IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 25, 2022, accepted May 9, 2022, date of publication May 20, 2022, date of current version May 31, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3176729

Optimizing MapReduce Task Scheduling on Virtualized Heterogeneous Environments Using Ant Colony Optimization

RATHINARAJA JEYARAJ AND ANAND PAUL, (Senior Member, IEEE)
School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, South Korea

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

This work was supported by the BK21 FOUR Project (AI-driven Convergence Software Education Research Program) through the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, South Korea, under Grant 4199990214394.

ABSTRACT Consuming Hadoop MapReduce via virtual infrastructure as a service is becoming common practice as cloud service providers (CSP) offers relevant applications and scalable resources. One of the predominant requirements of cloud users is to improve resource utilization in the virtual cluster during the service period. However, it may not be possible when MapReduce workloads and virtual machines (VM) are highly heterogeneous. Therefore, in this paper, we addressed these heterogeneities and proposed an efficient MapReduce scheduler to improve resource utilization by placing the right combination of the map and reduce tasks in each VM in the virtual cluster. To achieve this, we transformed the MapReduce task scheduling problem into a 2-Dimensional (2D) bin packing model and obtained an optimal schedule using the ant colony optimization (ACO) algorithm. As an added advantage, our proposed ACO based bin packing (ACO-BP) scheduler minimized the makespan for a batch of jobs. To showcase the performance improvement, we compared our proposed scheduler with three existing schedulers that work well in a heterogeneous environment. As expected, results show that ACO-BP significantly outperformed the existing schedulers while dealing with workload and VM level heterogeneities.

INDEX TERMS Ant colony optimization, bin packing, heterogeneity, MapReduce, resource utilization, task scheduling.

I. INTRODUCTION

There is an exponential growth in data volume [1] in the last couple of decades. Small scale businesses and research sectors wish to process such huge data and get insight for the benefit of decision making. There are different big data processing tools widely available to accomplish different data processing objectives. MapReduce [2] is one of the efficient batch processing tools to crunch big data. It processes huge data in parallel on a cluster of physical machines (PM) or VMs. However, deploying on-premise Hadoop infrastructure is still not affordable for small-scale businesses to store humongous data and process them on demand due to the complex responsibilities to manage the infrastructure. This drives them to seek cost-efficient cloud-based MapReduce services from CSPs like Amazon and Google. They offer MapReduce as a service over the Internet on a cluster of PMs or VMs [3]. As the virtual cluster is scalable on-demand and pay-per-use basis, most of the cloud users prefer Hadoop MapReduce on a cluster of VMs. Despite CSP offering an infinite amount of virtual resources, they are not utilized 100% at any point of time during the service period. On a rough estimation, if 0.25 GB of memory is unused per VM, overall unused memory in a virtual cluster of 200 VMs is approximately 50 GB. It largely affects cloud users to pay for unused capacity over time, which is highly undesirable on a business platform. Such resource under-utilization occurs due to many reasons for different applications. One of the primary problems is the existence of various heterogeneities in the Cloud Data-Center (CDC). A layer of heterogeneities is identified from the platform level (cluster of PMs) to the application level (MapReduce jobs), which are elaborately discussed in [4]. They are namely, hardware heterogeneity, VM heterogeneity, and workload heterogeneity.

The associate editor coordinating the review of this manuscript and approving it for publication was Tomas F. Pena.

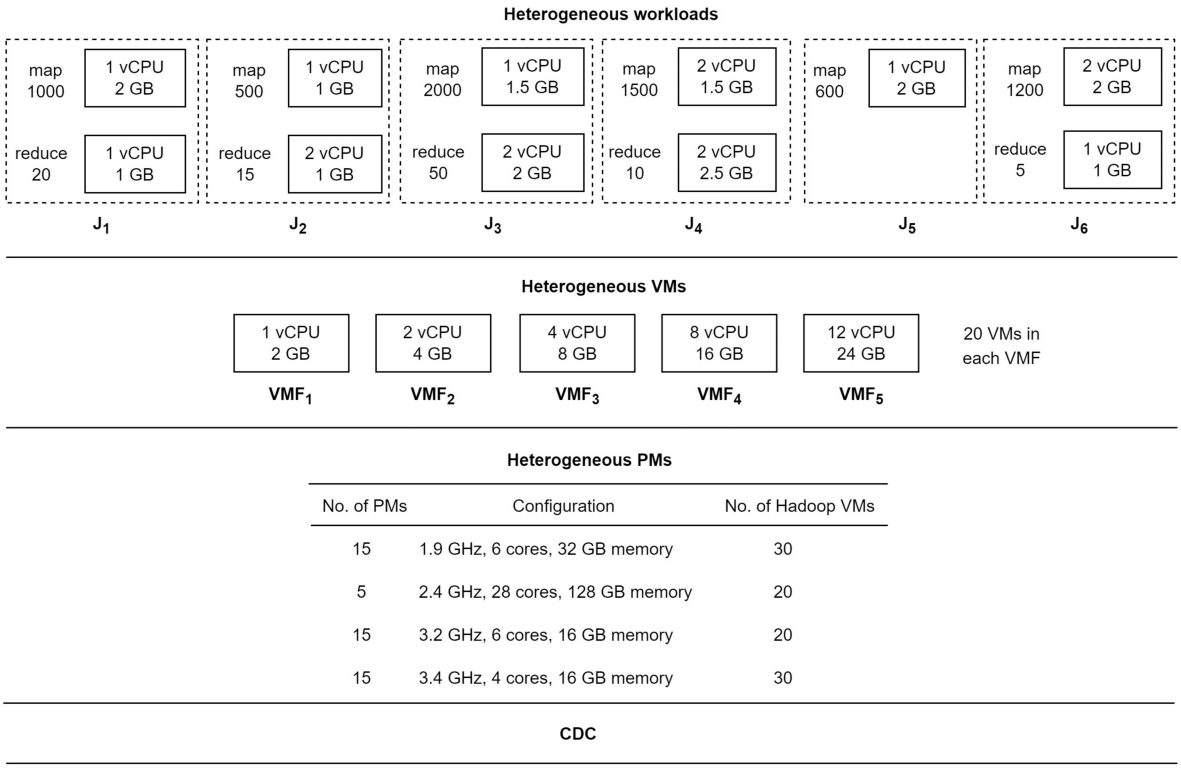

FIGURE 1. Heterogeneous workloads, VMs, and PMs in CDC.

These heterogeneities are further discussed by referring to the information given in Fig. 1. Consider a CDC that contains a set of PMs ($PM_1, PM_2, \ldots PM_{50}$), VMs ($VM_1, VM_2, \ldots VM_{100}$) that belong to different VM Flavors/types ($VMF_1, VMF_2, \ldots VMF_5$), and MapReduce jobs ($J_1, J_2, \ldots J_6$). In general, PMs in the CDC are not of similar configuration and capacity. It is because when the number of cloud users increases and applications demand varying resource requirements, CSPs are forced to equip the CDC with different servers in terms of configuration and performance. This negatively introduces heterogeneous performance for applications. Sometimes, services are hosted on a cluster VMs of different types [5] to satisfy users' requirements based on price. The resource configuration of these VMs may change over time due to competitors' business plans and other technological advancements. Therefore, the same application can be hosted on VM that belongs to different types. Finally, a batch of MapReduce jobs is periodically executed on the hired virtual cluster. These jobs exhibit heterogeneous nature based on varying resource requirements, input dataset size, job nature (CPU or IO-intensive), etc. Moreover, job execution order in a batch results in different job latency, makespan, and resource utilization. Hence, heterogeneity at any level can significantly impact MapReduce job performance [6] and pose challenges to satisfy user expectations. So, in this paper, we have addressed VM and workload level heterogeneities for MapReduce task scheduling to maximize resource utilization and minimize makespan on heterogeneous virtual environments.

Upon the user request for MapReduce as a service on a cluster of VMs, VMs in MapReduce virtual cluster is deployed [7] on different PMs located in different racks in the CDC. Then, huge data are loaded or streamed onto the MapReduce virtual cluster. Data is then divided into small units, called ''blocks'' (by default, 128 MB), and distributed across the MapReduce virtual cluster with default replication factor (3), which is processed by a set of MapReduce jobs on demand. A MapReduce job consists of two primary tasks: map and reduce.

- A map task receives a set of data blocks as input, reads data as records from the data blocks, and produces intermediate output records, by partitioning them into multiple segments, which are then distributed to many reduced tasks based on different criteria.
- A reduce task gets a portion of these intermediate output records from all the map tasks. In general, it performs three magical steps (merge, sort, and group). Each reduce task merges the collected intermediate output records into a single file, sorts, and groups them based on the key. Finally, reduce function executes a user-defined logic on the output of the group function to produce the result.

