ACO-HCO: Heuristic Performance Tuning Scheme for the Hadoop MapReduce Architecture

Chiang-Lung Liu[1], Hsiang-Fu Lo[2], Wei-Tsong Lee[3]

[1] Department of Electrical and Electronic Engineering, Chung Cheng Institute of Technology, National Defense University, Taiwan
[2] Chemical Systems Research Divisions, Chung-Shan Institute of Science and Technology, Taiwan
[3] Department of Electrical Engineering, Tamkang University, Taiwan
{chianglung.liu, superalf }@gmail.com, wtlee@mail.tku.edu.tw

Abstract

Hadoop MapReduce is a widely-used cloud computing technology for big data processing. However, the Hadoop configuration parameters settings can significantly change the execution performance. Manual adjustment of the Hadoop parameters will be a time consuming and difficult task. In this paper, we propose ACO-HCO, a Hadoop configuration tuning scheme for MapReduce applications. We use MapReduce applications job history records to generate specific job profiles. Based on these profiles, an objective function for execution time is constructed with gene expression programming algorithm by mining the correlation among the core Hadoop configuration parameters and input data size. Leveraging the objective function, an ACO-based configuration optimizer is able to heuristically search for the optimal configuration for a given application. Experimental results show that ACO-HCO enhances the performance of Hadoop significantly compared with the default configuration. Moreover, ACO-HCO performs better than heuristic approach and the cost-based model in Hadoop performance tuning.

Keywords: Hadoop performance tuning, Ant colony optimization algorithm, Gene expression programming

1 Introduction

MapReduce is a programming model for processing massive amounts of data on large clusters. Hadoop is the most popular open source MapReduce framework from the Apache Software Foundation [1]. The Hadoop MapReduce has over 190 configuration parameters, and overall performance is highly affected by these parameters' values. Because tuning Hadoop application specific performance requires expert knowledge and experience in Hadoop configuration [2], using the default or a set of best practices settings [3-4] to different applications generates unexpected performance results. Therefore, an algorithm that performs automatic Hadoop configuration tuning is required to improve the performance of the Hadoop MapReduce architecture.

The main focus of this work is to develop a novel approach based on ant colony optimization (ACO) [5], called ACO-based Hadoop Configuration Optimization (ACO-HOC), by heuristically searching the better Hadoop configuration parameters for a given MapReduce application. ACO-HOC outperforms existing cost-based tuning approaches, Starfish model [6]. ACO-HOC does not assume on the processing time of per-stage and the correlations between Hadoop configurable parameters. Previously proposed cost-based models [6] usually suppose the execution time of each processing stage to be constant, at the very time that Hadoop adjust configuration settings; linear regression statistical models [7] typically deem that the interrelation between configuration parameters is linear. Differently, ACO-HCO assumes and identifies that Hadoop configuration parameters exhibit non-linear relations with each other.

Additionally, Liao et al. [8] propose a genetic algorithm approach, called Gunther, to directly searches the suitable Hadoop configuration for a specific application. However, Gunther is short of an objective function in the proposed genetic algorithm. Moreover, Gunther repeatedly execute the given Hadoop application for each loop of the genetic algorithm, which is inefficient and unrealistic while accelerating actual Hadoop applications processing the big input datasets. In contrast to Gunther, ACO-HCO is much efficient for adopting a performance models to predict execution time with an ACO-based algorithm.

The rest of the paper is organized as follows. Section 2 describes the related work. In Section 3, the detail of the proposed ACO-HOC scheme is described. Section 4 explains the experimental setup and discuss the performance results; Finally, we conclude in Section 5.

*Corresponding Author: Chiang-Lung Liu; E-mail: chianglung.liu@gmail.com
DOI: 10.3966/160792642020072104023

2 Related Work

2.1 MapReduce and Hadoop

The overall MapReduce model is depicted in Figure 1. The process sequence can be given as follows: (A) the master node assigns the worker nodes with Map and Reduce tasks; (B) the MapReduce system divides all the input files into multiple splits and stores them in blocks so that the worker nodes of Map tasks can locally read the required data blocks; (C) during the Map phase, the worker nodes complete their tasks and store the Map results as intermediate files onto the worker node's local hard disks and wait for the other worker nodes in the Map phase to finish their tasks; (D) after the worker nodes of Reduce phase receive notification of completion from all the worker nodes of Map phase, they remotely read the results and execute the Reduce function after collating and sorting the results; (E) finally, the worker nodes of Reduce phase generate output files of Reduce tasks and store them in the distributed file system.

