RENDA: Resource and Network Aware Data Placement Algorithm for Periodic Workloads in Cloud

Hiren Kumar Thakkar, *Member, IEEE*, Prasan Kumar Sahoo, *Senior Member, IEEE*, and Bharadwaj Veeravalli, *Senior Member, IEEE*

Abstract—The Hadoop enabled cloud platforms are gradually becoming preferred computational environment to execute scientific big data workloads in a periodic manner. However, it is observed that the default data placement approach of such cloud platforms is not the efficient one and often ends up with significant data transfer overhead leading to degradation of the overall job completion time. In this article, a Resource and Network-aware Data Placement Algorithm (RENDA) is proposed to reduce the non-local executions and thereby reduce the overall job completion time for periodic workloads in the cloud environment. The entire job execution is modeled as a two-stage execution characterized as data distribution and data processing. The RENDA reduces the time of the stages as mentioned above by estimating the heterogeneous performance of the nodes on a real-time basis followed by careful allocation of data in several installments to participating nodes. The experimental results show that the proposed RENDA algorithm consistently outperforms over the recent state-of-the-art alternatives with as much as 28 percent reduction in data transfer overhead leading to 16 percent reduction in average job completion time with 27 percent average speedup on average job execution.

Index Terms—MapReduce, cloud computing, data placement, periodic workloads

1 Introduction

In the era of high-speed internet, the usage of web applications has increased many folds leading to the generation of a massive amount of data daily [1]. Among the parallel processing models, cloud computing is popular due to its simplicity, scalability, and fault tolerance characteristics, which has encouraged organizations such as Facebook to adopt the open-source implementation Apache Hadoop to build the big data analytic cloud platforms [2]. Usually, Hadoop enabled cloud platforms scale from a few machines to thousands of machines, also known as nodes.

The default data placement scheme of Hadoop assumes homogeneous computing resources of nodes and distributes an equal amount of input data among the nodes for the load balancing [3]. However, the default Hadoop scheme triggers a significant number of non-local executions in heterogeneous cloud platforms [4]. The non-local execution means the data processing happens in the node where data are not present and need to be brought from other nodes. The non-local executions result from inefficient data distribution, whereby the nodes with higher computing resources finish the data processing earlier and become idle. To maximize resource utilization, the data placement scheme needs to transfer the excess data from the sluggish nodes to the efficient nodes.

Several real-life applications generate huge amount of data in a periodic manner such as weblog analysis [5], distributed grep [6], reverse web-link graph [7], URL access frequency count [8] etc. Such applications can be easily expressed as MapReduce (MR) jobs [9], which enable their parallel execution in a cloud environment. Each MR job completes the execution in sequential phases known as a map and a reduce. The dedicated cloud machine known as a master node partitions the input MR job in equal-sized data blocks and distributes them onto the slave nodes. Later, in a map phase, map tasks are scheduled to the nodes, also known as mappers to generate the intermediate data. In a reduce phase, selective nodes, also called reducers collect the intermediate data from the mappers and generate the final output.

The Hadoop architecture is comprised of two fundamental stages, such as data distribution and data processing. The data distribution refers to the time master node takes to distribute the input data blocks to slave nodes. The data processing refers to the time slave nodes take to process the assigned input data blocks. In default Hadoop enabled cloud; job completion is the sequential execution of data distribution and data processing stages, as shown in Fig. 1. However, such sequential execution compulsion is not efficient, and it affects the job completion time. For instance,

- *Hiren Kumar Thakkar is with the Department of Computer Science and Engineering, SRM University, Andhra Pradesh 522240, India. E-mail: hirenkumar.t@srmap.edu.in.*
- *Prasan Kumar Sahoo is with the Department of Computer Science and Information Engineering, Chang Gung University, Guishan 333, Taiwan, and also with the Department of Neurology, Chang Gung Memorial Hospital, Linkou 33305, Taiwan. E-mail: pksahoo@mail.cgu.edu.tw.*
- *Bharadwaj Veeravalli is with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077 Singapore. E-mail: elebv@nus.edu.sg.*

Manuscript received 13 Mar. 2020; revised 27 Mar. 2021; accepted 29 Apr. 2021. Date of publication 14 May 2021; date of current version 3 June 2021. (Corresponding author: Prasan Kumar Sahoo.) Recommended for acceptance by Q. Zheng. Digital Object Identifier no. 10.1109/TPDS.2021.3080582

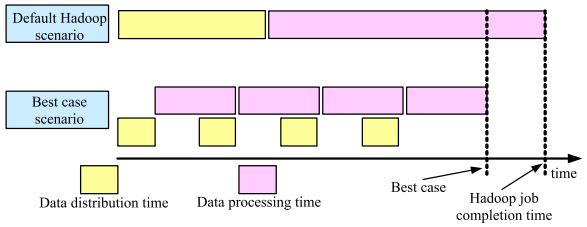


Fig. 1. Job completion time.

the data distribution stage takes longer to complete if the input data size is large or network resources are insufficient. The possible best-case scenario is presented in Fig. 1, where the data distribution is carried out in several installments concurrent with the data processing resulting in the reduction of the overall job completion time.

The data distribution without assessing the computing resources and network delay may trigger several non-local executions. The non-local executions increase the network traffic and therefore incur the data transfer overhead, which negatively impacts the overall job completion time. Our experiment on a 16 nodes virtual Hadoop cloud shows that data distribution takes as much as 48 percent of the total job completion time. Hence, it is highly essential to reduce the data distribution time and the number of non-local executions to reduce the overall job completion time. For simplicity, the terms *data* and *data blocks* are used interchangeably throughout the study.

In this paper, we design a Resource and Network-aware Data Placement Algorithm (RENDA) to reduce the data transfer overhead in the cloud and thereby to reduce the overall job completion time.

Our contributions are summarized as follow,

- The Hadoop enabled cloud's data placement problem is formulated as a sequential execution of data distribution and data processing stages.
- The major pitfalls in the existing data placement schemes are described.
- A Resource and Network-Aware Data Placement (RENDA) algorithm is designed by estimating the cloud nodes' performance on-the-fly.
- Extensive performance evaluation of RENDA is carried out against the state-of-the-art alternatives.

The scope of this work is mainly addressing the above-mentioned pitfalls and focuses on designing an efficient, in the sense of including the resource and network awareness, data placement strategy, conducting rigorous analysis, and to evaluate its performance on actual cloud platforms. The evaluation is also carried out by comparing our strategy with most commonly available and relevant strategies in the literature. The paper has all the four important contributions namely, design, analysis, performance evaluation and actual realization on a Cloud platform employing both the physical and virtual nodes.

The rest of the paper is organized as follows. In Section 2, related works are described. The problem statement is discussed in Section 3 followed by the pitfalls in existing schemes are described in Section 4. The proposed RENDA scheme is described in Section 5. The experimental results are presented in Section 6 and concluding remarks are made in Section 7.

2 Related Works

Divisible Load Theory (DLT) refers to the scheduling of large-scale partitionable workloads on networked computing platforms. Effective distribution of such partitionable workloads on compute nodes makes it possible for their rapid processing. Large-scale image processing [10], polynomial multiplication [11], and stream data processing [12] are few of the examples of the divisible workloads. In Hadoop enabled cloud, the divisible workloads are distributed in a single installment. In other words, the divisible workloads also referred to as the data blocks are distributed to the compute nodes in a single shot. However, in a heterogeneous cloud data center, workload distribution in a single installment may violate the "Optimality principle" [13], which states that compute nodes should conclude the workload processing simultaneously to ensure the shortest workload completion time. To overcome the shortcomings of single installment distribution, in [14], [15] multi-installment strategies are proposed. However, deriving an optimal multi-installment model is challenging as the model needs to capture all possible overheads.

Several proposals are made to address the workload distribution issues of a Hadoop cloud [3], [16], [17], [18], [19]. In [16], a data placement algorithm is introduced to balance the distribution of intermediate data of MR jobs. However, balancing the intermediate data is not sufficient to reduce the job completion time. In [18], a parallel distributed file system "Lustre" is proposed to benefit shuffle-intensive workloads of MR jobs. However, [18] may not efficiently work for map-intensive and reduce-intensive periodical workloads. In [19], a heuristic data placement algorithm is designed considering the topology of the nodes to cut down the network traffic during the MR job execution. However, fluctuating cloud performance makes it challenging to predict the optimum data layout leading to unexpected non-local executions. In [17], a genetic optimization algorithm is designed for the replica distribution in Hadoop.

