Handling Non-Local Executions to Improve MapReduce Performance Using Ant Colony Optimization

GURWINDER SINGH, ANIL SHARMA, RATHINARAJA JEYARAJ, AND ANAND PAUL, (Senior Member, IEEE)

1School of Computer Applications, Lovely Professional University, Phagwara, Punjab 144411, India
2School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, South Korea

Corresponding author: Anand Paul (paul.editor@gmail.com)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant by the Korean Government under Grant NRF-2017R1C1B5017464, and in part by the School of Computer Science and Engineering, Ministry of Education, Kyungpook National University, South Korea, through the BK21 Four Project, AI-Driven Convergence Software Education Research Program, under Grant 4199990214394.

ABSTRACT Improving the performance of MapReduce scheduler is a primary objective, especially in a heterogeneous virtualized cloud environment. A map task is typically assigned with an input split, which consists of one or more data blocks. When a map task is assigned to more than one data block, non-local execution is performed. In classical MapReduce scheduling schemes, data blocks are copied over the network to a node where the map task is running. This increases job latency and consumes considerable network bandwidth within and between racks in the cloud data centre. Considering this situation, we propose a methodology, “improving data locality using ant colony optimization (IDLACO),” to minimize the number of non-local executions and virtual network bandwidth consumption when input split is assigned to more than one data block. First, IDLACO determines a set of data blocks for each map task of a MapReduce job to perform non-local executions to minimize the job latency and virtual network consumption. Then, the target virtual machine to execute map task is determined based on its heterogeneous performance. Finally, if a set of data blocks is transferred to the same node for repeated job execution, it is decided to temporarily cache them in the target virtual machine. The performance of IDLACO is analysed and compared with fair scheduler and Holistic scheduler based on the parameters, such as the number of non-local executions, average map task latency, job latency, and amount of bandwidth consumed for a MapReduce job. Results show that IDLACO significantly outperformed the classical fair scheduler and Holistic scheduler.

INDEX TERMS Ant colony optimization, cloud computing, heterogeneous performance, MapReduce scheduler, virtualized environment.

I. INTRODUCTION Collecting big data is becoming more common in academia, industry, and research sectors. Hadoop MapReduce [1] is one of the efficient big data processing tools used for making decisions out of big data. Nowadays, Hadoop framework and relevant applications are offered as a service [2] on-demand by many cloud service providers (CSP) over the Internet, as on-premise IT infrastructure for Hadoop MapReduce is not affordable for short-term users. CSP deliver MapReduce service to end users on different schemes, as given below, hosted in virtual machine (VM).

- Private Hadoop MapReduce (pay per VM).
  - Purchase VMs from CSP and setup MapReduce manually.
  - Purchase MapReduce as a service on a cluster of VMs.
- Sharing MapReduce service with more than one user (pay per job basis).

As VMs for Hadoop virtual cluster are hosted across racks in a cloud data centre (CDC), it introduces various heterogeneities at different levels in a virtualized environment:
These two data blocks are logically linked and processed to point two data blocks, only two map tasks are launched. The number of replications in HDFS. If each IS is configured of 64 MB. This results in four data blocks with the desired logical entity in the MapReduce during execution. As shown in Fig. 1, consider a file of size 256 MB with data block size which points to one or more data blocks. It is worth noting that data block and IS are not the same. A data block is a physical entity in Hadoop distributed file system (HDFS), while IS is a logical entity in the MapReduce during execution. As shown in Fig. 1, consider a file of size 256 MB with data block size of 64 MB. This results in four data blocks with the desired number of replications in HDFS. If each IS is configured to point two data blocks, only two map tasks are launched. These two data blocks are logically linked and processed by each map task. Thus, the MapReduce scheduler forms a logical link to execute all data blocks sequentially for a map task. As shown in Fig. 1, consider the map task (map1) launched in node1 where the first data block resides. Now, data block2 is copied from node2 over the virtual network to node1 to finish map1 execution. Similarly, if an IS is configured to contain five data blocks, a map task is launched in the node where first data block resides. The other four data blocks are copied from their respective node to the node where the map task is launched. When the number of data blocks in an IS for a map task increases, the number of non-local executions increases, consuming more virtual network bandwidth. Besides, data blocks are stored based on topological (rack) awareness, which is not implemented for virtual cluster in cloud. Thus, VMs in the virtual cluster for MapReduce could be in the same rack or distributed across racks in the CDC. If all VMs in the virtual cluster are hosted in a single PM, there will not be any non-local executions. When VMs are hosted in different racks across the CDC, network bandwidth consumption is highly critical since it must transfer data through hierarchical switch connections. Sometimes, non-local execution is performed when slave nodes do not have any free slots to process data blocks locally. Thus, data blocks are copied over the local network, which incurs communication costs.

There are different types of data locality, as shown in Fig. 2: node local (NL), rack local (RL), and cluster local (CL) execution. Consider a cluster with two racks (rack1 and rack2), each with two physical servers (node1...node4). There is a set of data blocks (b1...b9) loaded in these physical servers. Assume each map task is assigned with an IS that refers to two data blocks.

1) NL is executing a map task where the required data block resides. Thus, map_task1 processes b1 and b2 locally in node1. This does not transfer data outside the server.

2) If a data block is copied from one server to another server, where map_task2 is running, in the same rack, it is called RL execution. Thus, map_task2 in node2 processes b4 and b3, in which b3 is retrieved from node1. The latency of RL is considerably higher than that of NL as it consumes network bandwidth. If there is more than one VM hosted in a single server, there could be virtual network bandwidth consumption between VMs in the same server.