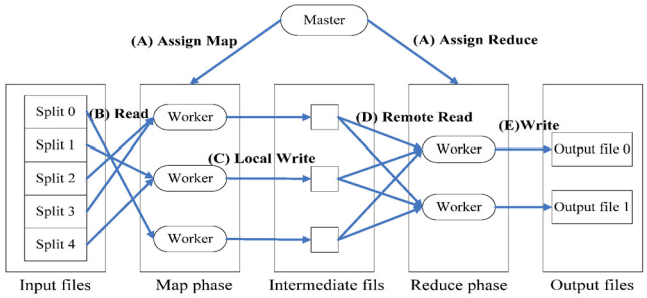

Figure 1. MapReduce model [9-10]

2.2 Hadoop Configuration Parameters

Apache Hadoop is an open source project from the Apache Software Foundation and is an implementation of the MapReduce model. By considering Apache Hadoop version 0.20.2 as an example, amounts of configurable parameters are used to tune the execution status of Map tasks and Reduce tasks.

Table 1 summarizes some of the Hadoop parameters with a description of their usage. These parameters are related to the execution performance. Different types of correlations are observed between these parameters. Some parameters exhibit positive correlations between each other; for example, when the JVM heap size is increased (*mapred.child.java.opt*), the upper limit for the memory buffer size (*io.sort.mb*) in the Map sorting stage is also increased; further, some parameters exhibit negative correlations between them. Although tuning certain parameters can help to reduce the costs in a certain area, it may also increase the costs in another area as the byproduct. For example, if the Map task output results are compressed, the amount of data that is transmitted from the Map tasks to the Reduce tasks as well as the I/O and network transmission costs are reduced; however, the compression and extraction process adds an additional load onto the CPU resources.

2.3 Gene Expression Programming

Gene expression programming (GEP) [11] is popular in data exploration applications for investigating internal correlations between various parameters. GEP exhibits the variability and flexibility of the tree structures in genetic programming (GP) and the simple linear independent fixed length coding of genetic algorithms (GAs). Furthermore, GEP is significantly faster than both GP and GAs and uses relatively simpler coding and is more advantageous while solving complex problems. In this paper, we use GEP method to analyze the complex relation between the major Hadoop parameters.

Table 1. Summary of the major Hadoop parameters

Configuration parameters	Default value	Description
io.sort.factor	10	The number of streams that can be merged while sorting.
io.sort.mb	100	The in-memory buffer size allocated to each task.
io.sort.spill.percent	0.8	A threshold which determines when to run the spill procedure, transferring the in-memory data into the local storage.
mapred.reduce.tasks	1	The number of reduce task(s) configured for a Hadoop job.
mapred.tasktracker.map.tasks.maximum	2	The number of map slots configured on each worker node.
mapred.tasktracker.reduce.tasks.maximum	2	The number of reduce slots configured on each worker node.
mapred.child.java.opts	200	The maximum size of the physical memory of JVM for each task.
mapred.job.shuffle.input.buffer.percent	0.7	The amount of memory in percentage allocated to a reducer to save map outputs during the shuffle procedure.
mapred.reduce.parallel.copies	5	The number of parallel data transfers in the reduce phase.
mapred.compress.map.output	False	Compression rate of map task outputs.
mapred.output.compress	False	Compression rate of reduce task outputs.

3 ACO-HCO Design

3.1 Overview

ACO-HCO is a probabilistic performance tuning method that locates optimal Hadoop configuration solutions by moving through a parameter space resenting all possible solutions for a given application running on a Hadoop cluster to attain optimized performance. The architecture of ACO-HCO is shown in Figure 2. For a given application that need to be optimized, we first collect the settings of core Hadoop parameters and the total processing times with small input data sizes by a job profiler [12]. The resulting profile is used to train the GEP model in the GEP algorithm, which is ultimately used to predict the execution time of given application. Using a GEP guided model for execution time, it is practical to combine ACO approach to search the huge Hadoop parameter space to find the optimized parameters. The main components of ACO-HCO are described in detail in the following subsections.

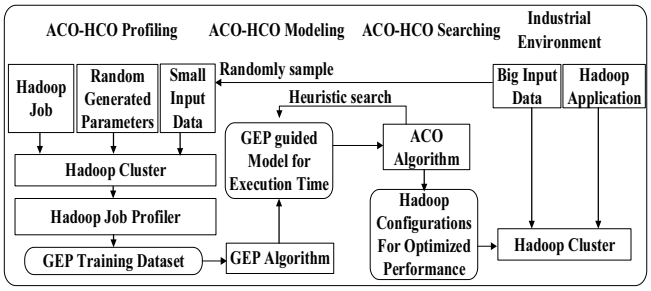

Figure 2. Architecture of ACO-HCO

3.2 ACO-HCO Modeling

The Hadoop application execution time can be depicted in Equation (1) where $p_0, p_1, \ldots, p_n$ are the core Hadoop configurable parameters. Table 2 shows 10 important Hadoop parameters we investigate in this paper. The data types of each parameters in Table 2 determine the corresponding GEP method mathematic functions. Then, GEP algorithm extract the correlation among these parameters from Training dataset.