Considering the impact of data placement on cloud performance, various data placement approaches are proposed in the past [4], [20], [21], [22], [23], [24]. In [21], an application-specific data placement scheme is introduced, which is exclusively designed for the scenarios, where data access patterns exhibit data grouping and interest locality among the data. In [20], [22], a sampling-based data distribution approach is proposed, where it is mandated to execute a sample task on each virtual node to estimate computational capacity the proportional data placement. In [23], [24], overlapped execution of MR jobs is explored to improve the overall performance. In [24], an algorithm is designed to address the overlapping execution of different phases, such as map, shuffle, and reduce. However, [24] algorithm relies on a sampler that collects data distribution information from partitions before executing an MR job. In [23], a joint scheduling optimization of overlapping map and shuffle phases is carried out by introducing the strong pairs to minimize the average job makespan. However, it is assumed that an entire set of jobs can be decomposed into strong pairs whereby map and shuffle workloads of one job equal to shuffle and map workloads of another job. Such assumptions may not hold for all jobs in a real production cluster.

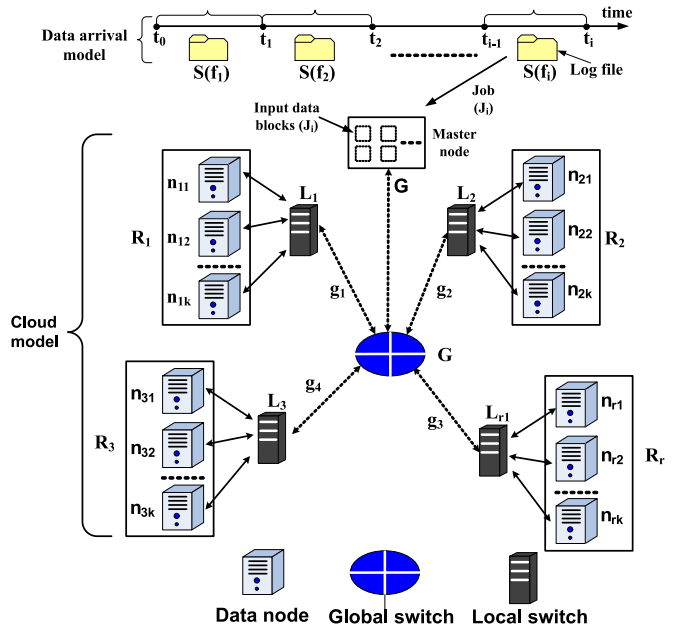


Fig. 2. System model of RENDA.

The previous approaches' fundamental shortcoming is to build the notion of heterogeneity based on the sole factor of computational resources. We argue that several other factors are responsible for the heterogeneous behavior of the cloud nodes, such as the concurrency of tasks, resource availability, and network delay. In the presence of the factors as mentioned above, the estimation of the nodes' computational capacity may not be fully realized by merely running sample tasks. A competent data placement scheme comprehensively evaluates the impact of the aforementioned factors instead of relying on the nodes' computational capacity for data distribution.

3 Problem Statement

In a typical MR job, data blocks are distributed followed by the data blocks' processing in a sequence. Hence, for any MR job J_i, the job completion time $T_{JC}^{J_i}$ can be defined as shown in Eq. (1).

$$T_{JC}^{J_i} = T_D^{J_i} + T_P^{J_i}. \tag{1}$$

Here, $T_D^{J_i}$ and $T_P^{J_i}$ represent data blocks distribution time and data blocks processing time of job J_i, respectively. In MR, data blocks are processed in two phases, such as a map and a reduce. Hence, $T_P^{J_i}$ can be defined as shown in Eq. (2).

$$T_P^{J_i} = T_M^{J_i} + T_R^{J_i}. \tag{2}$$

Here, $T_M^{J_i}$ and $T_R^{J_i}$ represent the completion time of the map phase and reduce phase of job J_i, respectively. The periodic job arrival scenario is shown in Fig. 2. The system model is comprised of data arrival model and cloud model. The data arrival model is defined as a periodic time series model represented as $T = \{t_0, t_1, \ldots, t_i\}$, where input data generated in time interval $\Delta = \{\Delta_{0,1}, \Delta_{1,2}, \ldots, \Delta_{i-1,i}\}$ are stored in log files $F = \{f_1, f_2, \ldots, f_i\}$, respectively. The input data are continuously generated by applications such as streaming tweets, Google search queries, etc., which are aggregated periodically and logged into the files to form the workload batches. Here, $\Delta_{i-1,i}$ represents the time interval between t_{i-1} and t_i, and f_i represents the log file consisting of data generated in $\Delta_{i-1,i}$. The size of the f_i is represented as $S(f_i)$. Since, data generation rate may vary from one time interval to another, $S(f_i) \neq S(f_{i+1})$. This set F of log files acts as periodical input workload to be processed by the slave nodes.

TABLE 1
Notations and Corresponding Meaning

Notation	Meaning
J_i	i^{th} MR job
$T_{JC}^{J_i}$	Job completion time of J_i
$T_D^{J_i}$	Data blocks distribution time of J_i
$T_P^{J_i}$	Data blocks processing time of J_i
$T_M^{J_i}$	Map phase completion time of J_i
$T_R^{J_i}$	Reduce phase completion time of J_i
N	Number of data nodes in a cloud data center
r	Number of racks in a cloud data center
R_i	i^{th} rack of cloud data center
n_{rk}	k^{th} node of r^{th} rack of a cloud data center
d_{rk}	Data blocks allocated to n_{rk}
$\mathbb{m}_{rk}$	Time taken by n_{rk} to process $\lvert d_{rk}\rvert$ during map phase
L_i	i^{th} local switch of cloud data center
r_p	p^{th} reduce node (reducer)
$\mathbb{r}_p$	Time taken by p^{th} reducer in reduce phase
$\mathcal{S}_{rk}$	Sample task execution time on node n_{rk}
$\mathcal{R}_{rk}$	Relative performance of n_{rk} with respect to $\mathcal{S}_{rk}$
pt_{rk}	Mean data block processing time of n_{rk}

The cloud model is presented in Fig. 2. Let N be the number of nodes, and r be the number of racks in a cloud data center. Let r racks be represented as $R = \{R_1, R_2, \ldots, R_r\}$. It is assumed that each rack $R_i \in R$ is comprised of identical number of nodes. Let $k = \lfloor \frac{N}{r} \rfloor$ be a number of nodes in a rack R_i represented as $\{n_{i1}, n_{i2}, \ldots, n_{ik}\}$. Let the communication among the nodes within a rack $R_i \in R$ takes place via local switch L_i; whereas communication among the nodes across the racks $R_i \in R$ and $R_j \in R$ takes place via combination of local switches (L_i, L_j) and global switch G. The cloud model is assumed to have heterogeneous nodes with different computational capacities, and network with different latency and bandwidth. The notations and corresponding meaning are described in Table 1.

Let, job J_i be arrived at time t_i in a cloud data center of N nodes with an input file f_i of size $S(f_i)$. The master node divides the file f_i into set D_{J_i} of data blocks with each data block of size b, where $|D_{J_i}| = \left\lceil \frac{S(f_i)}{b} \right\rceil$. Later, the master node distributes $|D_{J_i}|$ data blocks among N nodes. Let nodes $n_{11}, n_{12}, \ldots, n_{rk}$ receive $d_{11}, d_{12}, \ldots, d_{rk}$ data blocks, respectively. Here, $|D_{J_i}| = \sum_{i=1}^{r} \sum_{j=1}^{k} |d_{ij}|$, where $|d_{ij}|$ is the number of data blocks in j^{th} node of i^{th} rack. Let us assume that master node takes $\mathbb{d}_{11}, \mathbb{d}_{12}, \ldots, \mathbb{d}_{rk}$ amount of time to distribute $|d_{11}|, |d_{12}|, \ldots, |d_{rk}|$ data blocks to nodes $n_{11}, n_{12}, \ldots, n_{rk}$, respectively. The data blocks are distributed in parallel, and therefore the longest distribution time among $\mathbb{d}_{11}, \mathbb{d}_{12}, \ldots, \mathbb{d}_{rk}$ is considered as the total data blocks distribution time $T_D^{J_i}$ for job J_i as shown in Eq. (3).

$$T_D^{J_i} = max(\mathbb{d}_{11}, \mathbb{d}_{12}, \ldots, \mathbb{d}_{rk}). \tag{3}$$

In data blocks processing stage, job J_i executes in map phase followed by reduce phase. Let $\mathbb{m}_{ij}$ be the time taken by n_{ij} to process $|d_{ij}|$ data blocks in the map phase. To be specific, $n_{11}, n_{12}, \ldots, n_{rk}$ take $\mathbb{m}_{11}, \mathbb{m}_{12}, \ldots, \mathbb{m}_{rk}$ time to process $|d_{11}|, |d_{12}|, \ldots, |d_{rk}|$ data blocks, respectively. Here, the nodes participating in map phase are referred as mappers. For any job J_i, the map phase completion time represented as $T_M^{J_i}$ is the time taken by the longest running mapper expressed as follow.

$$T_M^{J_i} = max(\mathbb{m}_{11}, \mathbb{m}_{12}, \ldots, \mathbb{m}_{rk}). \tag{4}$$