3) If a data block is copied off the rack across data center, it is called CL execution. Copied data blocks must cross, at least, three switches (top of the rack (ToR) switches and a central switch) to reach the target node. As shown in Fig. 2, IS for map_task3 includes b9 and b6, in which b6 is copied over the network to the node where map_task3 is running. It increases job latency and consumes more physical network bandwidth. So, closer racks are preferred for non-local execution in this case.

MapReduce scheduler performance is highly affected if there is a greater number of non-local executions, especially

![FIGURE 1. HDFS data blocks vs MapReduce IS.](image-url)
in virtualized environment. While offering MapReduce as a service, virtual network bandwidth availability is not always guaranteed in multi-tenant environment as it is shared. Besides, more virtual network bandwidth is consumed in the shuffle phase during the MapReduce job execution. It is a critical situation when the map phase of one job occurs along with the shuffle phase of another job. To overcome these situations and improve the MapReduce performance, we propose IDLACO to minimize the NNLE, thereby, minimizing job latency. Firstly, it minimizes the overall bandwidth consumption during job execution by finding a set of data blocks for each map task of a job to copy across the virtual cluster. Secondly, the target VM is determined based on its performance to copy the data blocks to perform a non-local execution. Since VM performance is heterogeneous, it is essential to dynamically determine the target VM. Finally, if a set of data blocks is copied frequently for repeated job execution, it is temporarily cached to avoid consuming bandwidth during job execution.

The remainder of the paper is organized as follows. A brief literature survey by comparing different parameters is presented in Section II. Proposed methods are modelled and discussed in Section III while Section IV presents the results and analysis of our proposed methodology by comparing with fair scheduler and Holistic scheduler. Finally, Section V concludes the paper.

II. LITERATURE SURVEY
MapReduce task scheduling prefers data blocks to be processed where they reside. However, non-local executions are unavoidable when IS is assigned with more than one data block or required resources are not available in a node despite data blocks are available. To investigate more about data locality, non-local executions in heterogeneous virtualized environment, some of the previous works (PW) are referred based on the parameters H, V, DL, IS, NL, JL, and B, as tabulated in Table 1.

A novel data placement technique [3] is proposed to maximize the MapReduce scheduler performance in virtualized cloud environment. Authors devised a data block placement model as NP-Hard problem to minimize the unexpected global data transfer cost by using replica-balanced distribution tree. The results were compared based on data locality and overall data transfer cost to prove its effectiveness.
Choi et al. [4] proposed a data locality classifier considering the location of all data blocks that constitute IS. After classification, map tasks are scheduled sequentially based on the NNLE. Authors claimed that the proposed algorithm improved total processing time and data copying frequency up to 25% and 28%, respectively, compared with the classical MapReduce scheduler. Scheduling-aware data prefetching was introduced by Li et al. [5] for hybrid cloud data transfer to minimize non-local map tasks using the bandwidth of idle an network. Authors identified the popular data blocks to cache for repeated job execution. The proposed algorithm was compared with capacity and fair scheduler to show its effectiveness. A centralized mapping strategy [6] is proposed to minimize the inter-rack communication cost by cutting down the NNLE in a virtualized and heterogeneous environment. Authors logically divided map and reduce tasks of different jobs and form groups to improve data local executions and minimize communication cost. Based on the number of data local executions of different jobs, map tasks are grouped and scheduled. Based on the communication cost, reduce tasks are grouped and scheduled.

A hybrid scheduling algorithm, HybSMRP, is proposed in [7] to improve data local execution and job latency. Authors proposed two techniques to achieve their objectives: dynamic priority and localization ID. Dynamic priority helps to determine which tasks of jobs should be assigned to the available resource node. Localization ID is assigned to each node in the cluster to get fair amount of data local executions. Li et al. [8] proposed a replica-aware task scheduling and cache mechanism to improve job latency and minimize unnecessary replications. Initially, non-local executions for respective data blocks and frequent failed tasks are traced from the logs to identify where repeated executions are taking place in multi-cloud heterogeneous environment. To minimize makespan and improve resource utilization for a batch of MapReduce jobs in heterogeneous environment, a novel task scheduler is proposed in [9]. It includes two policies HaSTE and HaSTE-A for YARN distributed system. The scheduler assigns the resources to the tasks based on tasks urgency and fitness, especially, for iterative jobs. To improve resource utilization and job latency, DRMJS is proposed in [10] to exploit heterogeneous performance in virtualized environment. DRMJS calculates the performance score for map and reduce tasks separately. Based on the performance score, map and reduce tasks of different jobs are scheduled. There are various classical MapReduce scheduling strategies (FIFO, Capacity, Fair) discussed in [11]. FIFO schedules jobs and allocates entire cluster resources in sequential order as they arrive. Capacity scheduler shares the cluster resources in different proportions based on the requirement of each job. Fair scheduler shares the resources equally among all the jobs that are currently running. These schedulers do not consider heterogeneity and frequent non-local executions.