Table 2. Important Hadoop parameters used in GEP

GEP variables	Hadoop core parameters	Data type
p_0	*io.sort.factor*	integer
p_1	*io.sort.mb*	integer
p_2	*io.sort.spill.percent*	float
p_3	*mapred.reduce.tasks*	integer
p_4	*mapred.tasktracker.map.tasks.maximum*	integer
p_5	*mapred.tasktracker.reduce.tasks.maximum*	integer
p_6	*mapred.child.java.opts*	integer
p_7	*mapred.job.shuffle.input.buffer.percent*	float
p_8	*mapred.reduce.parallel.copies*	integer
p_9	*input data size (GB)*	integer

$$ET = f(p_0, p_1, p_2, \ldots, p_n) \quad \textbf{(1)}$$

GEP algorithm employs $p_0, p_1, \ldots, p_n$ as inputs of combined mathematical functions and then maintain and develop a linear chromosome within the evolution process. At the same time, the linear chromosome constructs an expression tree and generates a form of $f(p_0, p_1, \ldots, p_n)$ to compute estimated execution time and compare it with the real execution time. GEP outputs a ultimate form of $f(p_0, p_1, \ldots, p_n)$ at the termination of evolution procedure and the predicted execution time is the closest to the real working time.

Figure 3 shows the design of the GEP algorithm. The training dataset generated from Hadoop job profiler is the input of GEP algorithm. We run 360 experiments on a Hadoop cluster to establish the training dataset. We adjust the configurable parameter values by hand and operate each application three times and compute the mean processing times. In Figure 3, Lines 1 to 5 set the first-generation initialization of 500 chromosomes, which stand for 500 probable correlations among different Hadoop core parameters. Lines 8 to 29 run an evolution procedure in which each loop indicates a generation of the evolution procedure. An expression tree is transformed from each chromosome. Lines 11 to 17 compute the utility value of a chromosome. GEP generates a predicted Hadoop job processing time and then compares it with the real Hadoop job processing time for each training data. If the comparing result is less than a predefined bias interval, the current chromosome utility value will be added by 1.

We set the bias interval as 50 seconds to allow a maximum of 10% of the error space for considering the Hadoop job real processing time. Line 18 indicates that the evolution loop ends in an ideal condition when the utility value is equal to the amount of training datasets. On the contrary, Lines 20 to 23 show that the evolution procedure follows, GEP will save the chromosome with the best utility value. Lines 24 to 25 indicate that GEP take a genetically modified operation to the current generation for producing variations of the next generation chromosomes at the termination of each generation. We changed the amount of generations from 20000 to 80000 in the evolution procedure of GEP and discovered that the chromosome quality (the ratio of the utility value to the amount of training datasets) was at last larger than 90%. Therefore, we set the amount of generations as 80000. After 80000 generations, GEP produces Equation (2), standing for a correlation between the estimated execution time and the important Hadoop parameters listed in Table 2.

$$\begin{aligned} f = (x_0, x_1, \ldots, x_9) &= (x_7 \times x_6) + (sqrt(1/\log_{10}(x_6) + \bmod \\ &(sqrt((x_0 \times x_8) + (x_3 \times x_1)), pow(x_5, (x_2 + x_1)))) \\ &+ (x_6 + x_4) \times (x_8 + x_9) \end{aligned} \quad \textbf{(2)}$$

```
Input: Hadoop job training dataset generated by the Hadoop Job Profiler;
Output: The correlation between Hadoop job execution time and configurable parameters;
1.  FOR i=1 TO population size DO
2.   construct chromosome (i) by combining the mathematic function and Hadoop parameter GEP variables;
3.   utility (i)=0;
4.   i++;
5.  ENDFOR
6.   best chromosome = chromosome (1);
7.   best utility = 0;
8.  WHILE x < final generation amount DO
9.    FOR a = 1 TO population size DO
10.      Transform chromosome (a) into expression tree (a);
11.      FOR b = 1 TO the amount of training datasets DO
12.         compute the predicted Hadoop job processing time for case (b)
13.       IF ABS (timeDiff) < bias interval THEN
14.          utility (a)++;
15.       ENDIF
16.       b++;
17.      ENDFOR
18.       IF utility (a)=the amount of Hadoop job training datasets THEN
19.          best chromosome =Chromosome (a) GOTO 29;
20.       ELSE IF utility (a) > best utility THEN
21.         best chromosome = Chromosome (a);
22.         best utility = utility (a);
23.       ENDIF
24.   Take replication, selection and genetic operation on chromosome (a) in proportion;
25.   Adopt the adjusted chromosome (a) to replace the original one;
26.   a++;
27.  ENDFOR
28.   x++;
29. ENDWILE
30.  Return best chromosome
```