In a reduce phase, the number of nodes also called as reducers may vary between $[1, N]$, where N is the maximum number of nodes in a cloud. Let p be number of nodes that participate in the reduce phase, where $1 \leq p \leq N$. Let reducers $\chi_1, \chi_2, \ldots, \chi_p$ take $\mathbb{r}_1, \mathbb{r}_2, \ldots, \mathbb{r}_p$ time in the reduce phase, respectively. Here, $\chi_i \in \{n_{11}, n_{12}, \ldots, n_{rk}\}$. The reducers process the data concurrently and therefore $T_R^{J_i}$ represents the time taken by the longest running reducer expressed as follow.

$$T_R^{J_i} = max(\mathbb{r}_1, \mathbb{r}_2, \ldots, \mathbb{r}_p), \ where \ 1 \leq p \leq N. \tag{5}$$

Using the Eqs. (2), (3), (4), (5), and (1) can be rewritten as shown in Eq. (6). The Eq. (6) infers that for any MR job J_i, the $T_{JC}^{J_i}$ is the summation of longest time taken by the respective nodes in data blocks distribution stage, map phase, and reduce phase.

$$\begin{aligned} T_{JC}^{J_i} = {} & max(\mathbb{d}_{11}, \mathbb{d}_{12}, .., \mathbb{d}_{rk}) + max(\mathbb{m}_{11}, \mathbb{m}_{12}, .., \mathbb{m}_{rk}) \\ & + max(\mathbb{r}_1, \mathbb{r}_2, .., \mathbb{r}_p). \end{aligned} \tag{6}$$

From Eq. (6), it is inferred that a single straggling node can potentially hold up the job progress and delay its completion.

4 Pitfalls in Existing Schemes

In this section, the major pitfalls of data placement approaches are described in detail.

4.1 Longer Data Blocks Distribution Time

Several applications generate a huge amount of data such as IP trace streams [25], search queries [26], streaming tweets [27] etc. Due to the huge input data size and a constrained bandwidth in a cloud data center, the data distribution stage itself takes a substantial amount of time followed by the data processing stage which continues for the multiple rounds of MR phases depending on the "cloud data center size" and the "concurrency factor". The "cloud data center size" is defined as the number of nodes in a data center and the "concurrency factor" is defined as the number of concurrently running tasks on any node.

Let node n_{ij} be allocated the $|d_{ij}|$ number of input data blocks. If the concurrency factor of n_{ij} is c_{ij}, then n_{ij} takes $\left\lceil \frac{d_{ij}}{c_{ij}} \right\rceil$ rounds to process $|d_{ij}|$ number of data blocks. Here, $c_{ij} > 0$. Fig. 3 illustrates the example for the execution of node n_{ij} with $|d_{ij}| = 08$ and $c_{ij} = 2$.

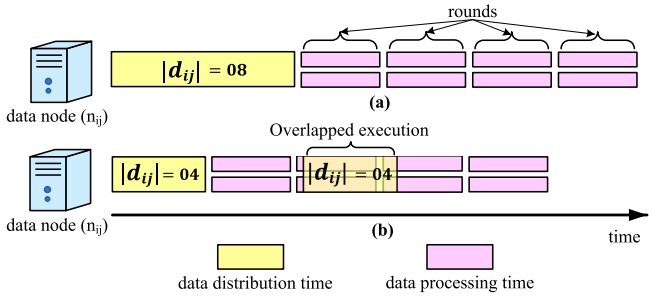


Fig. 3. Multiple rounds of executions of data blocks, (a). in traditional manner, (b). in overlapped manner.

Fig. 3a shows the single installment traditional Hadoop execution [9]. First, the data blocks $|d_{ij}| = 08$ are distributed in the cloud and then n_{ij} processes two data blocks concurrently and takes four rounds to complete depending on the concurrency factor $c_{ij} = 02$. For simplicity, it is assumed that the nodes take equal time to process the data blocks. The improved multi-installment overlapped execution is shown in Fig. 3b. Let initially $|d_{ij}| = 04$ data blocks be distributed and the remaining $|d_{ij}| = 04$ data blocks be distributed in another installment concurrent with the processing of first four data blocks. The main advantage of overlapped execution is a significant reduction in data blocks distribution time. Besides, the overlapped execution facilitates the early start of the data blocks processing stage as well. In overlapped execution, the data blocks distribution and data blocks processing progress simultaneously, which reduces the longer data blocks distribution time, as shown in Fig. 3.

4.2 Uniform Distribution

In a uniform distribution [9], [21], the nodes are assigned with equal number of data blocks irrespective of their processing capacities. Let job J_i be arrives in a cloud data center of N nodes with input data blocks of $|D_{J_i}|$. Each node $n_{ij} \in N$ receives approximately $d = \left\lfloor \frac{|D_{J_i}|}{N} \right\rfloor$ data blocks. Let $pt_{11}, pt_{12}, \ldots, pt_{rk}$ be represents mean data block processing time of nodes $n_{11}, n_{12}, \ldots, n_{rk}$, respectively. For d data blocks, it takes $\mathbb{m}_{11}, \mathbb{m}_{12}, \ldots, \mathbb{m}_{rk}$ amount of time for nodes $n_{11}, n_{12}, \ldots, n_{rk}$ to complete the processing, respectively. Here, $\mathbb{m}_{ij} = d \times pt_{ij}$, and $\mathbb{m}_{ij} \neq \mathbb{m}_{uv}$, where $i, u \in [1, r]$, and $j, v \in [1, k]$. It is worth noting that nodes $n_{11}, n_{12}, \ldots, n_{rk}$ take different amount of time $\mathbb{m}_{11}, \mathbb{m}_{12}, \ldots, \mathbb{m}_{rk}$ to process d blocks, respectively.

From the workings, it is clear that the uniform distribution strategy is easy to realize in practice and suits well for data blocks that need more-or-less homogeneous processing that would consume almost identical amounts of processing time. However, when data blocks take different amounts of time, certain nodes tend to become idle earlier. Under such circumstances, one can always find nodes n_{ij} and n_{uv} with a minimum processing time $\mathbb{m}_{ij}$ and a maximum processing time $\mathbb{m}_{uv}$, respectively. The node n_{ij} likely completes the data blocks processing earlier and become idle. To fully utilize the cloud resources, the node n_{ij} starts fetching the data blocks from node n_{uv} resulting in non-local executions. Such non-local executions increase in cloud data centers with a highly heterogeneous computing resource. Hence, the uniform data blocks distribution is not suitable in a heterogeneous cloud environment.

4.3 Sampling-Based Distribution

In a sampling-based distribution, the Naïve approach [20] is used to allocate data blocks in the proportion of the nodes' computing resources. However, it is challenging to measure the exact computing resources of nodes. Hence, an estimation is made by executing a sample task on nodes.

Let $|D_{J_i}|$ be a number of input data blocks of job J_i, which is to be distributed among N nodes. Let sample task execution time be $\mathcal{S}_{11}, \mathcal{S}_{12}, \ldots, \mathcal{S}_{rk}$ for nodes $n_{11}, n_{12}, \ldots, n_{rk}$, respectively. Using a sample task execution time, a relative performance $\mathcal{R}_{ij}$ of node n_{ij} can be derived as follow.

$$\mathcal{R}_{ij} = \frac{(\mathcal{S}_{ij})^{-1}}{\sum_{x=1}^{r}\sum_{y=1}^{k}(\mathcal{S}_{xy})^{-1}},\ where\ i \in [1,r],\ j \in [1,k]. \tag{7}$$

Let nodes $n_{11}, n_{12}, \ldots, n_{rk}$ be assigned with $|d_{11}|$, $|d_{12}|, \ldots, |d_{rk}|$ data blocks in direct proportion to their observed relative performances $\mathcal{R}_{11}, \mathcal{R}_{12}, \ldots, \mathcal{R}_{rk}$, respectively as shown in Eq. (8). Here, $|d_{ij}| \leq |D_{J_i}|$.

$$|d_{ij}| = \mathcal{R}_{ij} \times |D_{J_i}|. \tag{8}$$

In the following, we explain how the sampling-based distribution performs better over the uniform distribution. Let $|D_{J_i}| = 06$, and each data block d_{ij} = 64 MB, where $d_{ij} \in D_{J_i}$. Let there be three nodes in a cloud data center represented as n_{11}, n_{12}, and n_{13}. Let sample task execution time be $\mathcal{S}_{11} =$ 32s, $\mathcal{S}_{12} =$ 16s, and $\mathcal{S}_{13} =$ 08s. Based on the sample task execution time, the relative performance calculated using Eq. (7) is $\mathcal{R}_{11} = 0.14$, $\mathcal{R}_{12} = 0.29$, and $\mathcal{R}_{13} = 0.57$. In a uniform distribution, each node receives equal number of data blocks. Therefore, $|d_{11}| = |d_{12}| = |d_{13}| = 02$. However, as observed during the sample task execution, nodes n_{11}, n_{12}, and n_{13} are expected to take 64s, 32s, and 16s for the processing of assigned two data blocks, respectively. Such skewed data processing time forces the cloud data center to perform non-local executions. On the contrary, in a sampling-based distribution, nodes n_{11}, n_{12}, and n_{13} are allocated with 01, 02, and 03 data blocks in direct proportion to their relative performance $\mathcal{R}_{11} =$ 0.14, $\mathcal{R}_{12} =$ 0.29, and $\mathcal{R}_{13} = 0.57$, respectively. Based on the sample task execution, nodes n_{11}, n_{12}, and n_{13} take nearly identical data blocks processing time 32s, 32s, and 24s, respectively. Contrary to 64s job execution time in a uniform distribution, the sampling-based distribution concludes in 32s and reduces the job completion time by half.