Chen et al. [12] categorizes a batch of MapReduce jobs into two groups (CPU-intensive and IO-intensive) using dynamic grouping integrated neighbouring search strategy to improve resource utilization and number of data-local executions in heterogeneous computing environment. There are four phases in this proposed method. Phase 1 classifies the MapReduce jobs into two groups. A ratio table is created in Phase 2 for both Task Tracker in MapReduce and Data Node in HDFS. Phase 3 groups a set of data blocks and map tasks. In phase 4, neighbouring approach is used to schedule tasks that consume CPU and IO separately. Data locality and resource utilization aware scheduler is proposed in [13] to save energy cost in heterogeneous cluster. Authors proposed a framework that contains three modules: constructing task list, scheduling, and updating task list. Fuzzy logic is used to calculate the availability of slots in each node based on processor, memory, and bandwidth availability for allocating tasks. In this scheme, DL and RL executions are preferred. Once tasks are scheduled, the task list is updated for the upcoming schedule using fuzzy logic. To improve the makespan and resource utilization, a heuristic method is proposed in [14] to estimate the MapReduce job latency. Firstly, log analysis is performed to profile the jobs already executed several times and understand the variables that affect the job latency. Then, a machine learning algorithm is used to estimate the execution time, which is used to calculate the makespan for a batch of jobs.

Improving data local executions also improves the profit of service providers. Authors employed dynamic programming and ChainMap/ChainReduce in [15] to minimize data transmission time during MapReduce workflow execution. The proposed approach largely relies on data locality to minimize the job latency with frequent replications of data blocks on demand. A holistic scheduler was designed by Handaoui et al. [16] to improve resource utilization and job latency. It consists of three components: resource utilization prediction, determining data local executions, and minimizing interferences from co-locating workloads. Authors demonstrated the proof of concept with the help of constraint programming, genetic algorithm, and local search-based algorithms to compare the performance.

As summarized in Table 1, most of the previous works target job latency by improving data locality in heterogeneous environment. In contrast, IDLACO improves MapReduce scheduler performance based on all the parameters given in the table. We used fair scheduler and Holistic scheduler with genetic algorithm to compare with the results of our proposed method.

### III. PROPOSED METHODOLOGY

MapReduce scheduler performance is affected if there is a greater NNLE in virtualized environment. So, we proposed IDLACO, as presented in Algorithm 1, to minimize the NNLE and amount of bandwidth consumed (ABC) during data block copy across virtual networks. IDLACO consists of the following steps:

1. Calculating the heterogeneous performance of VMs.
2. Modelling the NNLE and ABC.
3. Finding a set of data blocks for each IS using Ant Colony Optimization (ACO).
A. CALCULATING HETEROGENEOUS PERFORMANCE OF VMs

VMs are typically placed across racks in CDC based on the resource availability. The performance of each VM is affected by the resource consumption of co-located VMs. So, VM exhibits varying performance for the same task. DRMJS [10] can be used to model the heterogeneous performance of VMs hosted in heterogeneous PMs. It calculates the performance of a VM based on map and reduce tasks, separately. In this study, we considered the performance and suitability of map tasks of a job along with the node attraction of data blocks to perform non-local executions instead of using the map task performance in each VM. We modified DRMJS in this paper to suit our objectives.

Algorithm 1 shows the steps of our proposed methodology. First, we calculate the performance of vCPU of each VM. As given in Eq. (1), CPU performance of jth VM (V_NodeCPUij) in ith PM (P_Node) (V_NodeCPU) is calculated by finding the PM with maximum CPU frequency (CPU_freq) among all the PMs in which Hadoop VMs are hosted. Besides, the performance of V_Node is based on the number of cores allocated from the total number of P_Node cores (P_Node). Thus, we introduce the performance factor of V_Node in terms of the number of cores (V_NodeCPU) allocated to it.

\[
V_{NodeCPU} = \frac{V_{NodeCPUfreq}}{\max(\forall i, P_{NodeCPUfreq})} \times V_{Node} \quad (1)
\]

Since map tasks require huge disk IO interaction, we calculate the disk IO performance of jth V_Node in ith P_Node (V_NodeDiskIO) based on the current disk bandwidth rate of jth V_Node in ith P_Node (V_NodeDiskIOcurr_disk_band) over the disk bandwidth of kth disk in ith P_Node (P_NodeDisk_band) (Eq. (2)). Besides, it is essential to determine the capability of executing more number data local map tasks performed in a specific V_Node. To specify the disk IO performance based on data block size, we add one more component, data locality (DLj), in Eq. (2) to emphasize that V_Node can run more data local map tasks over time. It indicates the number of data local executions in jth V_Node hosted in ith P_Node.

\[
\forall i,j, V_{NodeDiskIO} = \forall k, \sum_{k} \frac{V_{NodeDiskIOcurr_disk_band}}{P_{NodeDisk_band}} \times DL_j \quad (2)
\]

MapReduce tasks of a job have different resource requirements as users can explicitly configure it. A map task requires more CPU and storage accesses, while a reduce task needs CPU and network bandwidth. To launch map tasks in VMs, it should have seamless disk bandwidth while starting the job and seamless network bandwidth while moving map outputs to reduce nodes where reduce tasks are running. To find the suitable VM for running a map task, we calculate the influence of jth V_Node in ith P_Node for map task (V_NodeCPU) by considering the latency of the last z number of map tasks and reduce tasks executed in jth V_Node, using Eq. (3). If the number of map tasks considered in a V_Node occurs differently, there is no point in considering them. However, as MapReduce jobs in production environment are periodically executed, it is essential to consider the map tasks of the same job executed in the recent past.

\[
\forall i,j, V_{Node^\text{map\_inf}} = \min \left( \forall z, \frac{\text{map\_latency}_{ji}}{\sum_{m=1}^{z} \text{map\_latency}_{jm}} \right) \quad (3)
\]

Using Eq. (4), we obtain the map task performance (V_NodeCPU) in each V_Node based on the CPU performance and disk IO bandwidth of respective V_Node hosted in each P_Node.

\[
\forall i,j, V_{NodeCPU} = V_{NodeCPU} \times \left( 1 - V_{NodeDiskIO} \right) \times (1 - V_{NodeCPU}) \quad (4)
\]