Figure 3. GEP algorithm implementation

3.3 ACO-HCO Searching

The proposed ACO-HCO scheme use ACO to tune Hadoop parameter settings in this section. We take Equation (2) generated by the GEP algorithm as a fitness function in ACO-HCO searching phase.

ACO is a type of evolutionary computational algorithms proposed by Dorigo et al. in 1997. ACO algorithm is designed by observing the foraging behavior of ant colonies. ACO is a type of multi-point search (multiple ants) algorithm and is not a single-point search strategy. In addition to the positive feedback that is obtained by increasing the pheromone concentration for solutions that exhibit a better performance, a negative feedback that reduces the pheromone concentration for solutions that perform poorly has also been observed. ACO manages two kinds of conflicting searching behavior, exploration and exploitation. Exploration is an algorithm's ability to search broadly through the problems search space and exploitation is an algorithm's ability to search locally around good solutions that have been found previously.

In this paper, we adjust ACO algorithm to deal with the Hadoop platform parameter optimization problem as follows: (1) each search node corresponds to one Hadoop platform parameter; (2) the expected value for an ant's selection of the subsequent Hadoop parameter node determines the length of the run time, that is, the lesser the predicted execution time as calculated by the GEP guided Model for execution time, the more is the probability that the path will be preferred by the ant. The expected values of node selection are set based on each Hadoop parameter values listed in Table 3.

The process of ACO-HCO searching is described as following.

Step 1. Initialization and configuration of the ACO parameter. Set the upper limit for the application run time (t_{max}), number of ants (S), initial pheromones (τ_0), the relative importance of exploitation versus exploration (q_0), pheromone decay parameter (ρ), the relative importance of the trail (α), and the relative importance of the visibility (β).

Step 2. Establish the optimal parameter node path. In the initial state, S ants are randomly placed on the Hadoop parameter nodes. Each ant will begin from its current node and use the conversion rule for calculating the subsequent node to be visited and will eventually complete a full journey step by step. The conversion rules are as Equation (3).

Table 3. Hadoop platform parameter values used in ACO-HCO

Hadoop parameters	Values	Explanations
p_0	10-230	Empirically.
p_1	65-100	According to the input dataset block size. We employ 64MB block size in Hadoop.
p_2	0.6-0.85	Empirically.
p_3	1-16	According to the total amount of reduce slots configured in a Hadoop cluster.
p_4	1-3	According to the condition of a worker node.
p_5	1-3	According to the condition of a worker node.
p_6	180-6000	According to the actual RAM of cluster node and the p_1 value.
p_7	0.70-0.85	Empirically.
p_8	1-10	Empirically.
p_9	The input dataset size in MB	User defined.

$$J = \begin{cases} \max_{u \in J_s(i)} \{[\tau_{iu}(t)]^{\alpha} \times [\eta_{iu}]^{\beta}\}, & if\ q \le q_0 \\ P_{ij}, & otherwise \end{cases} \quad \textbf{(3)}$$

$J_s(i)$ is the set of neighboring nodes that have not yet been visited by ant S located at node i. For nodes that do not belong to $J_s(i)$ or for nodes that have already been visited, the probability of selection of the node is 0. This design prevents the ants from revisiting a node; $\tau_{iu}(t)$ is the pheromone concentration in the segment (i, u) at time t, and η_{iu} is the expected value for the selection of the parameter value candidate as illustrated in Equation (4).. q is a random number between (0, 1), q_0 is a set parameter, $0 \leqq q_0 \leqq 1$, $max_x f(\,)$ is used to find the node u with the highest pheromone concentration (τ) and the lowest predicted run time cost (η) in the GEP guided object function, while p is the probability of selecting the subsequent node and can be obtained using Equation (5).

$$\eta_{iu} = 1/f(x_0, x_1, x_2, \ldots, x_9) \quad \textbf{(4)}$$

$$p_{ij}^{s}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{u \in j_s(i)} [\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}]^{\beta}} \quad \textbf{(5)}$$

When $q > q_0$, although the pheromone concentration is high and the probability of selecting the node with the shortest predicted runtime is high, an ant will still select a node randomly even though there is a node with higher pheromone concentration and a higher probability of a shorter path. Hence, the node with the highest probability is not always selected; therefore, it is still possible for ants to travel to another node. Thus, this behavior is biased toward exploration. When $q \leqq q_0$, the ants must select the node with the highest concentration of pheromones (the shortest predicted execution time); this behavior is biased toward exploitation.