However, sampling-based distribution does not perform well in a fluctuating cloud performance. Fig. 4 shows an experimental execution of an MR job (Word count) comprised of 69 tasks with their corresponding task completion time on node n_{ij}. Fig. 4 shows that the performance of n_{ij} fluctuates heavily during the job execution. The longest-running task takes 43s, and the least running task completes in 10s. We argue that in such a highly fluctuating cloud performance, the data blocks distribution becomes unreliable if a node's performance is ascertained based on the sample task execution. In the following, scenarios are described to show the vulnerability of the sampling-based approach.

Let sample task on node n_{ij} takes the minimum execution time i.e., $\mathcal{S}_{ij}$ = 10s as shown in Fig. 4. Let $|d_{ij}|$ data blocks be assigned to node n_{ij} based on the sample task execution time. However, $\mathcal{S}_{ij}$ = 10s is the best performance of node n_{ij}. For the rest of the tasks, node n_{ij} observes significantly more time than that of sample one. Therefore, the allocation of $|d_{ij}|$ data blocks based on the sample task execution results into an overestimation of computing capacity of node n_{ij}.

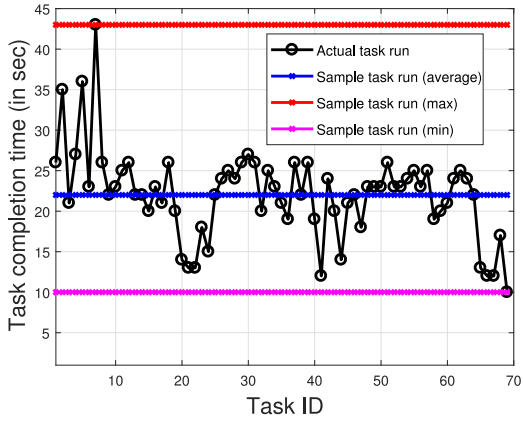


Fig. 4. Problems in sampling based approach.

The opposite is also true as shown in Fig. 4. It is assumed that the sample task takes the maximum execution time i.e., $\mathcal{S}_{ij}$ = 43s. Here, the data blocks distribution based on the sample task execution results in an underestimation of the computational resources of node n_{ij}. In such scenarios, node n_{ij} becomes idle earlier and requests the extra data blocks to improve resource utilization, which results in non-local executions. The third scenario is also shown in Fig. 4. It is assumed that the sample task takes time equals to the average task completion time i.e., $\mathcal{S}_{ij}$ = 22s. However, it is less likely to happen in a highly fluctuating cloud performance. Hence, the sampling-based data blocks distribution is unreliable.

5 RENDA Scheme

In this section, we describe the RENDA scheme with respect to the environment as shown in Fig. 2. The notations used to describe the RENDA scheme are explained in Table 2.

5.1 Overview of the RENDA

The overview of the RENDA is described as follow:

- *Initial distribution:* It is challenging to ascertain the performance of cloud nodes a priori. Therefore, initial distribution is carried out with a few sample data blocks. The nodes are assumed with equal computing resources and an equal number of data blocks are assigned to each node.
- *Estimation of remaining time:* Each node starts the processing of the assigned data blocks and simultaneously estimate the remaining time to finish the processing. Upon attaining the threshold time, the respective node sends a request message to the master node for more data blocks allocation. The threshold time is defined as the amount of remaining time prior to that the subsequent data blocks must be allocated.
- *Subsequent data blocks distribution:* The master node caters the request of the nodes well before they complete the processing of existing data blocks.

TABLE 2
List of Notations Used in RENDA

Notation	Meaning
m	Number of past observations
$P_{n_{ij}}(t)$	Processing rate of node n_{ij} in t^{th} time interval
$w_{n_{ij}}(t)$	Weighted moving average processing rate of n_{ij} in t^{th} time interval
$\mathbb{R}_{n_{ij}}(t)$	Remaining time of n_{ij} in t^{th} time interval
$\mathbb{P}_{n_{ij}}(t)$	Amount of data processed by n_{ij} in t^{th} time interval
δt	Time interval duration
$_{n_{ij}}(t)$	Subsequent data blocks allocated to n_{ij}
$\alpha_{n_{ij}}(t)$	Number of processed data blocks on node n_{ij} in t^{th} time interval
$\beta_{n_{ij}}(t)$	Number of currently under process data blocks on node n_{ij} in t^{th} time interval
$\gamma_{n_{ij}}(t)$	Number of unprocessed data blocks on node n_{ij} in t^{th} time interval
$U_{n_{ij}}(t)$	Total amount of unprocessed data on node n_{ij} in t^{th} time interval
$\mu_y(t)$	Amount of unprocessed data in a data block y in t^{th} time interval
b	Size of an individual data block
$\pi_{n_{ij}}$	Average startup delay to process a data block on node n_{ij}
$\zeta_y(t)$	Amount of processed data in a data block y in t^{th} time interval
r_p	p^{th} reduce node (reducer)
$\mathcal{P}_{ij}$	A unique path from master node to n_{ij}
$T_D^{H,n_{ij}}$	Data blocks distribution time to n_{ij} in traditional hadoop execution
$T_D^{O,n_{ij}}$	Data blocks distribution time to n_{ij} in overlapped execution
B_{cable}	Bandwidth of the cable

The benefit of the RENDA is its pro-activeness in data blocks distribution based on the continuous performance estimation of nodes rather than based on one-time sample task execution. By carefully distributing the set of input data blocks in several installments, a substantial reduction in the number of non-local executions is observed. Nevertheless, RENDA sticks to the default replication strategy of the Hadoop for the fault tolerance.

5.2 Initial Distribution

Let job J_i be arrived at time t_i to a cloud data center of N nodes with an input file size $S(f_i)$. The master node divides the file f_i into set D_{J_i} of data blocks with each data block of size b, where $|D_{J_i}| = \left\lceil \frac{S(f_i)}{b} \right\rceil$. Let $x\%$ of $|D_{J_i}|$ data blocks be initially assigned to the N nodes. Here, $x = \left\lceil \frac{N}{|D_{J_i}|} \times 100 \right\rceil$, which ensures that each node n_{ij} receives at least one data block. In a cloud data center, $|D_{J_i}| >> N$ is a normal scenario. It is challenging to predict the performance of nodes apriori, therefore the set of $x\%$ data blocks are equally assigned to the nodes with each node n_{ij} is assigned with $d = \left\lfloor \frac{\frac{x}{100} \times |D_{J_i}|}{N} \right\rfloor$ data blocks, where $i \in [1, r],\ j \in [1, k]$. It is assumed that due to the heterogeneous network resources, it takes $\mathbb{d}_{11}, \mathbb{d}_{12}, \ldots, \mathbb{d}_{rk}$ time to distribute d data blocks to nodes $n_{11}, n_{12}, \ldots, n_{rk}$, respectively. The data blocks distribution time $\mathbb{d}_{ij}$ of node n_{ij} is called as network delay. In Section 5.5, the network delay calculation is described in detail.

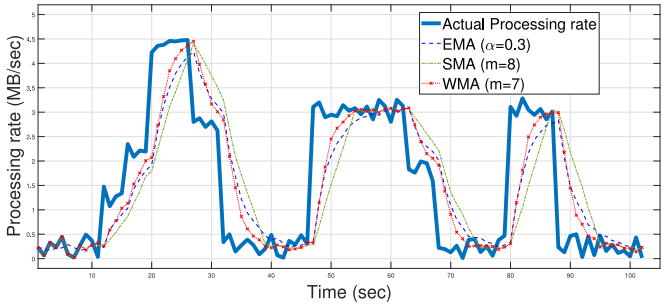


Fig. 5. Performance and applicability analysis of WMA in predicting processing rate.

5.3 Estimation of the Remaining Time

The nodes with heterogeneous processing capacity take a different amount of time to process the allocated data blocks. At any given time interval, RENDA employs the Weighted Moving Average (WMA) of processing rates to better estimate the processing rate and the remaining time. The WMA assigns more weights to the recent observations and less weights to the distant observations, which helps to capture the highly dynamic nature of the cloud environment. Moreover, WMA determines the trend direction more accurately than Simple Moving Average. The WMA and its alternatives are very popular and are used for the load prediction [28], online traffic prediction [29], and dynamic resource allocation [30] in cloud.