Finally, VMs rank list is prepared using Eq. (5) based on its performance to launch map tasks. We used merge sort; however, we could not find much difference with other sorting algorithm.

\[
\text{map\_rank} = \text{sort}(V_{NodeCPU}) \quad (5)
\]

B. MODELLING NNLE AND ABC

The number of data blocks for an IS is configured before launching MapReduce jobs. When the number of data blocks in an IS increases, the NNLE increases. The number of data blocks to transfer over network for non-local execution can
be intra-rack or inter-rack communication. If data blocks are copied between the racks, it introduces more network traffic for other applications. Thus, we should minimize the NNLE considering the racks in a data centre, which minimizes ABC as well. As discussed in Section I, there are three types of data local execution: NL, RL, and CL. If IS comprises s data blocks for a map task to process, we refer these s data block from the execution log (Fig. 3). Using such information, we can decide whether a data block should be copied over for non-local execution or use the cached data block we can decide whether a data block should be copied over from the execution log (Fig. 3). Using such information, we can decide whether a data block should be copied over network for non-local execution or use the cached data block in the target VM. Data block information is maintained as network for non-local execution or use the cached data block we can decide whether a data block should be copied over network for non-local execution or use the cached data block in the target VM. Data block information is maintained as a triplet $B_b^j(NL, RL, CL) \in IS_x$. It denotes that each data block b of a dataset that belongs to an IS x and resides in $j^{th}$ VM contains information on how a data block is executed (NL, RL, and CL). For instance, consider $B_b^j(1, 0, 0) \in IS_4$. It means that 5th data block that belongs to the 4th IS has been executed as NL in the 2nd VM. In general, the number of IS is equivalent to the number of map tasks. For instance, if there is 1 GB size of input data and data block size is 64 MB, the number of data blocks is 16. If IS size is configured to be 128 MB, it can comfortably include two physical data blocks. This forms 8 IS, which results in eight map tasks. Similarly, each IS information is represented as a triplet $IS_x(NL, RL, CL)$, indicating the information of $x^{th}$ IS. If an IS contains n data blocks, the number of data blocks executed as NL/RL/CL is denoted in the IS triplet. Consider $IS_4(1, 2, 1)$. It means that the 4th IS contains four data blocks, in which one data block is executed within the node, two data blocks are executed within the rack as non-local executions, and one data block is executed across the racks. Thus, the 4th IS accounts for one local and three non-local executions. The residence of each data block is denoted using the first triplet $B_b^j(NL, RL, CL) \in IS_x$ to calculate the virtual network traffic between the servers in a rack, and between the racks in a cluster. Algorithm 1 elaborately explains this sequence.

From the log files, we can find the number of NL, RL, and CL for each map task of a job in the past. The initial plans for map and reduce tasks are prepared by the scheduler. The number of non-local executions in each IS and the overall NNLE can be calculated using Eq. (6). Using this information, we can find the VMs which attract more data blocks and the amount of data consumed across virtual cluster. To represent the network bandwidth relationship among VMs, consider a graph with a set of vertices (V) and a set of edges (E). In a connected graph, V denotes VMs, and E denotes the bandwidth connection between VMs. Bandwidth consumed between the vertices and reading time of a data block from the busy VM denote the cost of a data block in IS. We use a bandwidth consumption ($BC_{(j,k)bw}$) data structure that records the number of data blocks (d) transferred and the amount of bandwidth (bw) consumed (in MBs) between VMs (j and k). For instance, $BC_{(2,3)300}$ indicates the bandwidth consumption between VMs 2 and 3. There are four data blocks transferred and 300 MB of bandwidth consumed between VMs 2 and 3. Using this data structure, we can calculate ABC using Eq. (7), from which we can also find the number of data blocks transferred between each VM. Besides, the number of replications for each data block plays a significant role in minimizing non-local executions. $REP_b(B_b^j(NL, RL, CL) \in IS_x, IS_x(NL, RL, CL))$ denotes the information on the replications of data block b. The downside is, if the replication factor is high, finding the right copy of the respective data blocks is not easy and takes time to find a solution. Thus, it is essential to decide the right replication factor for data blocks.

$$NNLE = \sum_{x=1}^{y} (RL_x + CL_x)$$  \hspace{1cm} (6)$$

$$ABC = \sum_{j=1}^{n} \sum_{k=1}^{n} BC_{(j,k)bw}^{d}$$  \hspace{1cm} (7)$$

C. FINDING A SET OF DATA BLOCKS FOR EACH IS USING ACO

In the previous section, the NNLE and ABC are modelled when a MapReduce job is periodically executed. With log information, we can infer the data blocks that are frequently executed non-local in different VMs. To decide dynamically,
Algorithm 1: IDLACO

| Load input dataset |
|--------------------|
| Find the number of data blocks stored in HDFS |
| Set IS size |

**DRMJS**

**Input:** V:\_Node parameters (virtual CPU, virtual disk, and virtual network), system-level parameters (Disk)

**Output:** V:\_Node performance in terms of map tasks

while (true) do

- Calculate the CPU performance V:\_Node^CPU\_i\_j
- Calculate the disk IO rate V:\_Node^disk\_i\_j
- Calculate the map task influence V:\_Node^map\_inf\_i\_j
- Calculate the map task performance V:\_Node^map\_perf\_i\_j
- Find the rank list of map nodes by sorting V:\_Node^map\_perf\_i\_j in descending order to find map\_rank

end

Find a set of data blocks for each IS using ACO

Get information from scheduler

Control parameter initialization

\( \tau_{x,z} \) - Pheromone matrix initialization

while (true) do

- RWS calculation (\( p_{x,z} \))
- Path construction (Path\_x\_z)
- Ants generation
- Mapping ants with path
- Evaluate objective function for the candidate solution constructed
- Calculate the NNLE and ABC for each data block including replicated data blocks.

\[ REP_{i\_j}(NL, RL, CL) \in IS \_x, IS \_x(NL, RL, CL) \]

- Min (NNLE) and Min (ABC)
- Local pheromone update
- Global pheromone update

end

Get a set of data blocks for each IS that minimizes the NNLE and bandwidth consumed.