For both map and reduce tasks, a resource unit is allocated for execution. It is denoted by ''slot'' in MapReduce

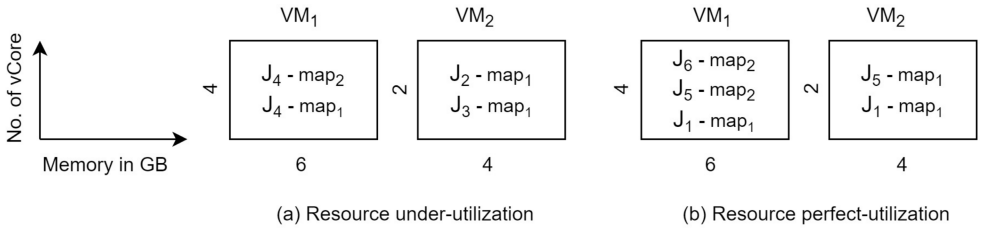

FIGURE 2. Task scheduling with/without heterogeneous capacity awareness.

v1 and represented as an ordered pair $< vCPU, memory >$. Therefore, a slot is a logical pack of virtual CPU (vCPU) and memory. VMs are logically partitioned into multiple discrete slots which can run individual map/reduce tasks. The downside of this approach is these slot sizes are fixed and cannot be configured on a job basis. As heterogeneous jobs in batch demand different slot sizes, we need to set the maximum resource requirement of a task among all jobs as common slot sizes. Moreover, a map task cannot be executed on reduce slots and vice versa. This leads to huge resource under-utilization, and hence, such a concept is eliminated in MapReduce v2 [4], wherein a slot is a called ''container''. Unlike the slot, the container is flexible and can be defined for map/reduce tasks of each job. Consider the MapReduce jobs $\{J_i$, where i=1 to 6$\}$ mentioned in Fig. 1. These jobs demand different sizes of containers for map and reduce tasks. For example, the map task of J_1 requires 1 vCPU and 2 GB memory, and the reduce task requires 1 vCPU and 1 GB memory. In addition, the number of map and reduce tasks, and the job nature also vary. When a batch of such heterogeneous jobs is periodically executed, flexible containers impact makespan and resource utilization. Moreover, VMs in MapReduce virtual cluster are also highly heterogeneous [7], as shown in Fig. 1. This increases the complexity of task scheduling due to the varying number of containers launched in VMs over time. Hence, workload and VM level heterogeneities together affect the performance task scheduling in terms of makespan and resource utilization. It is illustrated with an example given in Fig. 2. Consider two VMs with different configurations $< vCPU, memory >$: $< 4, 6 >$ and $< 2, 4 >$. As shown in Fig. 2(a), scheduling first and second map tasks (map_1, map_2) from J_4 in VM_1 leaves 3 GB of memory unused till their completion. Similarly, scheduling the first map task (map_1) from J_2 and J_3 in VM_2 leaves 1.5 GB of memory unused. However, scheduling map tasks of different jobs based on the size of containers and VMs can minimize memory wastage. For instance, as shown in Fig. 2(b), scheduling the first map task (map_1) from J_1, second map task (map_2) from J_5, and J_6 completely utilized all the resources in VM_1. Similarly, the first map task (map_1) from J_1 and J_5 utilized all the resources in VM_2. Reduce tasks of different jobs also demand varying size containers. In addition to these heterogeneities, map and reduce tasks have different constraints (data locality for map tasks and minimal network bandwidth for reduce tasks) to meet before scheduling. Therefore, finding the right combinations of map and reduce tasks from different jobs for each VM becomes a complex task. It ultimately results in virtual resource under-utilization, which in turn degrades the makespan.

Motivation: If there are hundreds of VMs and tens of MapReduce jobs with thousands of map and reduce tasks, finding a schedule that chooses the right combination of map and reduce tasks from different jobs in each VM becomes an NP-complete problem. To mitigate the impact of these heterogeneities, the following key contributions are provided in this paper.

- We, initially, formulate the typical MapReduce task scheduling problem into a constrained 2D bin packing problem to improve the Quality of Services (QoS), such as makespan and resource utilization. The idea of bin packing is given n bins of capacity c and m objects of different weights/sizes, the objective is to minimize the total number of bins to contain all m objects. We assume each VM in the virtual cluster as a bin and assign a suitable combination of map and reduce tasks from different jobs, satisfying the task-related constraints.
- Then, we employ Ant Colony Optimization (ACO) algorithm to obtain a suitable combination of map and reduce tasks from different jobs in a batch. The feasibility of other meta-heuristic optimization algorithms in a cloud environment for scheduling problems is critically reviewed in [8]. The reason for choosing ACO is the problem we solve in this paper involves a huge discrete search space, for which ACO is typically used for finding an optimal solution [9].

The rest of this paper is organized as follows. Related works on MapReduce task scheduling in a heterogeneous virtualized environment for heterogeneous workloads and bin packing are discussed in Section 2. The proposed method to find the right combinations of the map and reduce tasks using ACO is formulated in Section 3. Results of our proposed method are presented in Section 4, while the conclusion and future work are mentioned in Section 5.

II. LITERATURE SURVEY

In this section, we discussed some of the significant research works on heterogeneity that exists in MapReduce jobs and VMs, to improve job latency, makespan, and resource utilization. The classical fair scheduler [10] tends to fairly share the resources among the jobs in a batch. It does not consider the heterogeneities that exist on the cloud data processing platform. A heuristic-based MapReduce job scheduler (HMJS) is proposed in [11] to model the multi-layer heterogeneity that exists in a cloud-MapReduce virtualization environment. The authors modelled a MapReduce task scheduling problem as 2D bin packing using a simple heuristic. Then, they used roulette wheel scheme (RWS) based data block placement on heterogeneous VMs to improve makespan and resource utilization. Various scheduling schemes for Hadoop MapReduce in a heterogeneous environment are discussed in [12]. The authors focused on how the heterogeneous environment affects the performance in MapReduce job execution sequence. To improve data locality, minimize the number of non-local executions, and virtual network bandwidth consumption, ACO is used in [13], which splits and spreads the data block based on processing capacity VMs. A similar approach is used in [14] to improve the MapReduce scheduler performance in a heterogeneous environment. The data blocks are allocated to nodes based on their processing capacity and made scheduling decisions separately for the map and reduce tasks based on the computing ability of each node. A novel greedy scheduling algorithm is proposed in [15] to allocate resources for heterogeneous MapReduce jobs. The authors minimized service time and cost by allocating resources efficiently. Estimating execution time and resource usage of jobs in the batch is done in [16] to prepare a schedule in the heterogeneous environment. To achieve this, the authors extracted the execution log of jobs and used it to estimate the parameters, which are then used for scheduling jobs to improve resource utilization and makespan. Chi-Ting Chen *et al.* [17] proposed a dynamic grouping integrated neighbouring search strategy to improve resource utilization and data locality in a heterogeneous environment. First, the authors grouped heterogeneous jobs into two categories: IO-bound, and CPU bound. These job types require varying resources throughout the execution. Therefore, jobs reordering is performed to occupy the available resources. To handle the varying size of map/reduce tasks of different jobs, a flexible elastic container is devised in [18] to scale up/down the resources at runtime based on tasks' resource requirements. As the size of the container is not fixed in MapReduce v2, it is essential to manage the complexity introduced by workload heterogeneity.

Hardware heterogeneity is more prevalent when heterogeneous and federated clouds are used for hosting various big data applications. Therefore, Thourayas Gouasmi, *et al.* proposed a distributed heuristic algorithm in [19] to improve job response time and cost by increasing the number of data local executions and minimizing the bandwidth consumption between clouds. This problem is modelled as a mixed-integer program to evaluate the performance of the proposed scheduler. Mostly, dynamic schedulers ignore data block placement, especially, in the heterogeneous environment while concentrating on other parameters to improve. Therefore, a multi-objective optimization problem is modelled and solved in [20] using a genetic algorithm in the heterogeneous environment, for dynamically handling data block placement and resource scaling. A heuristic-based algorithm [21] is introduced to improve energy efficiency while scheduling a batch of heterogeneous jobs in a heterogeneous environment. Based on the energy consumption in each node, a decision is made to launch map and/or reduce tasks. An ACO algorithm is used in [22] to finalize the job execution in a batch based on heterogeneous job size and its expected latency. However, the job taking less data and response time is given high priority in the job schedule. In addition, the authors used Artificial Neural Network (ANN) to predict the resource usage of a job in the heterogeneous execution environment. In [23], performance and monetary cost tradeoffs in MapReduce job scheduling are addressed with the help of Pareto-optimization to find a near-optimal solution. It helps cloud service providers to charge their users based on the IO operations performed.