The WMA meets the use condition of predicting the processing rate of slave nodes as WMA captures the instantaneous processing rate fluctuations along with historical behavior of the slave nodes. To be specific, WMA gives more importance to the current processing rate, which helps to identify the near future trend of the corresponding slave nodes. The result of Fig. 5 confirms our observation and the appropriateness of the WMA. The WMA estimates any node's current processing rate by assigning the weights in decreasing order from the very recent to the older processing rates. For any node n_{ij}, the weighted moving average processing rate in t^{th} time interval is denoted as $w_{n_{ij}}(t)$. For the past m observations, $w_{n_{ij}}(t)$ is calculated as shown in Eq. (9).

$$w_{n_{ij}}(t) = 2 \times \frac{m.P_{n_{ij}}(t) + (m-1).P_{n_{ij}}(t-1) + \ldots + P_{n_{ij}}(t-m+1)}{m.(m+1)}. \tag{9}$$

Here, $m > 0$ is the number of past observations and it can be decided experimentally. In our experiment, $m = 7$ results in improved performance for Grep and Sort applications. The impact of m on estimation of the remaining time of the nodes is shown in Figs. 10 and 11. In Eq. (9), the $w_{n_{ij}}(t)$ on the left hand side represents the weighted moving average processing rate in t^{th} time interval and the right hand side represents the m numbers of processing rates $\{P_{n_{ij}}(t), P_{n_{ij}}(t-1), \ldots, P_{n_{ij}}(t-m+1)\}$ with discounted weights $\{m, m-1, \ldots, 1\}$, respectively.

$$P_{n_{ij}}(t) = \frac{\mathbb{P}_{n_{ij}}(t)}{\delta t}. \tag{10}$$

The processing rate of any node n_{ij} in t^{th} time interval is defined as $P_{n_{ij}}(t)$ and can be calculated as shown in

TABLE 3
Prediction Error Analysis Results for Grep

	EMA (α=0.3)	SMA (m=8)	WMA (m=7)
MAE	6.5%	8.1%	5.6%
STD	6.6%	8.0%	6.8%
MSE	8.7%	13.0%	7.7%
RMSE	9.3%	11.4%	8.8%

Eq. (10). Here, $\mathbb{P}_{n_{ij}}(t)$ represents the amount of data processed by node n_{ij} in t^{th} time interval, and δt represents the duration of the time interval.

To validate the applicability of WMA in predicting the processing rate, an experiment has been conducted by scheduling different number of VMs on different time instances. As shown in Fig. 5, initially (0-10 sec), the PMs are loaded with four VMs and three VMs were killed gradually during the 10-22 sec that resulted in improved processing rate. The processing rate of any active VM changes significantly by creating or terminating the co-VMs on any PM as shown in Fig. 5. The processing rate prediction ability of WMA is validated under this dynamic scenario against the Exponential Moving Average (EMA) with α = 0.3 and Simple Moving Average (SMA) with window size of m = 8, respectively. Here, α = 0.3 is the constant smoothing factor, which is obtained experimentally. The α = 0.3 and m = 8 for EMA and SMA, respectively is considered due to the improved result. On the contrary, m = 7 is considered for WMA based on the detailed experimental analysis as shown in Figs. 10 and 11. Fig. 5 shows that WMA adapts to the changes faster than the SMA and EMA. However, performance of EMA is in line with the WMA. On the contrary, SMA is bit lazy to adapt to the dynamic changes in the cloud layout and therefore it reduces the predictability of the processing rate.

$$MAE = \frac{\sum_{i=1}^{m} |w_{n_{ij}}(t) - P_{n_{ij}}(t)|}{m} \tag{11}$$

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(P_{n_{ij}}(t) - w_{n_{ij}}(t))^2 \tag{12}$$

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(P_{n_{ij}}(t) - w_{n_{ij}}(t))^2}. \tag{13}$$

To quantify the predictability of EMA, SMA, and WMA, prediction error analysis is performed for Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) using Eqs. (11), (12), and (13), respectively. The corresponding results along with the Standard Deviation (STD) are shown in Table 3. The results are obtained for at least 200 VMs using Grep MR application and then averaged to avoid the bias. It is evident from the results that WMA quickly adapts to the dynamic changes in the processing rate of VMs and has least prediction error over MAE, MSE, and RMSE. On the contrary, SMA suffers due to their lazy adaptiveness to the dynamic changes in processing rates. The WMA performs marginally better over EMA as WMA reacts quickly to the new processing rates.

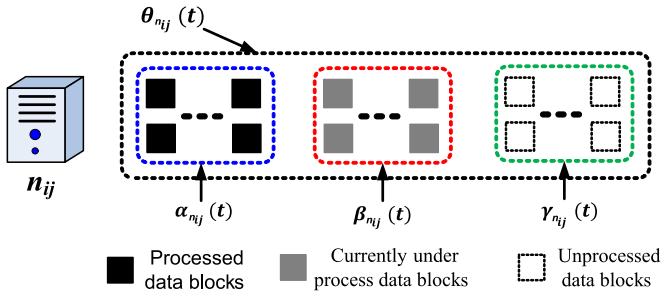


Fig. 6. Representation of data blocks on nodes.

Theorem 1. *In any given time interval, the time complexity to calculate $w_{n_{ij}}(t)$ is a unit time.*

Proof. As shown in Eq. (9), the calculation of $w_{n_{ij}}(t)$ is an iterative process and it takes $m > 0$ iterations. Here, m is the user-defined previous observations.

Let $w_{n_{ij}}(t+1)$ be the weighted moving average in next time interval $t+1$ and can be defined as shown in Eq. (14).

$$w_{n_{ij}}(t+1) = 2 \times \frac{m.P_{n_{ij}}(t+1) + (m-1).P_{n_{ij}}(t) + \ldots + P_{n_{ij}}(t-m)}{m.(m+1)}. \tag{14}$$

The denominators of Eqs. (9) and (14) are constants and can be represented as $\frac{m.(m+1)}{2}$. The difference between $w_{n_{ij}}(t+1)$ and $w_{n_{ij}}(t)$ can be represented as shown in Eq. (15).

$$w_{n_{ij}}(t+1) - w_{n_{ij}}(t) = m.P_{n_{ij}}(t+1) - P_{n_{ij}}(t) - \ldots - P_{n_{ij}}(t-m+1). \tag{15}$$

If we represent $w_{n_{ij}}(t) + \ldots + w_{n_{ij}}(t-m+1)$ as $\mathbb{S}(t)$, then $\mathbb{S}(t+1) = \mathbb{S}(t) + P_{n_{ij}}(t+1) - P_{n_{ij}}(t-m+1)$. Let $\mathbb{N}(t+1)$ and $\mathbb{N}(t)$ represent the numerator of Eqs. (14) and (9), respectively. The $\mathbb{N}(t+1)$ can be obtained using the known information from previous iteration such as $\mathbb{N}(t)$, $m.P_{n_{ij}}(t+1)$, and $\mathbb{S}(t)$. The $\mathbb{N}(t+1)$ can be calculated as shown in Eq. (16).

$$\mathbb{N}(t+1) = \mathbb{N}(t) + P.w_{n_{ij}}(t+1) - \mathbb{S}(t). \tag{16}$$

The iterative calculation of $w_{n_{ij}}(t)$ defined in Eq. (9) can be performed in a single operation using Eqs. (9), (14), and (16). Hence, the time complexity to calculate $w_{n_{ij}}(t)$ is a unit time. □

Without loss of generality, let us consider that in any time interval, each node has a different number of processed, currently under process, and unprocessed data blocks, as shown in Fig. 6. The remaining time of any node is defined as the summation of the time to complete the currently under process and unprocessed data blocks.

Let $\Theta_{n_{ij}}(t)$ be the total number of data blocks on node n_{ij} in t^{th} interval. Out of $\Theta_{n_{ij}}(t)$, let $\alpha_{n_{ij}}(t)$, $\beta_{n_{ij}}(t)$, and $\gamma_{n_{ij}}(t)$ be the processed, currently under process, and unprocessed data blocks, respectively on node n_{ij} in t^{th} interval.