Get the list of target V:\_Nodes based on the performance.

Schedule tasks.

Record the schedule into log for future use.

Scheduling information from the MapReduce job scheduler is obtained. With this information, we can find a set of data blocks to transfer over virtual network, which can minimize the NNLE and ABC. It is ensured that the resulting data block set for each IS constitutes minimal NNLE and overall bandwidth consumption. To achieve this, we employ the ACO algorithm for finding a set of data blocks for each IS that can optimize the NNLE and overall bandwidth consumed. The reason for using ACO is to deal with discrete solution space.

We map our proposed model within the ACO algorithm such that parameters in ACO will determine an optimal solution. IDLACO ultimately finds a set of data blocks that belong to different IS that minimize the NNLE, and overall bandwidth consumed. We map the data block selection problem with the optimization problem and give a short glimpse of the ACO algorithm and the parameters used in ACO. Ants foraging behavior can be mapped with an optimization problem as a tree structure with more than one level to find an optimal solution. As shown in Fig. 4, there are \( y \) IS for a MapReduce job. Each IS consists of \( s \) data blocks with three replications \( (r) \) by default. Each level is a decision variable \( (IS) \) in an objective function, and nodes (data block) in each level denote possible solutions from search space. Each data block in IS contains information whether it is NL/RL/CL and the amount of data to transfer. Moreover, each data block has \( r \) replications, which complicate further selection of the right data blocks to copy. For example, if the replication factor is 3 for each data block, \( s \) data blocks are chosen in each level. Once all decision variables obtain a set of data blocks, we can evaluate an objective function and update the solution iteratively until the optimal solution or specified iteration limit is reached.

Algorithm 1 briefly describes the ACO algorithm for this problem.

To discuss this approach more elaborately, we have given the steps of the ACO algorithm and map this problem with ACO and include our objective function. There are problem-specific parameters, such as IS and the number of data blocks in an IS. Besides, there are algorithm-specific parameters. Some of the problem and algorithm-specific parameters are mapped, as presented in Table 2. Pheromone is initialized between every level for each edge in the tree. This value is not a problem-specific parameter. Thus, a random value is assigned for the pheromone matrix in each level and each edge in the tree. Then, based on roulette wheel scheme (RWS) selection, a path from source to destination is selected. In the later iterations, this selection is affected by the amount of pheromone Accumulated in the path. Eq. (8) is used to calculate the density of pheromone in each level, where Path\_x\_z is a path matrix, and \( \tau_{x,z} \) is a pheromone matrix for each path in the tree.

\[ p_{x,z} = \frac{\tau_{x,z}}{\sum_{z=1}^{y} \tau_{x,z}} x = 1, 2, \ldots y \quad z = 1, 2 \ldots s \]  

Using this pheromone matrix \( (p_{x,z}) \), the path is constructed for ants to have chosen in random. However, instead of creating a random path, we construct path using RWS (Eq. (9)). The path for each level is constructed using a probability matrix. If there are four paths in a level, we construct a range for each path. After constructing a path between each level, several ants are randomly generated at each level. Here, an artificial ant is a random number generated between 0 and 1. The number of ants is equal to the number of

| Algorithm-specific parameters | Notation | Problem-specific parameters |
|------------------------------|----------|----------------------------|
| Decision variable (level)    | \( y \)  | IS                         |
| Nodes in each level          | \( s \)  | Data blocks in IS          |
candidate solutions. These ants are mapped with each path at the respective levels. Finally, one ant from each level is selected. If the number of ants is greater, the optimal solution may be reached quicker. However, it takes more computation time. Once a (node) data block in each level is selected, it is then combined to obtain the result. Each node contains two data structures, which have already been explained: $\{ IS_x (NL, RL, CL) \}$ and $B^i_j (NL, RL, CL) \in IS_x$. $IS_x$ denotes NNLE initially decided by the task scheduler.

If there are RL/CL executions mentioned in $IS$, $B^i_j$ is explored to find the right data block number (b) residing in three VMs (j) according to the replication factor. From these three VMs, the amount of bandwidth currently available and size of the data block are noted along with the total number of data blocks to be processed in all target VMs. This is done in every level, resulting in combination of data blocks in each IS. These values are then passed to the objective functions Eqs. (6) and (7).

The combination that gives the optimal value for NNLE and overall bandwidth consumed is selected from the best path. Subsequently, pheromone from the best path is updated with the new probability value in the path matrix. Finally, the respective local and global paths are updated with the pheromone to increase the chance of the current best path to have chosen in the upcoming iterations. Once the algorithm produces a set of data blocks for each IS, they are scheduled for execution based on the performance of target VMs, as presented in the DRMJS part of Algorithm 1. Information on those data blocks was noted for repeated execution of the same job. Over time, if those data blocks are frequent to the same target VMs, they are cached in the target VM until the storage gets exhausted. If this information is not useful later times, it is discarded from the scheduler, which will remove the cached data blocks from the target VMs.