Bin packing is being widely applied in various fields, especially in a cloud environment, from task scheduling [24], job scheduling [20], to VM scheduling [25], for different objectives and improving various QoS, such as latency, makespan, resource utilization, etc. To improve the resource utilization and application response time, Jesus *et al.* [26] proposed a system that consists of three modules to place the maximum number of tasks in VMs. The authors are predicting the maximum resource requirement of a task, scheduling the task to suitable VMs, and monitoring the resource availability. A dynamic bin packing model for cloud resource allocation is discussed in [24] to minimize the total cost of the VMs used over time. A multi-dimensional bin packing problem for minimizing energy is introduced in [27] along with a fine-grained map/reduce task scheduling. The authors specifically addressed dynamic resource management and job scheduling for big data applications in a heterogeneous virtualized environment. Task consolidation using bin packing with meta-heuristic algorithms has many use cases including resource management in cloud computing. Based on the problem, the dimension of bin packing differs. For example, a simple 1D bin packing problem and applicability of evolutionary algorithms are explored in [28]. A hybrid evolutionary algorithm is used in [29] to solve a 2D bin packing problem to assign a set of rectangular items into uniformly sized bins. A 3D bin packing problem [30] is studied for consolidating tasks using a differential evolutionary algorithm to increase container space utilization. A multi-objective optimization technique to solve 2D bin packing is proposed in [31], in which a particle swarm optimization algorithm is used to explore multiple favourable solutions. When the number of constraints

TABLE 1. List of notations.

Symbol	Definition
X	number of racks in a CDC
R_g	g^{th} rack (R), such that $\{g = 1...X\}$
Y	number of PMs in each R_g
PM_h^g	h^{th} PM in g^{th} rack, such that $\{h = 1...Y\}$
N	number of bins (VMs)
$B_i^{h,g}$	i^{th} bin in h^{th} PM in g^{th} rack, such that $\{i = 1...N\}$
M	number of MapReduce jobs (workloads)
J_j	j^{th} Job (J), such that $\{j = 1...M\}$
$BR_i^{v,u}$	denotes Bin Resource (BR) <vCPU(v), Memory (u)> available in $B_i^{h,g}$
$Task_j^{p,q}$	p and q denotes the number of map and reduce tasks of J_j
$Task_j^{r,status,i}$	status (running/finished) of r^{th} map task from J_j in $B_i^{h,g}$
$Task_j^{s,status,i}$	status (running/finished) of s^{th} reduce task from J_j in $B_i^{h,g}$
$AR_{i,j}^{v,u,r}$	denotes Allocated Resources (AR) $< v, u >$ for r^{th} map task from J_j in $B_i^{h,g}$
$AR_{i,j}^{v,u,s}$	$< v, u >$ for s^{th} reduce task from J_j in $B_i^{h,g}$
T_i	total number of tasks assigned in $B_i^{h,g}$
$Comb_{J_j} < n, m >$	map and reduce task combination of J_j

increases, possible search space decreases, which increases the complexity of finding an optimal solution. For real-time applications, tasks depend on data that is streamed, which eventually causes a gap between different task execution in heterogeneous resource clusters. To solve this problem, the authors [32] combine multiple imprecise computations for continuous execution by avoiding the execution gap as maximum as possible. Some works [33], [34] focus on varying the bin size to manage heterogeneity, while some works focus on varying the size of the workloads [35], [36] for placing them into homogeneous/heterogeneous bins. Bin packing is also widely used in mapping VMs to PMs to improve resource utilization. Various bin packing methods for VM placement are discussed in [37].

In summary, improving makespan and resource utilization are the major challenges in heterogeneous virtualized environments. Even though the approaches discussed in the literature use optimization algorithm for scheduling decisions at the MapReduce job level, there is a chance to improve makespan and resource utilization further at the task level by exploiting workload and VM level heterogeneities. However, MapReduce task scheduling is not like a general task scheduling that is based on the directed acyclic graph. Applying meta-heuristic optimization algorithms face a lot of constraints that limit the search space for exploring the solution for MapReduce task scheduling. To handle this problem, we model the MapReduce task scheduling problem as a constrained 2D bin packing model and solve using ACO to find the possible combinations of map and reduce tasks from different jobs in the batch and assign to VMs in the Hadoop virtual cluster.

III. 2-DIMENSIONAL BIN PACKING MODEL FOR MAP/REDUCE TASKS

Bin packing tasks is largely studied in a heterogeneous virtual environment for various applications. Our objective of this proposed method is to pack the right combination of MapReduce tasks from heterogeneous jobs into VMs (bins) based on heterogeneous VM capacity and container size to improve makespan and resource utilization using ACO. In this section, we initially define the problem statement and objective function, model bin packing problem for MapReduce task scheduling, and use the ACO to explore various possible combinations. Various notations used throughout this paper are listed out in Table 1.

A. PROBLEM DEFINITION

The 2D bin packing model for MapReduce task scheduling was first proposed by the authors in [11] to exploit the VM and workload level heterogeneities, using simple heuristics. In this paper, the same 2D bin packing model is modified further to adopt ACO. The model definition is ''scheduling right combination of map and reduce tasks from a batch of heterogeneous jobs onto heterogeneous bins to maximize virtual resource utilization and makespan'' is represented by Eq. 1.

$$\forall_{i,j}, Comb_{J_j} < nmap, mreduce > \rightarrow B_i^{h,g} \tag{1}$$

Here, n map tasks and m reduce tasks combinations ($Comb$) of j^{th} job (J) can be executed at any point of time on a bin ($B_i^{h,g}$) running on h^{th} PM mounted in g^{th} rack in CDC, where $Comb_{J_j}$ is an ordered pair. A $B_i^{h,g}$ might execute any combination of map and reduce tasks ($Comb_{J_1} < 5, 0 > \wedge Comb_{J_2} < 0, 2 > \wedge Comb_{J_3} < 0, 0 > \wedge \ldots \wedge Comb_{J_M} < 3, 0 >$) from different jobs. However, it is not certain that all the map and/or reduce tasks are executed in $B_i^{h,g}$ at any time. Because, if there are no data blocks to be processed by a job, map/reduce tasks of that particular job will not be included in this pair sequence.

B. OBJECTIVE FUNCTION

As the number of map tasks is typically huge, we initially discuss the bin packing model with only map tasks to find the right combination of map tasks of different jobs. This is

applicable for the reduce tasks and the combination of map and reduce tasks as well. The motivation of this problem is to improve virtual resource utilization while executing heterogeneous jobs on heterogeneous VMs. When we aim to improve resource utilization, makespan could be a tradeoff. So, we also focused on a fair share of resources between the jobs in a batch to reach optimal makespan, such that every job gets its turn to execute the tasks. Consequently, this may affect job completion time. For example, the shortest job in the batch may finish its execution after a long job is completed. Because a batch of jobs is submitted, individual job latency is ignored to improve resource utilization. However, minimizing the makespan is highly preferred. Firstly, we find the possible combinations <map, reduce> of J_j in every bin. For example, as shown in Fig. 3, consider 2 jobs (J_1, J_2), and 100 VMs, each with 4 containers. For simplicity, only map tasks of these two jobs are considered for finding the right combinations. So, possible combinations for jobs (J_1, J_2) in $B_i^{h,g}$ are $< 4, 0 >< 0, 0 >$, $< 3, 0 >< 1, 0 >$, $< 2, 0 >< 2, 0 >$, $< 1, 0 >< 3, 0 >$, and $< 0, 0 >< 4, 0 >$. One of these combinations is $< 1, 0 >< 3, 0 >$, which indicates the number of $< map, reduce >$ tasks from J_1 and J_2 in a bin. So, 1 map task and 0 reduce task from J_1, 3 map tasks and 0 reduce task from J_2 are possible in a bin. Such possible combinations are identified for each bin in the virtual cluster.

As we ensure fair sharing of resources among the jobs while bin packing, it guarantees the performance of the task scheduler in terms of makespan. Therefore, we take every combination from each bin (for instance, $< 4, 0 >< 0, 0 >$ from $B_1^{h,g}$, $< 2, 0 >< 2, 0 >$ from $B_2^{h,g}$, $< 1, 0 >< 3, 0 >$ from $B_3^{h,g}, \ldots, < 1, 0 >< 3, 0 >$ from $B_N^{h,g}$) and evaluate whether all the resources are completely utilized using Eq. 2, which comprises two components (*vCPU*, and *memory*) to calculate Resource Utilization of $B_i^{h,g}$ (RU_i). To find the utilization of *vCPU* in $B_i^{h,g}$, we find a ratio between the number of *vCPU* occupied ($\sum_{k=1}^{T_i} AR_k^v$) by all the tasks (T_i) running in $B_i^{h,g}$ at present and the total number of *vCPU* available in $B_i^{h,g}$ (BR_i^v). Similarly, we calculate the utilization of memory, as well. If any one of the resources is utilized very less, for example, 90% (*vCPU*) and 10% (memory), then RU_i is just 0.09 (0.9 x 0.1), which is not desired. Therefore, for all the combinations found in each bin, RU_i is calculated. Table 2 lists out the RU_i calculated for map task combinations from different jobs described in Fig. 1 for a VM that belongs to different VMF. Finally, we choose the combinations that result RU_i over 90%.

$$\forall_i, RU_i = \frac{\sum_{k=1}^{T_i} AR_k^v}{BR_i^v} \times \frac{\sum_{k=1}^{T_i} AR_k^u}{BR_i^u}, k \in Task_j^{k,running,i} \tag{2}$$

After calculating *RU* for each bin, our objective is to improve the virtual resource utilization in each bin using Eq. 3, and overall resource utilization in the virtual cluster using Eq. 4.