Let $U_{n_{ij}}(t)$ be the total amount of unprocessed data on node n_{ij} in t^{th} interval. The $U_{n_{ij}}(t)$ can be expressed as shown in Eq. (17). Here, $\mu_y(t)$ represents the amount of

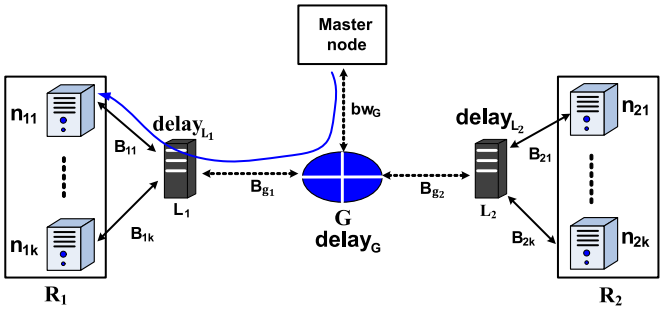


Fig. 7. Network delay of data nodes.

unprocessed data of data block y in t^{th} interval, where $y \in \beta_{n_{ij}}(t)$. The $\mu_y(t)$ can be expressed as $\mu_y(t) = b - \zeta_y(t)$, where b represents individual data block size and $\zeta_y(t)$ represents the amount of processed data of data block $y \in \beta_{n_{ij}}(t)$. The $\zeta_y(t)$ can be obtained using well-defined scheme such as Longest Approximate Time to End (LATE) [31]. For the Map tasks, the processed data is the fraction of input data read. For the Reduce tasks, the execution is divided into three phases such as copy, sort, and reduce, each of which accounts for $\frac{1}{3}$ of the processed data. Based on the fraction of data read, the amount $\zeta_y(t)$ is ascertained and communicated to the Master node in each time interval. $\gamma_{n_{ij}}(t)$ represents the number of unprocessed data blocks on node n_{ij} in t^{th} interval.

$$U_{n_{ij}}(t) = \sum_{y \in \beta_{n_{ij}}(t)} \mu_y(t) + \gamma_{n_{ij}}(t) \times b. \tag{17}$$

Using Eqs. (9) and (17), the remaining time $\mathbb{R}_{n_{ij}}$ can be estimated as defined in Eq. (18). Here, $\pi_{n_{ij}}$ represents the average startup delay to process a data block on node n_{ij}.

$$\mathbb{R}_{n_{ij}}(t) = \frac{U_{n_{ij}}(t)}{w_{n_{ij}}(t)} + \gamma_{n_{ij}}(t) \times \pi_{n_{ij}}. \tag{18}$$

In Eq. (18), the $w_{n_{ij}}(t)$ represents the overall average data processing rate in time interval t, which also includes the concurrent processing rate of the node. The number of concurrent tasks on any node may vary from one-time interval to another. Therefore, without losing the generality, the average data processing capacity $w_{n_{ij}}(t)$ is considered for any node n_{ij}.

5.4 Subsequent Data Blocks Distribution

During the data blocks processing, nodes communicate $\mathbb{R}_{n_{ij}}(t)$ to master node in each time interval. Later, master node compares the $\mathbb{R}_{n_{ij}}(t)$ with network delay $\mathbb{d}_{ij}$ and decides the threshold value using Eq. (19). The threshold represents the time at which subsequent data blocks should be assigned to respective node n_{ij}.

$$threshold = \begin{cases} 0 & if\ (\mathbb{R}_{n_{ij}}(t) - \mathbb{d}_{ij}) > 0 \\ 1 & if\ (\mathbb{R}_{n_{ij}}(t) - \mathbb{d}_{ij}) <= 0 \end{cases}. \tag{19}$$

Here, $threshold = 0$ indicates that the node has enough data to process. On the contrary, $threshold = 1$ indicates that the subsequent data blocks allocation should be initiated. Let $l_{n_{ij}}$ be the subsequent data blocks allocated to node n_{ij}. The $l_{n_{ij}}$ can be calculated using Eq. (20).

$$l_{n_{ij}} = x \times \frac{\mathcal{R}_{n_{ij}}(t)}{\sum_{i=1}^{r}\sum_{j=1}^{k} \mathcal{R}_{n_{ij}}(t)}. \tag{20}$$

Here, $\mathcal{R}_{n_{ij}}(t)$ is the relative performance of node n_{ij} in a cloud data center in t^{th} time interval. The $\mathcal{R}_{n_{ij}}(t)$ can be calculated using Eq. (21).

$$\mathcal{R}_{n_{ij}}(t) = \frac{P_{n_{ij}}(t)}{min\{P_{n_{11}}(t), \ldots, P_{n_{rk}}(t)\}}. \tag{21}$$

Theorem 2. *Given the* $\{\mathbb{R}_{n_{11}}(t), \mathbb{R}_{n_{12}}(t), \ldots, \mathbb{R}_{n_{rk}}(t)\}$ *and set* $\{\mathbb{d}_{11}, \mathbb{d}_{12}, \ldots, \mathbb{d}_{rk}\}$, *it takes linear time for master node to verify the threshold in each time interval.*

Proof. Consider the worst scenario as follow. The nodes $n_{11}, n_{12}, \ldots, n_{rk}$ send the remaining time information $\mathbb{R}_{n_{11}}(t), \mathbb{R}_{n_{12}}(t), \ldots, \mathbb{R}_{n_{rk}}(t)$ in the form of heartbeat message at the same time to master node, respectively.

For simplicity $N = |\{n_{11}, n_{12}, \ldots, n_{rk}\}|$. This implies that cloud data center is comprised of total N nodes, which together send N heartbeat messages in each time interval. As shown in Fig. 2, the nodes have unique path from master node, which implies that each node is associated with single network delay. Hence, $N = |\{\mathbb{d}_{11}, \mathbb{d}_{12}, \ldots, \mathbb{d}_{rk}\}|$.

In each heartbeat message, master node performs one-to-one comparison of N remaining time messages $\mathbb{R}_{n_{11}}(t), \mathbb{R}_{n_{12}}(t), \ldots, \mathbb{R}_{n_{rk}}(t)$ from nodes to corresponding N network delay messages $\mathbb{d}_{11}, \mathbb{d}_{12}, \ldots, \mathbb{d}_{rk}$, respectively. The comparison process concludes in the most N comparisons. Hence, the worst case time complexity to obtain threshold information is $\mathcal{O}(N)$. □

5.5 Determining Network Delay

In RENDA, network delay $\mathbb{d}_{ij}$ is defined as the data blocks distribution time to node n_{ij}. Fig. 7 shows the part of the cloud model shown in Fig. 2. Here, the network configuration is comprised of switches and cables with different latencies and bandwidths, respectively. Let nodes $n_{i1}, n_{i2}, \ldots, n_{ik}$ of rack R_i be connected to local switch L_i using network cables with bandwidths $B_{i1}, B_{i2}, \ldots, B_{ik}$, respectively. The local switch L_i connects to the global switch G using a cable with bandwidth B_{g_i} and the global switch in turn connects to the master node using the cable with bandwidth bw_G.

To calculate the network delay $\mathbb{d}_{ij}$, a unique path $\mathcal{P}_{ij}$ is defined comprised of network cables and switches. Eq. (22) represents the network delay $\mathbb{d}_{ij}$ that incurs to transfer N data blocks from the master node to node n_{ij}.

$$\mathbb{d}_{ij} = \sum_{switch \in \mathcal{P}_{ij}} delay_{switch} + \sum_{cable \in \mathcal{P}_{ij}} \frac{N}{B_{cable}}. \tag{22}$$

Eq. (22) calculates the network delay $\mathbb{d}_{n_{ij}}$ by incorporating the delays incurred by switches and transmission time consumed by cables in path $\mathcal{P}$ to transfer N data blocks. The network delay is comprised of switch delay and data transmission time. The switching delay, i.e., $delay_{switch}$ is

perceived as a fixed minimum time (latency) a switch takes to forward a data block irrespective of its size, and therefore it is a constant term. On the contrary, the data transmission time is a function of the number of data blocks, i.e., N and cable bandwidth, i.e., B_{cable}.

The step by step process of RENDA data placement procedure is presented in Algorithm 1. Initially, an equal number of data blocks are assigned to each node and a unique network delay $\mathbb{d}_{ij}$ is obtained using Eq. (22). In each heartbeat, nodes obtain the information of the number of processed ($\alpha_{n_{ij}}(t)$), currently under processing ($\beta_{n_{ij}}(t)$), and unprocessed ($\gamma_{n_{ij}}(t)$) data blocks and estimate the remaining time ($\mathbb{R}_{n_{ij}}(t)$) using the WMA ($w_{n_{ij}}(t)$). In any time interval, if the remaining time $\mathbb{R}_{n_{ij}}(t)$ becomes less than the network delay $\mathbb{d}_{ij}$, $l_{n_{ij}}$ data blocks are distributed by calculating the relative performance of the node $\mathcal{R}_{n_{ij}}(t)$.