IV. RESULTS AND ANALYSIS

We simulated MapReduce task scheduler (Hadoop 2.7.0) to evaluate the proposed methodology on Ubuntu server with 12-core CPU (hyper-threaded), 64 GB memory, storage $4 \times 1$ TB HDD, and disk bandwidth rating 100 MB maximum. We compared IDLACO with classical fair scheduler and Holistic scheduler [16] based on the parameters, such as the NNLE, average map task latency, job latency, and the ABC (within and between racks). We analysed these parameters for various configurations of the number of data blocks and IS. We assumed 1 TB size of input file and 128 MB for data block size to simulate our algorithm. Wordcount job is used to compare the performance of IDLACO with other schedulers for those parameters. Besides, we assumed CDC with physical servers and VMs to be highly heterogeneous. We considered 10 racks each with 10 PMs that belong to different types, as given in Table 3, to launch VMs of different flavours, as given in Table 4. Altogether, we consider 100 PMs of different types and, 100 VMs (20 VMs in each flavour) are launched across racks in the cluster. Each VM is a
The objective of IDLACO is to minimize the NNLE; thereby, on average over fair scheduler and Holistic scheduler, considerably showed improvement up to 25.2% and 13.5% for the Holistic scheduler, and IDLACO. It is observed that IDLACO minimized the NNLE and overall ABC across clusters are significantly minimized.

Initially, MRAppMaster schedules map tasks by following the locality principle. Then, using the proposed methodology, the NNLE for a MapReduce job. This minimized map task consumes less bandwidth for a map task. Thus, IDLACO helps to identify the data blocks that need not be moved or be copied over virtual network. So, the number of data blocks across the cluster, map task is executed where the first data block resides. Rest of the data blocks are yet to be copied across the cluster. Therefore, it is important to note that doubling the RF and s in an IS does not result in doubling the performance.

Table 3. PMs configuration.

| PMs Types | Configuration of PMs |
|-----------|----------------------|
| PM_Type1  | Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz 6 cores, 32 GB memory, 1 TB HDD |
| PM_Type2  | Intel(R) Xeon(R) CPU E5-24680 v4 @ 2.40GHz 28 cores, 132 GB memory, 3 TB HDD |
| PM_Type3  | Intel(R) Core(TM) i5 CPU 650 @ 3.20GHz 4 cores, 8 GB memory, 1 TB HDD |
| PM_Type4  | Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 4 cores, 8 GB memory, 1 TB HDD |

Table 4. VM Types.

| VM Type | vCPU | Memory (GB) |
|---------|------|-------------|
| Very small | 1 | 2 |
| Small | 2 | 4 |
| Medium | 4 | 4 |
| Large | 8 | 16 |
| Extra large | 12 | 24 |

Table 5. RF and s combinations.

| RF | 3 | 3 | 3 | 3 | 6 | 6 |
|----|---|---|---|---|---|---|
| s  | 1 | 2 | 3 | 4 | 8 | 4 | 8 |

Hadoop node. In total, 98 slave nodes, one resource manager, and one name node are configured. We experimented the schedulers for different combinations (Table 5) of replication factor (RF) and the number of data blocks (s) in an IS.

In Hadoop 2.x version, resource manager, a component of YARN, looks after the scheduling of jobs from different distributed processing tools, such as MapReduce, Spark, etc. Once a MapReduce job is scheduled by a resource manager, application master (MRAppMaster), a component of MapReduce 2.x, schedules map and reduce tasks. Each MapReduce job gets an MRAppMaster to manage their map and reduce tasks running independently. MRAppMaster collects log information and data block locations from the namenode, component in HDFS, for scheduling map tasks across the virtual cluster. Log information is used to understand the repeated pattern of data blocks copied to different VMs. As shown in Fig. 3, MapReduce job log is maintained to identify a set of data blocks frequently copied to different VMs. Initially, MRAppMaster schedules map tasks by following locality principle. Then, using the proposed methodology, the NNLE and overall ABC across clusters are significantly minimized.

Fig. 5 shows the NNLE performed with fair scheduler, Holistic scheduler, and IDLACO. It is observed that IDLACO considerably showed improvement up to 25.2% and 13.5% on average over fair scheduler and Holistic scheduler. The objective of IDLACO is to minimize the NNLE; thereby, minimizing the ABC across racks in the virtual cluster. We initially used ACO to find a set of data blocks to copy on the virtual network. When s is increased for different RF, NNLE increases. This is because map task is executed in the node where the first data block of IS is stored. Thus, other blocks in the IS must be copied to the node where map task is processing the first data block. Fair scheduler copies the data blocks by default and causes to increase in NNLE. However, IDLACO minimized it when s is increased in each IS. For instance, consider \( RF = 3, s = 4 \) and \( RF = 6, s = 8 \). IDLACO improved, on average, up to 20.4% and 17% compared to fair scheduler and Holistic scheduler for \( RF = 3, s = 4 \). Similarly, 9.1% and 6.3% improvement is observed using IDLACO compared fair scheduler and Holistic scheduler for \( RF = 6, s = 8 \). ACO is time consuming, typically, for NP-HP problems when input search space is countably infinite. In the proposed algorithm, the search space for ACO is the number of IS (map task) for the next schedule. So, ACO does not take much time (not more than 3 seconds) to arrive optimal solution for the next schedule.