(i) Utilization_of_individual_bin=

$$Min(1 - RU_i) \tag{3}$$

(ii) Utilization_of_virtual_cluster=

$$Min \sum_{i=1}^{N} (1 - RU_i) \tag{4}$$

TABLE 2. Resource utilization in bins that belong to different VMF for all six jobs given in Fig. 1.

J_1	J_2	J_3	J_4	J_5	J_6	RU_i
$VMs \in VMF_1$						
1	0	0	0	0	0	100
0	0	0	0	1	0	100
$VMs \in VMF_2$						
2	0	0	0	0	0	100
1	0	0	0	1	0	100
0	0	1	0	1	0	93.75
1	0	1	0	0	0	93.75
$VMs \in VMF_3$						
4	0	0	0	0	0	100
2	0	0	0	2	0	100
1	0	0	0	3	0	100
0	0	0	0	4	0	100
0	0	1	0	3	0	96.88
3	0	1	0	0	0	96.88
2	0	1	0	1	0	96.88
3	1	0	0	0	0	93.75
0	1	1	0	2	0	90.62
$VMs \in VMF_4$						
8	0	0	0	0	0	100
0	0	0	0	8	0	100
5	0	0	0	0	2	100
0	0	0	0	0	6	100
0	0	1	0	7	0	98.44
0	2	5	0	2	0	98.44
5	1	0	0	2	0	96.88
4	0	3	0	1	0	95.31
1	1	2	0	4	0	93.75
5	1	2	0	0	0	93.75
2	1	3	0	2	0	92.19
1	2	2	0	3	0	90.62
3	2	1	0	2	0	92.19
$VMs \in VMF_5$						
0	0	0	0	0	6	100
6	0	0	0	0	4	100
0	0	0	5	2	2	98.96
0	4	0	5	2	0	98.96
0	3	0	1	5	2	96.88
8	0	3	0	1	0	96.88
4	1	2	0	5	0	95.83
5	0	4	0	3	0	95.83
3	2	1	0	4	1	90.62
1	2	1	0	6	1	90.62
4	4	1	0	3	0	90.62
1	2	0	1	7	0	90.62
0	3	0	0	7	1	90.58

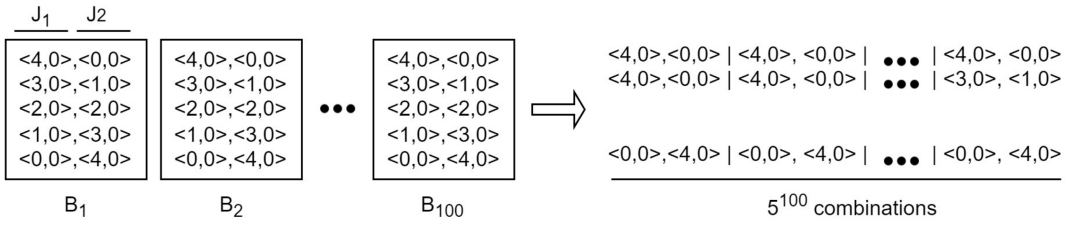

FIGURE 3. Number of map task combinations from different jobs.

TABLE 3. Maximum number of map tasks possible for each job in each VMF.

Job	VMF_1	VMF_2	VMF_3	VMF_4	VMF_5
1	1	2	4	8	12
2	1	2	4	8	12
3	1	2	4	8	12
4	0	1	2	4	6
5	1	2	4	8	12
6	0	1	2	4	6

Once the right combinations (that improve resource utilization) of map tasks from different jobs are identified, task related constraints are verified. For instance, data local execution is the desirable property for a map task, which is verified using Eq. 5. So, p^{th} map task of J_j ($Task_j^p$) in $Comb_{J_j}$ is checked whether $B_i^{h,g}$ has a right data block to process, such that maximum non-local executions are avoided to minimize the unnecessary local bandwidth consumption and job latency. Then, the fairness in resource sharing among jobs that are active in a batch is ensured using Eq. 6 at every C_t seconds. C_t is a variable constant, which can be set based on the expected latency of a job in the batch.

$$\forall_j, Comb_{J_j} \leftarrow \{\forall_{j,p}, Task_j^p \in B_i^{h,g}\ block\_location\} \tag{5}$$

$$\forall_j, \frac{\sum_{r=1}^{p} AR_j^v + \sum_{s=1}^{q} AR_j^v}{\sum_{i=1}^{N} B_i^v} \times \frac{\sum_{r=1}^{p} AR_j^u + \sum_{s=1}^{q} AR_j^u}{\sum_{i=1}^{N} B_i^u} \leq \frac{\sum_{i=1}^{N} B_i^v}{|J|} \times \frac{\sum_{i=1}^{N} B_i^u}{|J|} \tag{6}$$

The total ratio of $vCPU$ allocated for each job (p map tasks, and q reduce tasks) is calculated by using $\sum_{r=1}^{p} AR_j^v + \sum_{s=1}^{q} AR_j^v$ over total $vCPU$ in the virtual cluster ($\sum_{i=1}^{N} B_i^v$). Similarly, the ratio of memory used by each job is also calculated. Then, the outcome of multiplication of memory and $vCPU$ usage of each job should not exceed the equal share of resources for the jobs running in the cluster.

C. BIN PACKING MAP/REDUCE TASKS USING ANT COLONY OPTIMIZATION

Finding the right combinations of the map and reduce tasks from different jobs is less compute-intensive when there are less number of jobs and bins. If there are hundreds of bins and tens of MapReduce jobs, finding the right combinations of tasks for every bin involves a huge search space, which takes huge time to linearly evaluate. The number of task combinations in a bin depends on the size of the container and bins. For instance, as shown in Fig. 3, consider 100 bins, 2 MapReduce jobs, and 4 containers in each bin. In this case, each bin gets five different combinations. The combination $< 3, 0 >, < 1, 0 >$ in $B_i^{h,g}$ denotes that 3 map tasks, 0 reduce task from J_1 and 1 map task, 0 reduce task from J_2. If there are 100 bins, then 5^{100} combinations are possible in the search space, which is huge to linearly evaluate RU for every combination and find fair share among the jobs. But the scheduling decision is needed very quickly after a batch is submitted. Therefore, we use ant colony optimization (ACO) algorithm, which is a well-known and suitable meta-heuristic

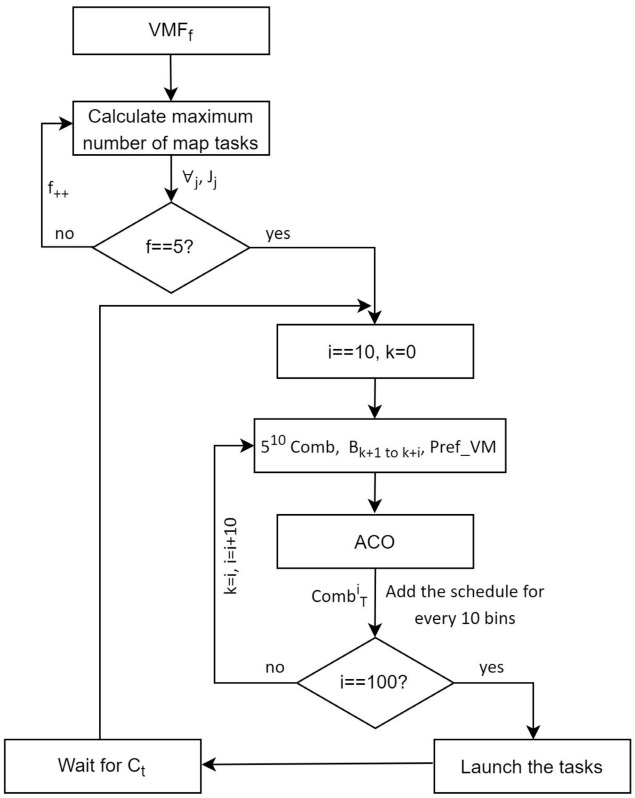

FIGURE 4. Finding tasks combinations using ACO.

Algorithm 1 Bin Packing Map/Reduce Tasks Using ACO

1) Get the information on workloads and bins.
2) Find different combinations of map/reduce tasks of different jobs that can be run in each bin.
   For map tasks
   a) Find the maximum number of map tasks of each job possible in each VMF (Table 3).

$$Maxpos_i^j = \forall_j, \frac{\sum AR_{j,r}^{v,u}}{BR_i^{v,u}} \tag{7}$$

   b) Find all possible combinations of map tasks from all the jobs that are currently active in the batch.
   c) Calculate RU_i (as in Table 2).
   d) Consider the top 5 combinations of map tasks in each bin ensuring data locality.