Algorithm 1. RENDA Algorithm

Input: $S(f_i)$, b, N
Output: $l_{n_{ij}}$
Number of data blocks: $|D_{J_i}| = \left\lceil \frac{S(f_i)}{b} \right\rceil$) ;
Determine the initial percentage: $x = \left\lceil \frac{N}{|D_{J_i}|} \times 100 \right\rceil$;
Equally distribute $\left\lfloor \frac{\frac{x}{100} \times |D_{J_i}|}{N} \right\rfloor$ data blocks ;
foreach *node:* $n_{ij} \in N$ **do**
  Prepare network path $\mathcal{P}_{ij}$;
  Calculate network delay $\mathbb{d}_{ij}$ using Eq. (22);
end
foreach *heartbeat message at time t* **do**
  foreach *node:* $n_{ij} \in N$ **do**
    Obtain number of processed data blocks $\alpha_{n_{ij}}(t)$;
    Obtain number of currently under process data blocks $\beta_{n_{ij}}(t)$;
    Obtain number of unprocessed data blocks $\gamma_{n_{ij}}(t)$;
    Calculate processing rate $P_{n_{ij}}(t)$ using Eq. (10) ;
    Calculate weighted moving average processing rate $w_{n_{ij}}(t)$ using Eq. (9) ;
    Calculate amount of unprocessed data $U_{n_{ij}}(t)$ using Eq. (17) ;
  Calculate remaining time $\mathbb{R}_{n_{ij}}(t)$ using Eq. (18) ;
  if $(\mathbb{R}_{n_{ij}}(t) - \mathbb{d}_{n_{ij}}) \leq 0$) **then**
    if *More data blocks?* **then**
      Calculate relative performance $\mathcal{R}_{n_{ij}}(t) = \frac{P_{n_{ij}}(t)}{min\{P_{n_{11}}(t),\ldots,P_{n_{rk}}(t)\}}$;
      Dispatch $l_{n_{ij}} = x \times \frac{\mathcal{R}_{n_{ij}}(t)}{\sum_{i=1}^{r}\sum_{j=1}^{k}\mathcal{R}_{n_{ij}}(t)}$ data blocks to node n_{ij} ;
    end
  end
end
end

Theorem 3. *The data blocks distribution time in overlapped execution is at most the distribution time in a traditional execution.*

Proof. Let the cloud data center comprises of N number of nodes with $|D_{J_i}|$ input data blocks. First, the theorem's correctness is described for one node, which is later generalized for all cloud nodes.

In traditional Hadoop execution, data blocks distribution is a one time process and is carried out in a load balanced manner. The $|D_{J_i}|$ data blocks are distributed in a manner that each node n_{ij} receives $|d| = \frac{|D_{J_i}|}{N}$ data blocks. For any node n_{ij}, the data blocks distribution time $T_D^{T,n_{ij}}$ can be expressed as shown in Eq. (23) using Eq. (22). For simplicity, the first part of Eq. (22) is ignored as irrespective of the data blocks, the delay of network switches remain constant.

$$T_D^{T,n_{ij}} = \sum_{cable \in \mathcal{P}_{ij}} \frac{|d|}{B_{cable}}. \tag{23}$$

As the summation is commutative and associative, we can rewrite Eq. (23) as shown in Eq. (24), where $1 \leq x \leq 100$.

$$T_D^{T,n_{ij}} = \sum_{cable \in \mathcal{P}_{ij}} \frac{x\% \times |d|}{B_{cable}} + \sum_{cable \in \mathcal{P}_{ij}} \frac{(100-x)\% \times |d|}{B_{cable}}. \tag{24}$$

In overlapped execution, except initial $x\%$ of $|D_{J_i}|$ uniform data blocks distribution, rest $(100-x)\%$ of $|D_{J_i}|$ data blocks are distributed in concurrent of data blocks processing. Hence, for any node n_{ij}, the data blocks distribution time in overlapped execution can be expressed as shown Eq. (25). Again, for simplicity, the delay of network switches is ignored.

$$T_D^{O,n_{ij}} = \sum_{cable \in \mathcal{P}_{ij}} \frac{x\% \times |d|}{B_{cable}} + \sum_{cable \in \mathcal{P}_{ij}} \frac{(100-x)\% \times |d|}{B_{cable}}. \tag{25}$$

In overlapped execution, the distribution of $(100-x)\%$ of $|d_{ij}|$ data blocks is carried out concurrently with data blocks processing. Hence, the effective data blocks distribution time is considered the difference between network delay and the corresponding node processing time, as expressed in Eq. (26).

$$T_D^{O,n_{ij}} = \sum_{cable \in \mathcal{P}_{ij}} \frac{x\% \times |d|}{B_{cable}} + \sum_{cable \in \mathcal{P}_{ij}} \frac{(100-x)\% \times |d|}{B_{cable}} - \frac{(100-x)\% \times |d|}{P_{n_{ij}}}. \tag{26}$$

From Eqs. (24) and (26), we can write $T_D^{O,n_{ij}} \leq T_D^{T,n_{ij}}$. □

6 Experimental Results

In this section, the cloud environment setup is described, and the evaluation results of the RENDA are presented.

6.1 Environment and Settings

The cloud environment settings for physical nodes (PNs) and virtual nodes (VNs) are shown in Table 4, respectively. To be specific, four high performance PNs are employed with each PN comprised of four cores Intel(R) Core(TM) i7-3770 process, 8.0 GB of main memory, and 1.0 TB of storage. Consequently, sixteen VNs are created, with each PN

TABLE 4
The Cloud Data Center Environment Settings for Physical Nodes

Physical node	Specifications	# of Virtual nodes	Type of virtual node	Network switch	Virtual node	CPU	Memory	Storage
PN_1	CPU: Intel(R)Core(TM)i7-3770, Memory: 8.0GB, Storage: 1.0TB	4	1 master, 3 slaves	$L_1 = 100Mbps$	VN_1	2	1024 MB	45 GB
					VN_2	1	2048 MB	50 GB
					VN_3	1	1536 MB	35 GB
					VN_4	2	2048 MB	50 GB
PN_2	CPU: Intel(R)Core(TM)i7-3770, Memory: 8.0GB, Storage: 1.0TB	4	4 slaves	$L_2 = 1Gbps$	VN_5	2	1536 MB	35 GB
					VN_6	1	1024 MB	45 GB
					VN_7	1	2048 MB	40 GB
					VN_8	2	1024 MB	50 GB
PN_3	CPU: Intel(R)Core(TM)i7-3770, Memory: 8.0GB, Storage: 1.0TB	4	4 slaves	$L_3 = 100Mbps$	VN_9	2	1024 MB	35 GB
					VN_{10}	1	1536 MB	45 GB
					VN_{11}	1	1536 MB	40 GB
					VN_{12}	2	2048 MB	50 GB
PN_4	CPU: Intel(R)Core(TM)i7-3770, Memory: 8.0GB, Storage: 1.0TB	4	4 slaves	$L_4 = 100Mbps$	VN_{13}	2	1024 MB	35 GB
					VN_{14}	1	2048 MB	45 GB
					VN_{15}	1	1536 MB	40 GB
					VN_{16}	2	1024 MB	50 GB

hosting four VNs. The VNs are created using Oracle VM VirtualBox 5.1.22. The system configuration of VNs is carefully chosen and made diversified to ensure a heterogeneous cloud environment. Out of the sixteen VNs, the VN_4 acts as a master and slave node, and the rest VNs act as a slave. Additionally, the cloud environment is made heterogeneous in terms of network data transfer capacity. Three local switches such as L_1, L_3, and L_4 are chosen with a theoretical data transfer speed of 100 Mbps; two network switches such as L_2 and G are chosen with a theoretical data transfer speed of 1.0 Gbps. The VNs are equipped with the operating system Ubuntu 14.04.5 LTS (Trusty Tahr) and Hadoop repository Hadoop 1.0.1 (Stable release).

6.2 Results and Discussion

The performance evaluation is carried out on MR applications such as WordCount, Grep, and Sort. Each application is executed for at least ten rounds across different input data sizes, such as 32 GB, 64 GB, and 96 GB, and later averaged to obtain consistent results.

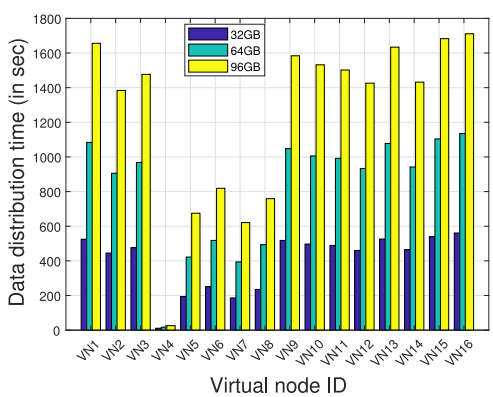
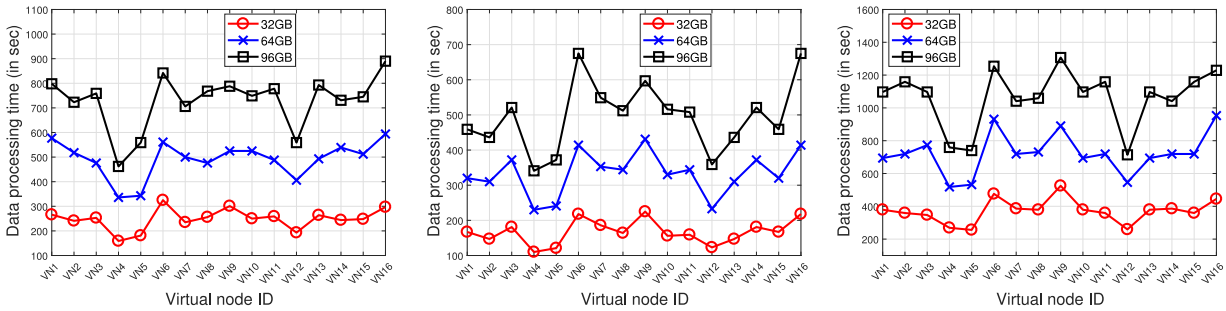


Fig. 8. Data distribution time of virtual nodes in a cloud.