Based on the observation, when s is increased, NNLE significantly increased with fair scheduler and Holistic scheduler. In contrast, IDLACO showed its considerable performance improvement, as it initially finds a set of data blocks for each IS using ACO. When RF is increased, the number of combination of data blocks in each level increased, but it helps to identify the data blocks that need not be moved or that consumes less bandwidth for a map task. Thus, IDLACO finds a set of data blocks for each IS that could minimize the NNLE for a MapReduce job. This minimized map task latency up to 20.6% and 15.8% in average, as shown in Fig. 6, when compared to fair scheduler and Holistic scheduler for different cases. When s is increased in an IS, it minimized map task latency. However, when RF is doubled for s in an IS, there is a little improvement for map task latency in average. The reason is, even though RF increased that placed a copy of data blocks across the cluster, map task is executed where the first data block resides. Rest of the data blocks are yet to be copied over virtual network. So, the number of data blocks are high to copy across the cluster. Therefore, it is important to note that doubling the RF and s in an IS does not result in doubling the performance.

MapReduce job latency is minimized further by scheduling map tasks based on the dynamic performance of VMs. Fig. 7 shows the job latency of different schedulers for different combinations of RF and s. High degree of heterogeneous configuration of PMs and different flavours of VMs cause heterogeneity in performance. So, even though scheduling a map task in a node, there may be high-performing VMs to process all data blocks in the IS in a short time. Therefore, soon after finding a set of data blocks for an IS that constitutes small NNLE, all VMs that contain data blocks from IS are examined whether the current VM performance is good enough to finish the task quickly. So, IDLACO schedules the map tasks to the VM that delivers high performance in...
processing data blocks. In effect, it improved job latency up to 18.8% and 12.7% on average for all different configurations compared fair scheduler and Holistic scheduler. More specifically, when $s$ and $RF$ increased, job latency was minimized. When $RF$ is high, there is a greater number of data blocks residing across the virtual cluster hosted in a CDC. Thus, MRAppMaster gets more opportunity to examine the performance of different VMs and assign map tasks accordingly. This ultimately caused reduction in job latency. However, the number of combinations to check in each $IS$ by ACO will
be high when $RF$ and $s$ are increased. For instance, when $RF$ and $s$ are doubled from \{RF = 3, s = 2\} to \{RF = 6, s = 4\}, job latency is minimized up to 11.7\% and 10\% compared to fair scheduler and Holistic scheduler. Thus, it is essential to understand the heterogeneous performance in heterogeneous environment to schedule map tasks. Another important claim of IDLACO is to minimize the ABC during map task execution. We assumed no other MapReduce job stands in the shuffle phase. As NNLE is minimized, with no surprise, the ABC during map task execution is also minimized, as shown in Fig. 8, up to 25.7\% and 15.2\% for all different combinations of $RF$ and $s$ compared to fair scheduler and Holistic scheduler. This significant performance gain is due to caching frequent set of data blocks copied over the virtual network. Once the data blocks are cached in the target VM, they are used for future map task execution as MapReduce jobs.
in production environment are periodically executed. If the pattern of data blocks executed is not the same, the cached data blocks are removed from the target VM.

Even though IDLACO performed better than fair scheduler and Holistic scheduler in minimizing the ABC, it is essential to analyse the bandwidth consumption within and between racks. Fig. 9 shows the difference between the schedulers based on ABC within racks. Since there is no rack awareness in the Hadoop virtual cluster, it is not easy to achieve the RL execution. Here, IDLACO always prefers the VM located in the same rack for bringing data blocks mentioned in IS. Thus, moving data blocks between racks is mostly avoided. IDLACO aims to copy the data blocks required for different IS within the rack and avoids transferring them over a network. Thus, the proposed method minimized the number of non-local executions across the cluster. Due to this, IDLACO observed RL bandwidth consumption up to 78.7% and 25.6%, in average, to keep the relevant data blocks within the rack compared to the fair scheduler and Holistic scheduler. Therefore, bandwidth consumption between racks, as shown in Fig. 10, is minimized up to 38.9% and 25.8% compared with fair scheduler and Holistic scheduler. When $s$ is high ($RF = 3, s = 8$), and $\{RF = 6, s = 8\}$ for an IS, IDLACO minimized the ABC up to 30.2% and 18.6% over fair scheduler and 17.7% and 10.2% over Holistic scheduler. When $s$ is a small number with default replication, it has little chance of getting data blocks from the VM residing in the rack. This is because there are insufficient copies of the same data block, especially in the same rack. Thus, it is essential to consider $RF$ high when the number of data blocks in an IS is high. This is the case when $\{RF = 3, s = 1\}$, for which 73.1% and 57.6% of overall bandwidth consumption is recorded between racks for fair scheduler and Holistic scheduler. Thus, using the right combination of $RF$ and $s$ in an IS plays a significant role in improving the job latency by minimizing the NNLE and ABC.

V. CONCLUSION

Hadoop MapReduce is widely used as a service by different sectors. Improving the performance of MapReduce is a primary objective, especially, in heterogeneous virtualized cloud environment. When an IS is assigned with a greater number of data blocks, NNLE increases, leading to high bandwidth consumption and job latency. To overcome this, we proposed IDLACO to find a set of data blocks for each map task of a job to minimize the NNLE and ABC. Then, the target VM is determined based on its heterogeneous performance to perform non-local executions. Finally, if a set of data blocks are copied over network repeatedly, it is decided to temporarily cache those data blocks in the target VM. Thus, IDLACO outperformed fair scheduler by 25.7%, 20.6%; 18.8%, 25.7%, and Holistic scheduler by 13.5%, 15.8%, 12.7%, 15.2% on average for the parameters NNLE, average map task latency, job latency, and the ABC, respectively, for a MapReduce job.