   For reduce tasks
   a) Map output in Racks (MOR):

$$\forall_g, MOR_g = \forall_{j,h}, \frac{\sum_{i=1}^{N}(\sum_{r=1}^{p} TaskOutput_j^r \in B_i^{h,g})}{\sum_{g=1}^{X}(\sum_{i=1}^{N}(\sum_{r=1}^{p} TaskOutput_j^r \in B_i^{h,g}))} \tag{8}$$

   b) Preferred rack (Pref_rack) to process reduce task: Pref_rack= $max(MOR)$
   c) Map Output in VMs (MOV) in Pref_rack:

$$\forall_i, MOV_i = \forall_{j,h}, \frac{\sum_{r=1}^{p} TaskOutput_j^r \in B_i^{h,PrefRrack}}{\sum_{i=1}^{N}\left(\sum_{r=1}^{p} TaskOutput_j^r \in B_i^{h,PrefRack}\right)} \tag{9}$$

   d) Preferred VM (Pref_VM) to process the reduce task: Pref_VM=sort_des_order(MOV)
   e) Consider the top 50% of bins from Pref_VM to launch reduce tasks.
3) Find the possible combinations $< map, reduce >$ tasks that belong to J_j in each bin using ACO.
   a) i = 10, k = 0
   b) Repeat untill $i! = 100$
      i) $\forall_j, Comb_{J_j} = ACO(B_{k+1\ to\ k+i}, 5^{10} combinations, Pref\_VM)$
      ii) k = i, i = i + 10
   c) end
4) Schedule the tasks.
5) Check for unused resources every C_t, and repeat Step 2 until all the jobs in a batch finish the execution.

search algorithm for huge discrete search space. As given in Algorithm 1, we initially find (Fig. 4) the maximum number of map tasks of each job possible in each VMF ($Maxpos_i^j$) using Eq. 7. We use only five VMFs in our experiment, as listed in Table 3, and it is fixed for a batch of jobs until its completion. Because, for any search space, we need to know the range within which we have to look for the solution. For instance, VMF_5 can execute up to 12, 12, 12, 6, 12, and 6 map tasks of $J_1, J_2, \ldots J_6$, respectively. Now, for each bin that belongs to different VMF finds different combinations of map tasks from different jobs, subsequently, RU_i are calculated, as given in Table 2. One possible combination of map tasks from different jobs in a bin that belongs to VMF_5 is (0, 0, 0, 5, 2, 2), for which RU_i is calcualted. Similarly, we calculate the resource utilization of all possible combinations and choose the combinations that result RU_i over 90%. Finally, we choose top five map task combinations from different jobs in each bin based on RU_i value.

For reduce tasks, we look for a set of VMs running in a specific rack. Firstly, we find a rack that might transfer more map output data among all the racks that host virtual cluster, using Eq. 7. As we cannot find the exact map output size in advance before all map tasks from different jobs, we just find the running total of map output size from map task's in-memory buffer. After finding a rack that could cause more map task output, we find a set of VMs in the rack that will produce more map output, in the same way, using Eq. 7. After this, the right combination of $< map, reduce >$ is found for each job. If the early reduce feature is enabled, before all the map tasks of a job is over, all the reduce tasks are launched. Therefore, we need to include reduce tasks also in finding the right combinations (as given in ordered pair) that result in considerable resource utilization. This way, we minimize the inter-rack bandwidth consumption after launching reduce tasks. In general, every job executes numerous map tasks as the input dataset is huge. Therefore, most of the time, only

Algorithm 2 ACO for Finding the Schedule

Function ACO ($B_{k+1\ to\ k+i}$, 5^{10}, *Pref_VM*) **:**
Initialization:
- Assign pheromone values on 5×5 edges between each bin from $k+1$ to $k+i$.
- Other algorithm control parameters, such as number of ants, exploration, and exploitation are initialized.

while *NOT end* **do**
1) Path construction between each bin from $k+1$ to $k+i$
2) Ants generation
3) Mapping ants with path
4) Evaluate the objective function $Min(1-RU_i)$ AND $Min\sum_{i=1}^{N}(1-RU_i)$
5) Fitness value calculation
6) Local pheromone update
7) Global pheromone update

end
Return the optimal schedule

map task combinations of different jobs are considered at the initial stage.

After finding the possible bins to launch the map and reduce tasks, as shown in Fig. 4, we invoke ACO, which runs Algorithm 2, for every 10 bins, sequentially, to find the right combination of map/reduce tasks to maximize the individual bin resource utilization. For instance, $< 4, 0 >< 0, 0 >$ from $B_1^{h,g}$, $< 2, 0 >< 2, 0 >$ from $B_2^{h,g}$, $< 1, 0 >< 3, 0 >$ from $B_3^{h,g}$, etc. In order to ensure a fair share of resources among jobs, we need to evaluate all 5^{100} combinations if the number of bins is 100. Therefore, we break down the combinations by taking every 10 bins to find the fair share, which will cause only 5^{10} combinations. This takes very less time to find a perfect schedule, part by part. We consider every 10 bins for finding the combinations because the algorithm should not face huge search space nor run more iterations. In Algorithm 2, bins and task combination details, such as $B_{k+1\ to\ k+i}$, 5^{10} combinations, *Pref_VM* are received as input. Final combinations (schedule for the current 10 bins) are sent back to Algorithm 1.

To begin with ACO, a pheromone value matrix is generated that records a value between 0 to 1. Between every bin from $k+1$ to $k+i$, 25 edges (5×5) are possible, on which the pheromone is deposited. Other algorithm related parameters such as the number of ants, exploration, and exploitation are initialized. Then, the probability value is calculated using the pheromone matrix to construct the path matrix between every bin from $k+1$ to $k+i$. After this, ants (a real number between 0 to 1) are generated on the edges (path). Each ant is mapped with the path constructed and decided whether to choose a particular combination from each bin pair. Once pairs are identified, objective functions are evaluated using the input values given to the algorithm to choose the right combination. Then, the fitness value is calculated to decide whether the current solution should be considered for the next iteration. Based on the fitness value, local and global pheromones are updated. This is repeated until the solution does not change or desired number of times. Finally, an optimal schedule for the current 10 bins is obtained and continued until all the bins (100 here) are evaluated. This whole process is repeated whenever a batch of jobs is submitted or every C_t seconds to find the resource availability and schedule tasks that are yet to be executed.

IV. RESULTS AND ANALYSIS

A. EXPERIMENTAL SETUP

To showcase the efficiency and effectiveness of our proposed bin packing model for MapReduce task scheduling (ACO-BP), we modified classical FS and compared it with the default fair scheduler (FS) [10], RWS and HMJS [11], for a batch of MapReduce jobs (wordcount (J_1), wordmean (J_2), word standard deviation (J_3), kmean (J_4), sort (J_5), and join (J_6)) on a heterogeneous virtualized environment. To compare and contrast, we used several parameters, such as latency, makespan, non-local executions, and finally utilization of vCPU and memory. To observe the working behaviour of our proposed method that deals with heterogeneous environments and workloads, we implemented our simulator framework in Java [38] and executed it on a server with 12 core hyper-threaded and 32 GB memory. The simulator is run several times and the median results are presented in this section. All the CDC and workload related parameters and their values are considered as in Fig. 1. However, values for algorithm, workload and CDC related parameters specified in this paper can be customized.

1) CDC RELATED PARAMETERS

We assumed the CDC related parameters, such as a number of racks, PMs, VMs, VMFs, and network bandwidth capacity for inter and intra-rack in CDC. We set 50 PMs in CDC with different configuration settings as an ordered pair <clock_rate in GHz, number of cores, memory size in GB>. Based on this, 15 PMs $< 1.9, 6, 32 >$, 5 PMs $< 2.4, 28, 128 >$, 15 PMs $< 3.2, 6, 16 >$, and 15 PMs $< 3.4, 4, 16 >$ are considered. These 50 PMs with different configurations and performances exhibit the heterogeneity at the hardware level in CDC. On top of this hardware set, we used 20 VMs (bins) from each VMF (types) to establish Hadoop virtual cluster and distributed them across CDC. $< vCPU, memory >$ configuration of $VMF_1,\ldots,VMF_5$ are $< 1, 2 >$, $< 2, 4 >$, $< 4, 8 >$, $< 8, 16 >$, and $< 12, 24 >$, respectively. In addition, it is assumed that there is no interference for Hadoop VMs by non-Hadoop VMs in the simulation.