Step 3. Local update of the pheromone concentration. When an ant searches for a feasible solution and passes through an edge (i, j), it will update the pheromone concentration for the edge once to prevent other ants from converging on a local solution and to increase the diversity in path search. The extent of the update has nothing to do with the performance or the selection results of the current ant. The calculation equation can be given as Equation (6).

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\tau_0 \quad \textbf{(6)}$$

where ρ is the pheromone decay ratio parameter, $(1-\rho)$ is the residual pheromone factor, and $0 < \rho < 1$. At the same time, regional update methods are also used so that the concentration of pheromones on the traversed path is reduced, which reduces the attraction of ants to a traversed path. This encourages ants to find new paths and prevents them from limiting the solution to a narrow definition. Return to step 2 until each ant produces a complete path.

Step 4. Global Update pheromone. Once all ants have completed traversing a path, the global update for the pheromones will be performed to reinforce the pheromone concentration of the current optimal path. The calculation is presented in Equation (7). Only the ants that have performed well can leave pheromones behind. This is because a design that only adds pheromones to the current optimal solution will help the ants to find the optimal solution as soon as possible. However, the optimal current path may not necessarily have been found by the current ant but may have been found by a previous ant.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij} \quad \textbf{(7)}$$

Let path $(i, j) \in T^{+}$. Thus, $\Delta\tau_{ij} = Q / L^{+}$; otherwise $\Delta\tau_{ij} = 0$; further, T^{+} is the optimal path that has been found previously, and L^{+} is the total runtime for the previous best path. Q is a parameter that represents the intensity of pheromones. This parameter will affect the speed of convergence to a certain degree. Generally, it is set to 100.

Step 5. Update the optimal path. If $min\ \{L_s\} < L^{+}$, $L^{+} = L_s$ and $T^{+} = T^{s}$, where L_s is the total length of the path found by ant S, T^{s} is the optimal path found, L^{+} is the run time of the current optimal path, and T^{+} is the current optimal path. After updating the optimal path, the updated time is $t = t + 1$.

Step 6. Test stop condition. The ACO stop condition is generally set when the upper time limit (t_{max}) is reached, i.e., when $t \geq t_{max}$; at this time, T^+ is the optimal path that has been found, and L^+ is its total run time; otherwise, return to step 2.

Using the aforementioned ACO-HCO searching process, a path with the highest pheromone concentration will be observed. The combination of nodes that is traversed by the path represents the optimal parameter values for the Hadoop platform. We considered all the optimal parameter values and verified it on the same Hadoop cluster; further, we compared those results with the test results obtained from the optimal parameters using other Hadoop parameter optimization methods to verify the improvements of ACO-HCO.

4 Performance Evaluation

4.1 Experimental Setup

The experimental platform consists of one Intel Xeon server machine equipped with 32 GB memory and an E5-2600 3.20 GHz quad-core processor. We use the same virtual machines (VMs) environments as the Starfish optimizer runs Hadoop applications for fair comparison. Virtualization is constructed by using VMware ESXi v5.1. We use six VMs with the same hardware specifications (2 vCPUs, 4 GB memory and 4 TB hard disk storage) and setup them as Hadoop work nodes. Each VM adopts CentOS 6.1 and Hadoop 0.20.2 CDH3.

We set up a Hadoop cluster in VMs with one Name Node and seven Data Nodes including the Name Node. The experimental Hadoop cluster adopts 64 MB data block and replication level 2 data block.

We use two representative Hadoop applications (i.e., WordCount and Sort) and one specific Hadoop application, MR-based Least Significant Bit (LSB) [13-14]. We also use Hadoop TeraGen and RandomWriter applications to generate four input data sets (5 GB, 10 GB, 15 GB and 20 GB). WordCount is CPU-intensive job; Both Sort and MR-based LSB are disk and memory intensive.

We use the GEP algorithm tool *GeneXproTools* (standard version 5.0.3883) software [15] to construct a GEP guided model for Execution time of Hadoop Job. The other parts of ACO-HOC can implement on a different VM or an independent machine when it runs the collected profiles indirectly.