REFERENCES

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in Proc. 6th Symp. Oper. Syst. Des. Implement., 2004, pp. 137–149.
[2] R. Boutaba, L. Cheng, and Q. Zhang, “On cloud computational models and the heterogeneity challenge,” J. Internet Services Appl., vol. 3, no. 1, pp. 77–86, May 2012.
[3] W. Chen, I. Paik, and Z. Li, “Tology-aware optimal data placement algorithm for network traffic optimization,” IEEE Trans. Comput., vol. 65, no. 8, pp. 2603–2617, Aug. 2016.
[4] D. Choi, M. Jeon, N. Kim, and B.-D. Lee, “An enhanced data-locality-aware task scheduling algorithm for Hadoop applications,” IEEE Syst. J., vol. 12, no. 4, pp. 3346–3357, Dec. 2018.
[5] C. Li, J. Zhang, Y. Chen, and Y. Liao, “Data prefetching and file synchronizing for performance optimization in Hadoop-based hybrid cloud,” J. Syst. Softw., vol. 151, pp. 133–149, May 2019.
[6] L. Chen and Z. H. Liu, “Energy- and locality-efficient multi-job scheduling based on MapReduce for heterogeneous datacenters,” Service Oriented Comput. Appl., vol. 13, no. 4, pp. 297–308, 2019.
[7] A. Gandomi, M. Reshadi, A. Movaghar, and A. Khademzadeh, “Hyb-SMRP: A hybrid scheduling algorithm in Hadoop MapReduce framework,” J. Big Data, vol. 6, no. 1, pp. 1–16, Dec. 2019.
[8] C. Li, J. Zhang, and H. Tang, “Replica-aware task scheduling and load balanced cache placement for delay reduction in multi-cloud environment,” J. Supercomput., vol. 75, no. 5, pp. 2805–2836, 2019.
[9] Y. Yao, H. Gao, J. Wang, B. Sheng, and N. Mi, “New scheduling algorithms for improving performance and resource utilization in Hadoop YARN clusters,” IEEE Trans. Cloud Comput., early access, Jan. 23, 2019, doi: 10.1109/TCC.2019.2949779.
[10] J. Rathinaraja, V. S. Ananthanarayana, and A. Paul, “Dynamic ranking-based MapReduce job scheduler to exploit heterogeneous performance in a virtualized environment,” J. Supercomput., vol. 75, no. 11, pp. 7520–7549, Nov. 2019.
[11] I. A. T. Hashem, N. B. Anuar, M. Marjani, E. Ahmed, H. Chirona, A. Firdaus, M. T. Abdullah, F. Alostaibi, W. K. M. Ali, I. Yaqoob, and A. Gani, “MapReduce scheduling algorithms: A review,” J. Supercomput., vol. 76, no. 7, pp. 4915–4945, 2020.
[12] C.-T. Chen, L.-J. Hung, S.-Y. Hsieh, R. Buyya, and A. Y. Zomaya, “Heterogeneous job allocation scheduler for Hadoop MapReduce using dynamic grouping integrated neighboring search,” IEEE Trans. Cloud Comput., vol. 8, no. 1, pp. 193–206, Jan. 2020.
[13] J. Wang, X. Li, R. Ruiz, J. Yang, and D. Chu, “Energy utilization task scheduling for MapReduce in heterogeneous clusters,” IEEE Trans. Services Comput., early access, Jan. 14, 2020, doi: 10.1109/TSC.2020.2966697.
[14] A. Gandomi, A. Movaghar, M. Reshadi, and A. Khademzadeh, “Designing a MapReduce performance model in distributed heterogeneous platforms based on benchmarking approach,” J. Supercomput., vol. 76, no. 9, pp. 7177–7203, Sep. 2020.
[15] J. Wang, X. Li, R. Ruiz, H. Xu, and D. Chu, “Allocating MapReduce workflows with deadlines to heterogeneous servers in a cloud data center,” Service Oriented Comput. Appl., vol. 14, no. 2, pp. 101–118, Jun. 2020.
[16] M. Handaoui, J. E. Dartois, L. Lemarchand, and J. Boukhobza, “Sala- mander: A holistic scheduling of MapReduce jobs on ephemeral cloud resources,” in Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. (CCGRID), May 2020, pp. 320–329.

GURWINDER SINGH is currently doing research at the School of Computer Applications, Lovely Professional University, Punjab, India. He is working as an Assistant Professor with the Sikh National College Banga, Punjab. He has more than 15 years of teaching experience. His research interests include big data, hadoop, databases, and cloud computing. He is a member of CSTA.
ANIL SHARMA received the Ph.D. degree from HPU Shimla, India. He is currently working as a Professor with the School of Computer Applications, Lovely Professional University, Punjab, India. He has over 18 years of academic experience. He has published various research articles in reputed journals. His areas of interests include databases, cloud computing, and the IoT.

RATHINARAJA JEYARAJ received the Ph.D. degree from the Department of Information Technology, National Institute of Technology, Karanataka, India. He is currently a Postdoctoral Researcher with the School of Computer Science and Engineering, Kyungpook National University, Daegu, South Korea. His current research interests include big data processing, cloud computing, the IoT, and applied soft computing.

ANAND PAUL (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the National Cheng Kung University, Tainan, Taiwan, in 2010. He is currently working as a full-time Associate Professor with the School of Computer Science and Engineering, Kyungpook National University, Daegu, South Korea. He represented Korea for the M2M Focus Group and MPEG during 2010–2015. His research interests include algorithm and architecture re-configurable embedded computing. He has guest edited various international journals.