2) WORKLOAD RELATED PARAMETERS

We used the values for workload related parameters, such as the number of map/reduce tasks and resource requirements for simulation, as given in Fig. 1. There are six workloads

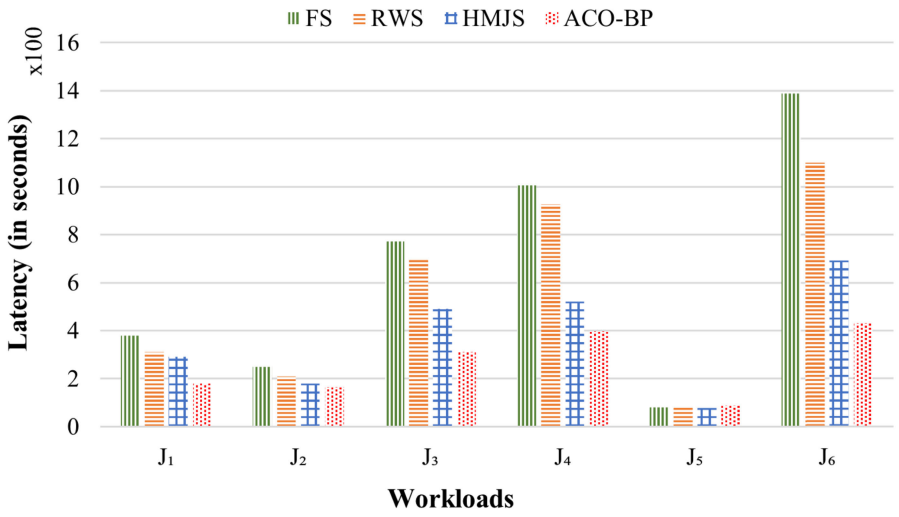

FIGURE 5. Job latency with FS, RWS, HMJS, and ACO-BP.

$J_1, J_2, \ldots J_6$ (MapReduce jobs) in a batch used for the experiment. The resource requirement of each workload varies, therefore container size is highly heterogeneous for the map and reduce tasks from different jobs. A container is represented by <vCPU, memory in GB> and the containers for map and reduce task are denoted as (<map container>, <reduce container>) for each job. Based on this, $(< 1, 2 >, < 1, 1 >)$, $(< 1, 1 >, < 2, 1 >)$, $(< 1, 1.5 >, < 2, 2 >)$, $(< 2, 1.5 >, < 2, 2.5 >)$, $(< 1, 2 >, < 0, 0 >)$, and $(< 2, 2 >, < 1, 1 >)$ size of containers are used for $J_1, J_2, \ldots J_6$, respectively. Each workload is given different dataset, whose sizes are 128, 64, 256, 192, 76.8, 153.6 in GBs. Default block size (128 MB) is used to distribute input dataset into HDFS cluster with default replication factor 3. Based on the size of each dataset and block size, the number of map tasks is decided. We set the number of reduce tasks based on the nature of the job. Hence, the number of (map tasks, reduce tasks) for each job is (1000, 20), (500, 15), (2000, 50), (1500, 10), (600, 0), and (1200, 5), respectively. Similarly, the latency of (map, reduce) tasks are approximately (7, 13), (6, 11), (5, 10), (7, 15), (6, 0), and (13, 27), seconds respectively. These latencies are the observed units from the real execution we conducted and fixed the same for our simulation.

3) ALGORITHM RELATED PARAMETERS

a: BIN PACKING PARAMETERS

We considered only *vCPU* and memory (2D) to pack map and reduce tasks from different jobs in a batch. Besides packing them into bins, typical map and reduce task-related constraints are preserved. For instance, a map task should adhere to data locality and reduce tasks are to be hosted in racks that minimize inter-rack bandwidth consumption. In addition, fair resource sharing is ensured every C_t seconds. It is set to 20, as map/reduce tasks are launched every 20 seconds based on the resource availability observed. In addition, ACO finds an optimal schedule for every 10 bins, and incrementally, adds the schedule up to 100 bins (as specified in this article). However, it is tunable to any whole number that is not huge to process, based on the problem settings.

b: ACO PARAMETERS

The number of ants (*no_ants*) and iterations (*no_iter*) used in the ACO algorithm are 10 and 50, respectively. We use the default local and global pheromone update rule [39] to emphasize the ant's exploration. Pheromone decay factor (ρ) in local and global pheromone update rule controls the intensification and diversification behaviour of ants to explore the search space. The parameters, *no_ants* and *no_iter* are fixed with the trial and error method, while ρ is set to 0.5 to balance intensification and diversification, as we divide the search space into small boundaries.

B. ACO BASED BIN PACKING MAP/REDUCE TASKS

Fig. 5 shows the latency of each job with four methods, FS, RWS, HMJS, and ACO-BP. As expected, ACO-BP minimized the job latency up to 53%, 34%, 60%, 69%, and 44% for J_1, J_2, J_3, J_5, and J_6, respectively, when compared to classical FS. It also outperformed the RWS scheduler with 42%, 22%, 55%, 57%, and 39% improvement for J_1, J_2, J_3, J_4, and J_6, respectively, which is very significant to consider. Similarly, ACO-BP achieved prominent results over HMJS, which outperformed RWS and FS while dealing with heterogeneous workloads in a heterogeneous environment. The improvement of ACO-BP over HMJS is considerably better with 38%, 9%, 36%, 24%, 37%, and 22% for J_1, J_2, J_3, J_5, and J_6, respectively. Surprisingly, a 13% increase in job latency is observed for J5. It is because J5 contains only 600 map tasks and does not have to reduce tasks to execute. As jobs are submitted as a batch, reduce tasks of all other

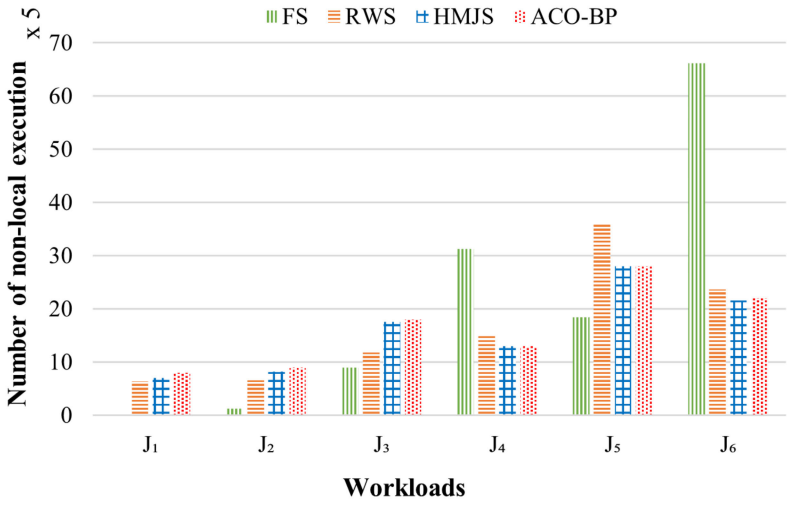

FIGURE 6. Number of non-local executions with FS, RWS, HMJS, and ACO-BP.

jobs are scheduled during the execution of map tasks. This prevents the shortest job that contains only 600 map tasks to delay its execution. Altogether, ACO-BP minimized the job latency up to 44%, 38%, and 22% over FS, RWS, and HMJS, respectively, on average.

In addition to scheduling the map tasks, data locality is also ensured. Fig. 6 exhibits the number of non-local executions for map tasks of different jobs with all four schedulers. Considering the results obtained, ACO-BP minimized the number of non-local executions up to 24.3% and 1.4%, compared to the FS and RWS. The FS focuses on sharing resources among the jobs, which eventually compromised the data-local executions, which account for the maximum number of non-local executions. On the other hand, RWS places the data blocks based on the computing capacity of each bin, which restricts fair sharing of the resources to a certain extent. Therefore, in the tug of war between fair sharing and data locality, the number of non-local executions minimized is not significant over RWS. In contrast, HMJS considers a set of heuristics, such as processor performance, amount of resources utilized or remaining in the bins, it is still possible to minimize the number of non-local executions compared to RWS. Therefore, the performance of ACO-BP dropped up to 3% when compared to HMJS. Because ACO-BP attempts to find the perfect combination of map and reduce tasks, with data local execution as a primary constraint for map tasks that belongs to network-intensive jobs. Hence, improvement in number of non-local executions were not beyond the expectation when compared to HMJS, because of trying to achieve a fair share among the jobs and resource utilization, which are partially contradicting the objectives.