The parameters used in the ACO algorithm implementation are consist of the max simulation runtime limit (t_{max} = 60), number of ants (S = 10), initial value of the pheromone (τ_0 = 0.0001), the relative importance of exploitation versus exploration (q_0 = 0.85), evaporation rate (ρ = 0.95), the relative importance of the trail (α = 2), and the relative importance of the visibility (β = 1). Table 4 and Table 5 separately presents the ACO-HCO and the heuristic approach (RoT) suggested configuration values for a Hadoop job for different input data size. Table 6 and Table 7 respectively show the suggested configuration values from the Starfish model for both applications.

Table 4. Suggested configuration values from the ACO-HCO scheme

Hadoop parameter name	Suggested configuration values			
input dataset (GB)	5	10	15	20
io.sort.factor	235	226	212	154
io.sort.mb	102	91	102	90
io.sort.spill.percent	0.86	0.71	0.70	0.75
mapred.reduce.tasks	15	8	11	8
mapreduce.tasktracker.Map.tasks.maximum	3	2	2	2
mapreduce.tasktracker.Reduce.tasks.maximum	3	2	2	2
mapred.child.java.opts	275	330	418	549
mapReduce.reduce.shuffle.input.buffer.percent	7	7	7	7
mapred.reduce.parallel.copies	11	8	7	8

Table 5. Suggested configuration values based on industry rule-of-thumb (RoT)

Hadoop parameter name	Suggested configuration values
io.sort.factor	25
io.sort.mb	250
io.sort.spill.percent	0.8
mapred.reduce.tasks	14
mapReduce.tasktracker.map.tasks.maximum	3
mapreduce.tasktracker.reduce.tasks.maximum	3
mapred.child.java.opts	600
input.buffer.percent	0.7
mapred.reduce.parallel.copies	20
mapred.compress.map.output	True
mapred.output.compress	False

Table 6. Suggested WordCount application configuration values from the Starfish system

Hadoop parameter name	Suggested configuration values			
input dataset (GB)	5	10	15	20
io.sort.mb	117	129	128	120
io.sort.factor	35	50	17	76
mapred.reduce.tasks	32	128	176	192
shuffle.input.buffer.percentage	0.43	0.72	0.63	0.83
min.num.spills.for.combine	3	3	3	3
io.sort.spill.percent	0.86	0.85	0.79	0.85
io.sort.record.percent	0.23	0.33	0.33	0.31
mapred.job.shuffle.merge.percent	0.86	0.85	0.83	0.69
mapred.inmem.merge.threshold	660	816	827	765
mapred.job.reduce.input.buffer.percent	0.42	0.43	0.60	0.77

Table 7. Suggested Sort application configuration values from the Starfish system

Hadoop parameter name	Suggested configuration values			
input dataset (GB)	5	10	15	20
io.sort.mb	110	127	109	123
io.sort.factor	48	35	54	27
mapred.reduce.tasks	48	112	160	176
shuffle.input.buffer.percentage	0.76	0.66	0.63	0.88
io.sort.spill.percent	0.84	0.68	0.87	0.82
io.sort.record.percent	0.21	0.15	0.23	0.11
mapred.job.shuffle.merge.percent	0.77	0.88	0.89	0.76
mapred.inmem.merge.threshold	393	787	783	972
mapred.job.reduce.input.buffer.percent	0.65	0.63	0.52	0.79

4.2 Experimental Results and Analysis

The performance comparison of proposed ACO-HOC with the default, Starfish and heuristic approach (RoT) are discussed in this section. We run WordCount and Sort applications on the experimental Hadoop cluster to deal with four different input dataset sizes ranging from 5 GB to 20 GB. Moreover, the input data for the MR-based LSB application contained 1,500 image files (Lena.bmp), which were used to compare the performances of information hiding and information extraction. We executed each application three times with the ACO-HCO suggested parameter values, and computed the mean execution times. Figure 4 and Figure 5 separately display the WordCount and Sort applications performance comparison results.

