions are depicted in Figure 7. Five lines are visible in these graphs: The theory makespans are given by Johnson’s Min and Max (right) and reschedule Johnson's (very bad) schedules, respectively. To put it another way, if MapReduce tasks meet exactly the predictions of a two-phase system, the full makespan can be calculated using the abstraction $J_i = (m_i, r_i)$. The performance benefits that can be expected under the best schedule for this quote are represented by the difference between M_{in} and M_{ax}. With a set of particular functions that download Johnson's schedule and reverse Johnson's programme, MinSim and MaxSim simulate SimMR-simulated makespans. Johnson's progression and timeframe Johnson's strategy does not guarantee the best and worst times for this task once we have mapped the Decrease activities at the work / slots level. Because MinSim's "worst" might be much worse than MaxSim's, the gap between MinSim and MaxSim shows the lowest possible development. Finally, the duration of the work program created using the new BalancedPools heuristic is simulated (via SimMR) and represented in BalancedPools. In the X axis, the size of the Hadoop cluster is represented (without losing the standard, by taking 1 map and 1 sliding slot per node). As the size of the collection grows (i.e., as the resources available for the collection become more numerous), the performance benefits decrease, as one would expect. Reimbursement points, on the other hand, vary depending on the workload. This simulation action can be used to determine the set size required to provide a specified (targeted) time for a set of tasks.

Figure 7(a) illustrates that for the Unimodal scenario, Jonhson's schedule (MinSim) reduces the makespan by up to 25% when compared to MaxSim. For bigger cluster sizes, the benefits start to dwindle. The bimodal workload, as seen in Figure 7(b), has quite different outcomes. When compared to Johnson's timetable, the Balanced Pools heuristic gives up to 38 percent makespan benefits (it is suboptimal for this workload). For a variety of cluster sizes, BalancedPools achieves significant extra makespan gains over Johnson's technique. Figure 8 shows the results of modelling the Yahoo! M45 workload. There are two types of bimodality: unimodal and bimodal. Johnson's schedule, on the other hand, yields poorer returns for Yahoo effort!' in both scenarios. We can only see up to a 12% boost in makespan in most trials. In the vast majority of cases, the Balanced Pools heuristic outperforms Johnson's technique by 10% to 30%. The proposed system automatically generates the best job schedule and determines its length based on available resources. These outcomes are fairly comparable to the simulations.

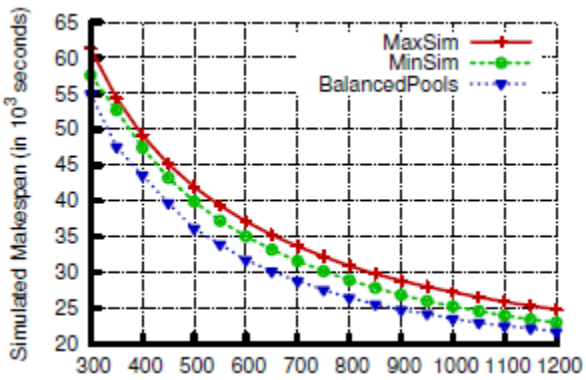


A) In a simulated cluster, the number of map/reduce slots.

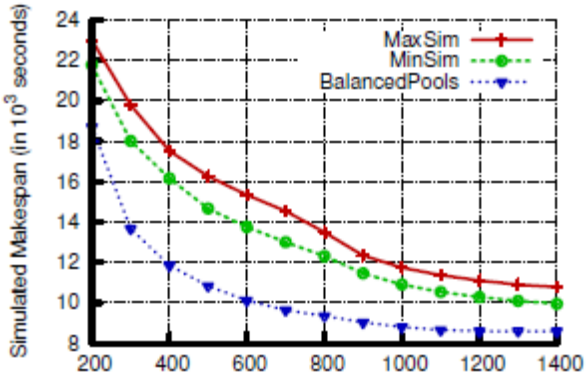


B) In a simulated cluster, the number of map/reduce slots.

Figure 7. Synthetic workload simulation: (a) Unimodal and (b) Bimodal.



A) In a simulated cluster, the number of map/reduce slots.



B) In a simulated cluster, the number of map/reduce slots.

Figure 8. Simulation the workload of Yahoo!: (a) Unimodal and (b) Bimodal.

Conclusion

The authors of this paper are tasked with devising a plan that reduces the time required to finish a collection of MapReduce activities. BalancedPools is a new and heuristic framework that effectively utilizes the structures and behaviors of MapReduce processes to a specific taskload in order to provide the best possible work plan. Heuristic testing is done on the MapReduce list to test what kind of performance improvement we can get. Data analysis functions are usually defined using high-quality SQL abbreviations, which may result in dependable Map Reduction processes. The next step is to tackle the broader issue of reducing the time frame for project completion, which includes MapReduce processes. The MECC method is used to encrypt and encrypt data sent to the cloud.

References

1. Kim, M., Liu, L., and Choi, W., “A GPU-Aware Parallel Index for Processing High-Dimensional Big Data,” IEEE Transactions on Computers, vol.67, no.10, pp.1388-1402, 2018.
2. H. Rehioui, A. Idrissi, M. Abourezq and F. Zegrari, DENCLUE-IM: a new approach for big data clustering, Proc. Comput. Sci. 83 (2016), 560–567.
3. Khalil, Y., M. Alshayeji, and I. Ahmad. "Distributed whale optimization algorithm based on MapReduce." Concurrency and Computation: Practice and Experience 31, no. 1 (2019): 4872.
4. N. Akthar, M. V. Ahamad and S. Khan, Clustering on big data using Hadoop MapReduce, in: 2015 International Conference on Computational Intelligence and Communication Networks (CICN), Jabalpur, pp. 789–795, IEEE, Piscataway, NJ, USA, 2015.
5. P. A. Traganitis, K. Slavakis and G. B. Giannakis, Sketch and validate for big data clustering,IEEE J. Select. Topics Signal Process. 9 (2015), 678–690.
6. S. Fries, S. Wels and T. Seidl, Projected clustering for huge data sets in MapReduce, in: Proceedings of International Conference on Extending Database Technology (EDBT), Athens, Greece, pp. 49–60, OpenProceedings.org, Konstanz, Germany, 2014.
7. M. Vadivel and V. Raghunath, Enhancing Map-Reduce framework for big data with hierarchical clustering, Innov. Res. Comput. Commun. Eng. 2 (2014), 490–498.
8. Hosseini B, Kiani K. 2019. A big data driven distributed density based hesitant fuzzy clustering using apache spark with application to gene expression microarray. Engineering Applications of Artificial Intelligence 79(342):100–113 DOI 10.1016/j.engappai.2019.01.006.

9. Chandra G, Tripathi S. 2019. A column-wise distance-based approach for clustering of gene expression data with detection of functionally inactive genes and noise. In: Mandal J, Dutta P, Mukhopadhyay S, eds. Advances in Intelligent Computing. Vol. 687. Singapore: Springer, 125–149.
10. Li M, Yao X. 2019. Quality evaluation of solution sets in multiobjective optimization: a survey. ACM Computing Surveys 52(26):1–38 DOI 10.1145/3300148.
11. D. Xia, B. Wang, Y. Li, Z. Rong and Z. Zhang, An eflcient MapReduce-based parallel clustering algorithm for distributed traflc subarea division, Discrete Dynam. Nat. Soc. 2015 (2015) Article ID 793010, 18 pp