Jobs J_1, J_2, and J_3 are highly network intensive, as they require to transfer of all the map task output to reduce tasks. In this case, the input size of reduce tasks is almost equivalent or higher to the size of job input. Hence, pushing the output of

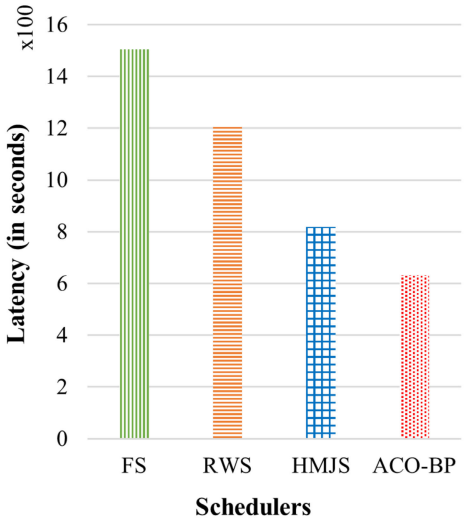

FIGURE 7. Throughput.

all map tasks into a less number of reduce tasks require a lot of local network bandwidth to transfer. So, for reduce tasks, latency minimization is realized when the expected amount of bandwidth consumption between racks, and between PMs (that host VMs) in each rack, as given in Algorithm 1. These factors ultimately help improve the makespan of the proposed method. When compared to FS, RWS, and HJMS, ACO-BP minimized the makespan for the batch of jobs considered, up to 60%, 48%, and 23%, respectively, as shown in Fig. 7. The ultimate objective of ACO-BP is to improve the virtual resource utilization by forming the optimal combinations of map and reduce tasks that belong to different jobs in the batch. As shown in Table 2, any one of the combinations in each bin is considered to pack the map/reduce tasks, based

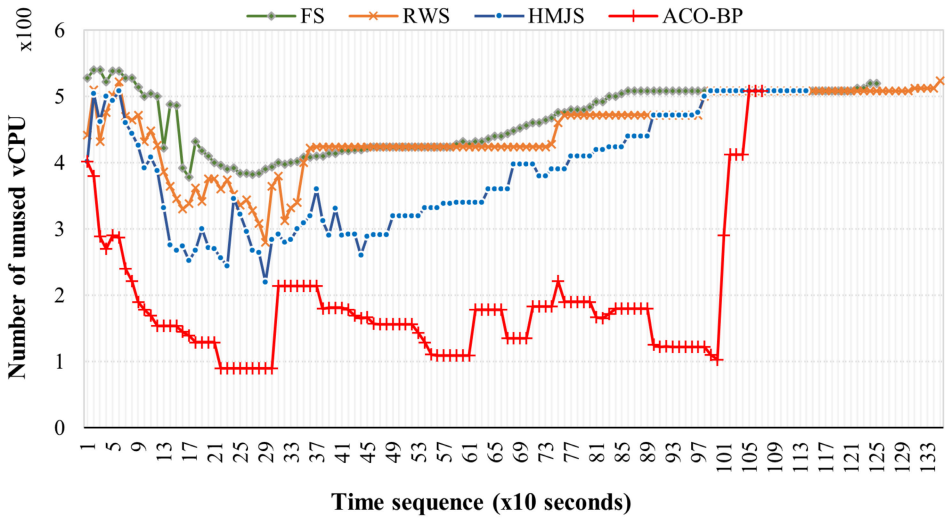

FIGURE 8. Unused vCPU over time.

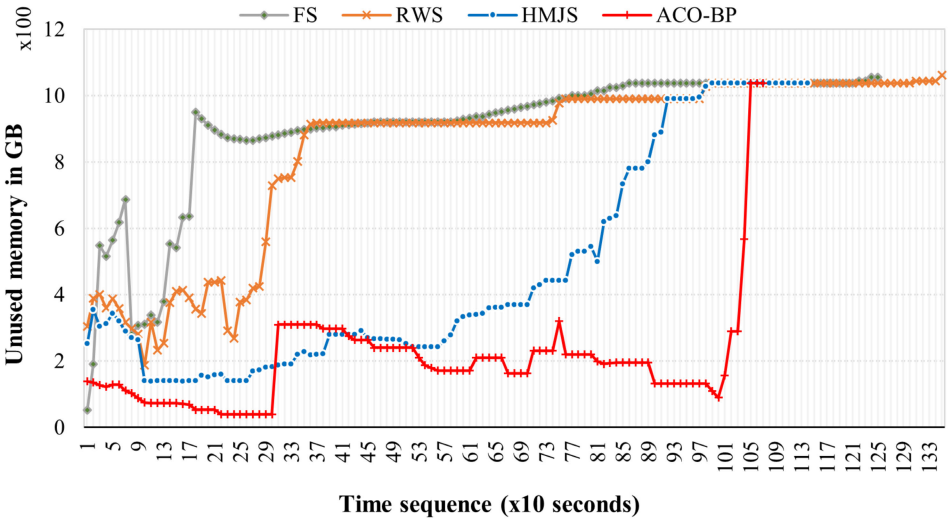

FIGURE 9. Unused memory over time.

on the resource availability every 20 seconds. This causes ACO-BP to come up with different possible schedules by relaxing the actual fair share to obtain a better combination of tasks. In a production environment, a MapReduce job typically contains thousands of map tasks and a very less number of reduce tasks, sometimes only map tasks are used in jobs for pre-processing the data. Therefore, most of the task combinations in the final schedule are only map tasks from different jobs. Therefore, we discuss the results of our proposed model based on map task combinations. To trace the resource usage, we recorded utilization of vCPU and memory in the Hadoop virtual cluster every 10 seconds with all four schedulers, as shown in Fig. 8, and Fig. 9.

The FS shares the resources among all the jobs that are active. Therefore, regardless of the map and reduce tasks in jobs, based on the resource quota, tasks are assigned. In the heterogeneous environment, where the resource requirements of tasks, underlying VM configuration (VMF), and physical resource performance, FS has failed to provide better resource utilization. On the other hand, RWS focuses on placing the number of data blocks based on the resource processing capacity of the individual bin. It means that a bin that contains low processing power attracts less number of tasks and vice versa. Therefore, it is slightly better than FS in minimizing latency, makespan and maximizing resource utilization. Similarly, HMJS uses runtime heuristics about the underlying physical and virtual resources, which improves the performance and resource utilization compared to HMJS. As ACO-BP finds the right combination of map and reduce tasks in each bin, it promises optimal resource utilization

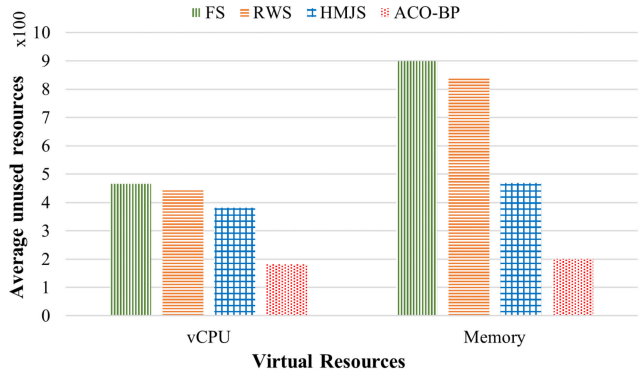

FIGURE 10. Average unused resources (vCPU and memory) after all the jobs completed.

at any point in time during execution. At times, to keep the resources busy, some non-local execution is performed, which indirectly affects the job latency. However, our proposed model largely minimized the number of idle resources (vCPU and memory) of the entire virtual cluster during execution, compared to other schedulers taken for comparison. As shown in Fig. 10, on average, it minimized unused vCPU up to 60%, 59%, and 52%, and unused memory up to 77%, 75%, and 57%, when compared to FS, RWS, and HMJS, respectively. To improve resource utilization using the bin packing model, sometimes it is required to compromise with the number of non-local execution, which, however, does not degrade the makespan.

V. CONCLUSION

Hadoop MapReduce on the cloud using virtual clusters is nowadays increasingly being used for various real-world applications like transportation monitoring, advertising, marketing, banking, etc. Even though the resources are scalable on-demand in the cloud, there is no guarantee that the hired virtual resources are completely utilized. In addition, various heterogeneities (hardware, VM, performance, workload) are realized between the underlying hardware and task scheduling. These heterogeneities result in vast resource underutilization in the hired virtual cluster. Motivated by this, to improve the virtual resource utilization, we introduced a constrained 2D bin packing model using ACO to find the right combination of map and reduce tasks from different MapReduce jobs. As expected, it significantly minimized the unused vCPU up to 60%, 59%, and 52%, and unused memory up to 77%, 75%, and 57%, when compared to FS, RWS, and HMJS, respectively. As part of future work, it is possible to think of energy consumption as another dimension to improve the scheduler performance, as it is also highly dynamic based on the workload and underlying resource capacity. Therefore, dynamic energy consumption can also be considered in addition to the layer of heterogeneities mentioned in this paper.

REFERENCES

[1] D.-H. Shin, "Demystifying big data: Anatomy of big data developmental process," *Telecommun. Policy*, vol. 40, no. 9, pp. 837–854, Sep. 2016, doi: 10.1016/j.telpol.2015.03.007.

[2] S. G. Jeffrey Dean, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 2140–2144, 2008, doi: 10.1145/1327452.1327492.

[3] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of 'big data' on cloud computing: Review and open research issues," *Inf. Syst.*, vol. 47, pp. 98–115, Jan. 2015, doi: 10.1016/j.is.2014.07.006.

[4] 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, doi: 10.1007/s11227-019-02960-0.