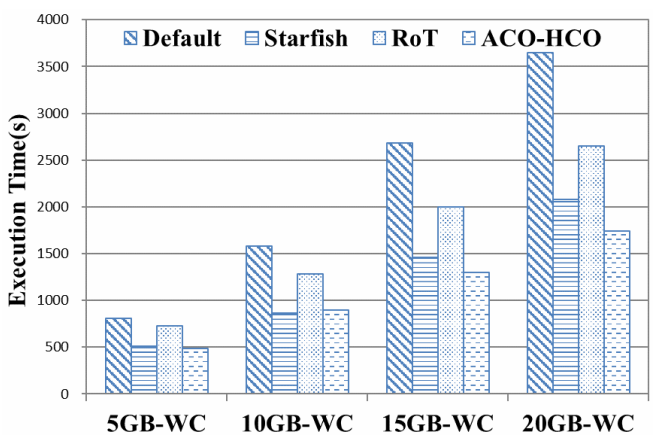

Figure 4. Performance comparison of WordCount

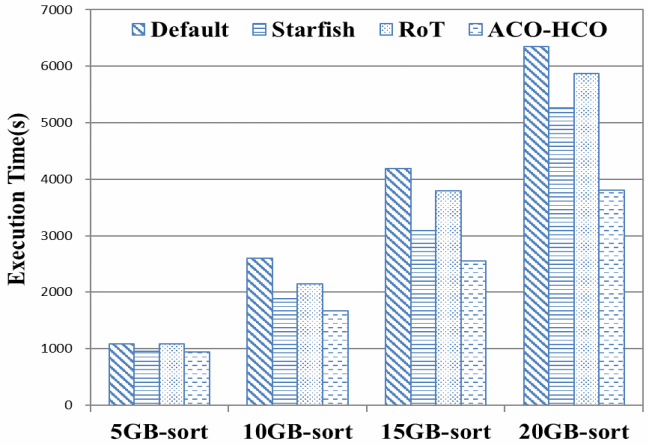

Figure 5. Performance comparison of Sort

In Figure 4, the proposed ACO-HCO enhances WordCount application performance by a mean of 49% in each different input data sizes compared to the default settings, 34% compared to RoT settings, and 10% compared to Starfish model settings. The significant performance advancement reaches 53% when increasing WordCount application input data size to 20 GB for ACO-HCO. In Figure 5, the ACO-HCO improves Sort application performance by a mean of 37% over the default settings, 30% over RoT settings, and 20% over the Starfish model settings. The significant performance improvement attains 44% when increasing Sort application input data size to 20 GB for ACO-HCO.

The proposed ACO-HCO scheme considers both the equipment specifications and the input data size and then suggests configuration values for each application. The RoT settings merely evaluate the equipment specifications (i.e., CPUs and actual RAM) and omits the input data size. The Starfish model also evaluates both the equipment specifications and the input data size. However, the Starfish model overrates the amount of reduce tasks. Such as the Starfish model suggested WordCount application with 192 reduce tasks and Sort application with 176 reduce tasks when increasing input data size to 20 GB. A large amount of reduce tasks increases storage utilization through task parallelization but spends more time setting for these reduce tasks. RoT work disregards the input data size; as a result, the RoT used the same suggested configuration values for each input data as displayed in Table 5.

Figure 6 shows performance results of MR-based LSB application for information embedding and extraction. The performance improvement of the ACO-HCO on the MR-based LSB application for embedding process and extraction process is on average 6% over the RoT work and 3% over the Starfish model. Because the input data of the MR-based LSB application is image type content, it cannot be divided into segments as the text type content of WordCount and Sort application input data. Then, the key-value pair of MR-based LSB application during the Map phase then generates the entire image content as the "value," and consumes plenty of IO processing time for accessing and writing the file onto the local hard disk. Therefore, the performance results of ACO-HOC is significantly reduced.

Experimental results show that ACO-HCO significantly improves the Hadoop performance compared to the default settings. In addition, it is superior to both RoT work and the previously proposed cost-based method, Starfish model, in Hadoop performance tuning.

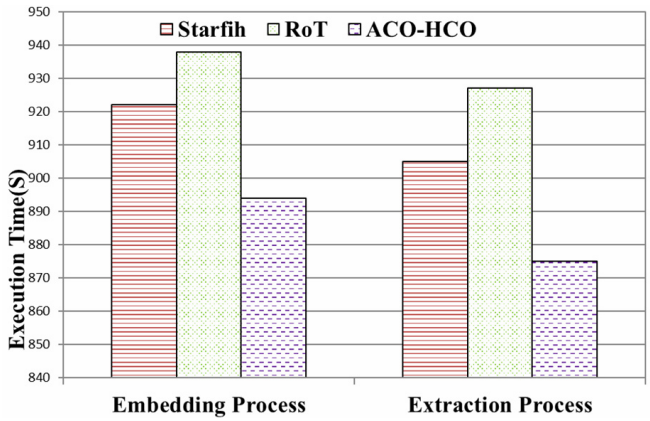

Figure 6. Performance comparison of MR-based LSB

5 Conclusions

Evaluating a Hadoop job with the default configuration values can result in performance problems. Hadoop performance tuning is a challenging issue due to the huge amount of Hadoop configurable parameters. The existing analytical models with impractical assumptions automatically adjust the Hadoop configuration values may decrease the overall model's accuracy and the expected performance enhancements.