[5] S. Bardhan and D. A. Menasce, "The anatomy of mapreduce jobs, scheduling, and performance challenges," in *Proc. Annu. Int. Conf. Comput. Meas.*, vol. 1, 2013, pp. 619–631.

[6] Z. Zhang, L. Cherkasova, and B. T. Loo, "Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 4, pp. 38–50, Jun. 2015, doi: 10.1145/2788402.2788409.

[7] 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, doi: 10.1007/s13174-011-0054-7.

[8] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informat. J.*, vol. 16, no. 3, pp. 275–295, 2015, doi: 10.1016/j.eij.2015.07.001.

[9] X.-S. Yang and M. Karamanoglu, *Nature-Inspired Computation and Swarm Intelligence: A State-of-the-Art Overview*. Amsterdam, The Netherlands: Elsevier, 2020.

[10] *Fair Scheduler*. Accessed: Mar. 20, 2022. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html

[11] R. Jeyaraj, V. S. Ananthanarayana, and A. Paul, "Improving mapreduce scheduler for heterogeneous workloads in a heterogeneous environment," *Concurrency Comput., Pract. Exper.*, vol. 32, no. 7, Apr. 2020, Art. no. e05558, doi: 10.1002/cpe.5558.

[12] K. Kalia and N. Gupta, "Analysis of Hadoop mapreduce scheduling in heterogeneous environment," *Ain Shams Eng. J.*, vol. 12, no. 1, pp. 1101–1110, Mar. 2021, doi: 10.1016/j.asej.2020.06.009.

[13] G. Singh, A. Sharma, R. Jeyaraj, and A. Paul, "Handling non-local executions to improve mapreduce performance using ant colony optimization," *IEEE Access*, vol. 9, pp. 96176–96188, 2021, doi: 10.1109/ACCESS.2021.3091675.

[14] N. S. Naik, A. Negi, B. R. T. Bapu, and R. Anitha, "A data locality based scheduler to enhance mapreduce performance in heterogeneous environments," *Future Gener. Comput. Syst.*, vol. 90, pp. 423–434, Jan. 2019, doi: 10.1016/j.future.2018.07.043.

[15] X. Zeng, S. K. Garg, Z. Wen, P. Strazdins, A. Y. Zomaya, and R. Ranjan, "Cost efficient scheduling of MapReduce applications on public clouds," *J. Comput. Sci.*, vol. 26, pp. 375–388, May 2018, doi: 10.1016/j.jocs.2017.07.017.

[16] 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, doi: 10.1007/s11227-020-03162-9.

[17] 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, doi: 10.1109/TCC.2017.2748586.

[18] Y. Xu, W. Chen, S. Wang, X. Zhou, and C. Jiang, "Improving utilization and parallelism of Hadoop cluster by elastic containers," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 180–188, doi: 10.1109/INFOCOM.2018.8486400.

[19] T. Gouasmi, W. Louati, and A. H. Kacem, "Exact and heuristic mapreduce scheduling algorithms for cloud federation," *Comput. Electr. Eng.*, vol. 69, pp. 274–286, Jul. 2018, doi: 10.1016/j.compeleceng.2018.01.021.

[20] V. Seethalakshmi, V. Govindasamy, and V. Akila, "Real-coded multi-objective genetic algorithm with effective queuing model for efficient job scheduling in heterogeneous Hadoop environment," *J. King Saud Univ.-Comput. Inf. Sci.*, Aug. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157820304298, doi: 10.1016/j.jksuci.2020.08.003.

[21] V. Pandey and P. Saini, "A heuristic method towards deadline-aware energy-efficient mapreduce scheduling problem in Hadoop YARN," *Cluster Comput.*, vol. 24, no. 2, pp. 683–699, Jun. 2021, doi: 10.1007/s10586-020-03146-7.

[22] R. Alanazi, F. Alhazmi, H. Chung, and Y. Nah, "A multi-optimization technique for improvement of Hadoop performance with a dynamic job execution method based on artificial neural network," *Social Netw. Comput. Sci.*, vol. 1, no. 3, pp. 1–11, May 2020, doi: 10.1007/s42979-020-00182-3.
[23] N. Zacheilas and V. Kalogeraki, "A Pareto-based scheduler for exploring cost-performance trade-offs for mapreduce workloads," *EURASIP J. Embedded Syst.*, vol. 2017, no. 1, Dec. 2017, doi: 10.1186/s13639-017-0077-7.
[24] Y. Li, X. Tang, and W. Cai, "Dynamic bin packing for on-demand cloud resource allocation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 157–170, Jan. 2016, doi: 10.1109/TPDS.2015.2393868.
[25] Z. Wang, "Variable size vector bin packing heuristics–application to the machine reassignment problem," *Hal*, vol. 6025, pp. 16–29, May 2014.
[26] J. O. Iglesias, M. De Cauwer, D. Mehta, B. O'Sullivan, and L. Murphy, "Increasing task consolidation efficiency by using more accurate resource estimations," *Future Gener. Comput. Syst.*, vol. 56, pp. 407–420, Mar. 2016, doi: 10.1016/j.future.2015.08.018.
[27] Y. Shao, C. Li, J. Gu, J. Zhang, and Y. Luo, "Efficient jobs scheduling approach for big data applications," *Comput. Ind. Eng.*, vol. 117, pp. 249–261, Mar. 2018, doi: 10.1016/j.cie.2018.02.006.
[28] A. Stawowy, "Evolutionary based heuristic for bin packing problem," *Comput. Ind. Eng.*, vol. 55, no. 2, pp. 465–474, Sep. 2008, doi: 10.1016/j.cie.2008.01.007.
[29] C. Blum and V. Schmid, "Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm," *Proc. Comput. Sci.*, vol. 18, pp. 899–908, Jan. 2013, doi: 10.1016/j.procs.2013.05.255.
[30] X. Li and K. Zhang, "A hybrid differential evolution algorithm for multiple container loading problem with heterogeneous containers," *Comput. Ind. Eng.*, vol. 90, pp. 305–313, Dec. 2015, doi: 10.1016/j.cie.2015.10.007.
[31] D. S. Liu, K. C. Tan, S. Y. Huang, C. K. Goh, and W. K. Ho, "On solving multiobjective bin packing problems using evolutionary particle swarm optimization," *Eur. J. Oper. Res.*, vol. 190, no. 2, pp. 357–382, Oct. 2008, doi: 10.1016/j.ejor.2007.06.032.
[32] G. L. Stavrinides and H. D. Karatza, "Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes," *Future Gener. Comput. Syst.*, vol. 28, no. 7, pp. 977–988, Jul. 2012, doi: 10.1016/j.future.2012.03.002.
[33] C. Paquay, S. Limbourg, and M. Schyns, "A tailored two-phase constructive heuristic for the three-dimensional multiple bin size bin packing problem with transportation constraints," *Eur. J. Oper. Res.*, vol. 267, no. 1, pp. 52–64, May 2018, doi: 10.1016/j.ejor.2017.11.010.
[34] C. Bassem and A. Bestavros, "Multi-capacity bin packing with dependent items and its application to the packing of brokered workloads in virtualized environments," *Future Gener. Comput. Syst.*, vol. 72, pp. 129–144, Jul. 2017, doi: 10.1016/j.future.2016.08.017.
[35] C. Liu and S. Baskiyar, "Scheduling mixed tasks with deadlines in grids using bin packing," in *Proc. 14th IEEE Int. Conf. Parallel Distrib. Syst.*, Dec. 2008, pp. 229–236, doi: 10.1109/ICPADS.2008.127.
[36] Z.-H. Jia and J. Y.-T. Leung, "A meta-heuristic to minimize makespan for parallel batch machines with arbitrary job sizes," *Eur. J. Oper. Res.*, vol. 240, no. 3, pp. 649–665, Feb. 2015, doi: 10.1016/j.ejor.2014.07.039.
[37] S. Kumaraswamy and M. K. Nair, "Bin packing algorithms for virtual machine placement in cloud computing: A review," *Int. J. Elect. Comput. Eng.*, vol. 9, no. 1, pp. 512–524, 2019, doi: 10.11591/ijece.v9i1.pp512-524.
[38] R. Jeyaraj. *Simulator*. Accessed: Mar. 21, 2022. [Online]. Available: https://github.com/rathinaraja/DBP
[39] M. Randall and E. Tonkcs, "Intensification and diversification strategies in ant colony system," *Complex. Int.*, vol. 9, pp. 1–7, Jan. 2002.

RATHINARAJA JEYARAJ received the Ph.D. degree from the Department of Information Technology, National Institute of Technology Karnataka, 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 cloud computing, big data, the IoT, and data science. Personal website: https://jrathinaraja.co.in/

ANAND PAUL (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from 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 is a Delegate representing South Korea for M2M Focus Group and for MPEG. His research interests include algorithm and architecture re-configurable embedded computing. He was a guest editor of various international journals.