In this paper, we propose a novel architecture, ACO-HCO, to tune Hadoop performance by using GEP-guided fitness function that stands for a correlation between the execution time and Hadoop configuration parameters. The proposed ACO-based optimizer can efficiently search the optimized parameter settings and notably enhances Hadoop performance comparing with the default settings. Experiment results show that the ACO-HCO provides better improvements than those provided by the existing representative works including the heuristic approach (RoT) and the Starfish model in Hadoop performance tuning.

Acknowledgements

This study was partially supported by the National Industrial Development Foundation, Taipei, Taiwan.

References

[1] T. White, *Hadoop: The Definitive Guide*, 4th Ed., O'Reilly Media, Inc., 2015.

[2] H. Herodotou, S. Babu, Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs, *Proceedings of the VLDB Endowment*, Vol. 4, No. 11, pp. 1111-1122, August, 2011.

[3] Cloudera Developer Blog, *7 Tips for Improving MapReduce Performance*, http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-MapReduce-performance/.

[4] Intel, *Optimizing Apache Hadoop Deployments*, http://www.intel.com/content/www/us/en/cloud-computing/cloud-computing-optimizing-hadoop-deployments-paper.html.

[5] M. Dorigo, L. M. Gambardella, Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 53-66, April, 1997.

[6] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, S. Babu, Starfish: A Self-tuning System for Big Data Analytics, *Proceedings of the 5th Conference on Innovative Data Systems Research (CIDR '11)*, Asilomar, CA, 2011, pp. 261-272.

[7] H. Yang, Z. Luan, W. Li, D. Qian, MapReduce Workload Modeling with Statistical Approach, *Journal of Grid Computing*, Vol. 10, No. 2, pp. 279-310, June, 2012.

[8] G. Liao, K. Datta, T. L. Willke, Gunther: Search-based Auto-tuning of MapReduce, in: F. Wolf, B. Mohr, D. Mey (Eds.),

19th International Conference on Parallel Processing, Berlin, Heidelberg, 2013, pp. 406-419.

[9] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, *6th Conference on Symposium on Operating Systems Design and Implementation*, San Francisco, CA, 2004, pp. 137-150.

[10] W. T. Lee, M. Z. Wu, H. W. Wei, F. Y. Yu, Y. S. Lin, Dynamically Iterative MapReduce, *Journal of Internet Technology*, Vol. 14, No. 6, pp. 953-962, November, 2013.

[11] C. Ferreira, Gene Expression Programming: A New Adaptive Algorithm for Solving Problem, *Complex System*, Vol. 13, No. 2, pp. 87-129, March, 2001.

[12] Starfish Hadoop Log Analyzer, https://www.cs.duke.edu/starfish/release.html.

[13] H. F. Lo, F. H. Liu, W. T. Lee, C. L. Liu, Y. C. Chou, A. Anpalagan, Applications Transformation Model for MapReduce Information Hiding, *Journal of Internet Technology*, Vol. 18, No. 1, pp. 157-164, January, 2017.

[14] C. F. Lee, J. J. Shen, K. L. Hou, F. W. Hsu, A High-performance Computing Method for Photographic Mosaics Upon the Hadoop Framework, *Journal of Internet Technology*, Vol. 20, No. 5, pp. 1343-1358, September, 2019.

[15] GeneXproTools, http://www.gepsoft.com/gxpt.htm.

Biographies

Chiang-Lung Liu received the B.S. degree in electrical engineering from the Chung Cheng Institute of Technology (CCIT), Taiwan, in 1988, the M.S. degree in information management from the National Defense Management College, Taiwan, in 1995, and the Ph.D. degree in electronic engineering from CCIT, National Defense University, in 2002. Since 2003, he has been with the Department of Electrical Engineering at CCIT, where he is currently a professor. His research interests include cryptography, steganography, steganalysis, multimedia security, image processing, and cloud computing.

Hsiang-Fu Lo received B.S. and M.S. degree in Computer Science from Management College of National Defense University, Taiwan, in 2000 and 2004, and the Ph.D. degree in electronic engineering from CCIT, National Defense University, in 2016. He served Chemical Systems Research Divisions, Chung-Shan Institute of Science and Technology (CSIST), Taiwan. as assistant researcher (2016-08). His research interests include Wireless Video Transmission, Cloud Computing and Computer Networks.

Wei-Tsong Lee received B.S., M.S. and Ph.D. degrees in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan. In 2003, he joined the department members of Electrical Engineering of Tamkang University, Taiwan, as associate professor, and reached professor in 2007. From 2010, he is the chairman of Electrical Engineering Department. His research interests are Embedded System, Computer Architecture, Micro-processor Interface and Computer Networks.