The performance of the virtual cloud data center is shown in Fig. 8 to demonstrate the time each VN takes to distribute the input data. It is to note that the input data is equally distributed among the VNs. For example, the 32 GB input file is partitioned into data blocks each with 64 MB and assigned in equal number to all of the sixteen VNs. Each VN receives 32 data blocks with an accumulated input data size of 2.0 GB. Despite allocating the equal amount of data, the heterogeneity in network switches, the amount of memory, and hard disk rotational speed, the data distribution time varies across the VNs. As evident from Fig. 8, except VN_4, the data distribution time is noticeably less for VN_5, VN_6, VN_7, and VN_8 hosted on PN_2 compared to those hosted on PN_1, PN_3, and PN_4. Since VN_4 acts as master and slave node, a little data transfer time is observed. From Fig. 8, it can be observed that despite the same number of CPUs and amount main memory allocated to VN_7 and VN_{14}, the data transfer time differs significantly. This infers that network delay is also a prominent factor behind cloud heterogeneity in addition to factors such as computing capacity.

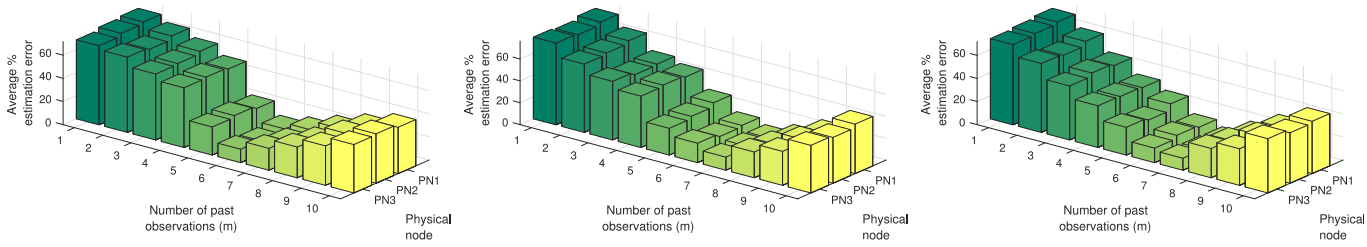
The heterogeneous performance of VNs is presented in Fig. 9. For each VN, the data processing time is calculated separately, and the mean of at least ten rounds of execution is considered. It is observed that the VN_4, VN_5, and VN_{12} with better computing capacity perform consistently better across the applications with different input data sizes. The performance evaluation outcomes for WordCount, Grep, and Sort is shown in Figs. 9a, 9b, and 9c, respectively. The evaluation results show that irrespective of the computing capacity, the VNs take significant processing time in Sort's case, followed by Grep, and WordCount. The Sort involves two computer-intensive operations, such as search and compare, followed by Grep, which involves only search operation. On the contrary, the WordCount involves less compute-intensive operation count.

The RENDA employs the WMA for the estimation of the remaining time of the node. However, the value of m, the number of past observations to include in WMA, is critical



(a) WordCount (b) Grep (c) Sort

Fig. 9. Data processing time across the virtual nodes in a cloud.



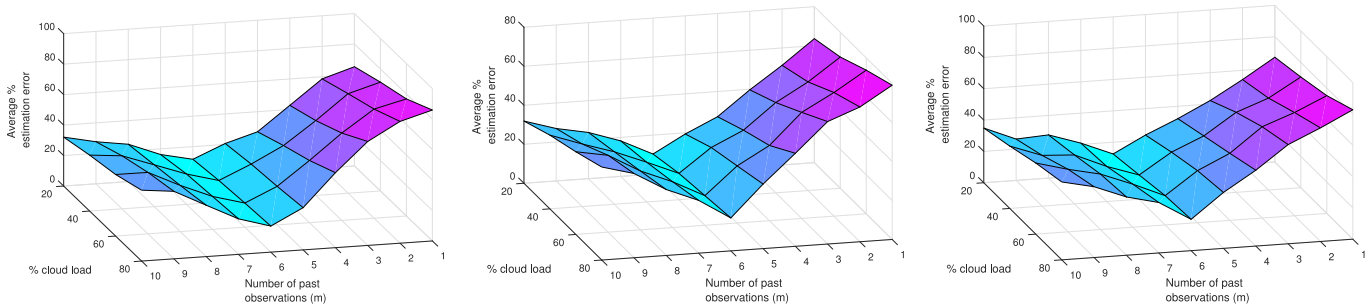
(a) WordCount (b) Grep (c) Sort

Fig. 10. Impact of past "m" observations of WMA on estimation of the remaining time of the node.

to RENDA's performance. Therefore, an error analysis is carried out across three MR applications, as shown in Figs. 10 and 11 to experimentally obtain the sub-optimal window size for m. In the first experiment of error analysis shown in Fig. 10, the average % estimation error is considered as a metric. At least 10 MR tasks are executed on each PN and remaining time estimation is made for the $m = 0$ to $m = 10$. The difference between actual and estimated task completion time is averaged, and results are plotted for WordCount, Grep, and Sort, as shown in Figs. 10a, 10b, and 10c, respectively. The results show the consistently lower estimation error for window size $m = [5 - 8]$ with $m = 6$ provides the least estimation error across the three MR applications. The results provide insights that a sufficient number of past observations are required to better estimate the remaining time. However, the inclusion of more than eight and less than five previous observations likely increases the estimation error. The potential reason might be the older values become irrelevant and contribute less in generalizing the node performance. On the contrary, an insufficient number of previous observations may not help WMA to realize the actual node performance.

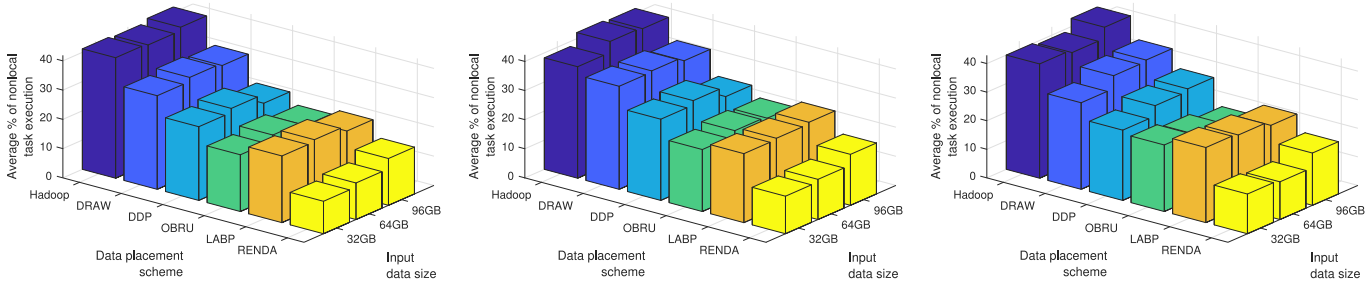
Despite the window $m = [5 - 8]$ provides evidence of lower estimation error, another experiment was carried out for detailed investigation under different % of cloud loads. The average % estimation error is obtained for $m = 1$ to $m = 10$ under cloud loads 20, 40, 60, and 80 percent and the corresponding results for WordCount, Grep, and Sort are shown in Figs. 11a, 11b, and 11c, respectively. The results shown in Fig. 11 confirm the window size $m = [5 - 8]$ under varying cloud load scenario as well.

The performance of RENDA is evaluated against the state-of-the-art schemes such as Dynamic Data Placement (DDP) [20], Data-gRouping-AWare Data Placement (DRAW) [21], Locality Aware Block Placement (LABP) [4], Overlapping-Based Resource Utilization (OBRU) [24], and default Hadoop Placement. The schemes are evaluated for the following performance metrics. 1). The percentage of non-local task executions, 2). The percentage of data transfer overhead, and 3). The average job completion time.



(a) WordCount (b) Grep (c) Sort

Fig. 11. Impact of past "m" observations of WMA on estimation of the remaining time of the node with different percentage of cloud load.



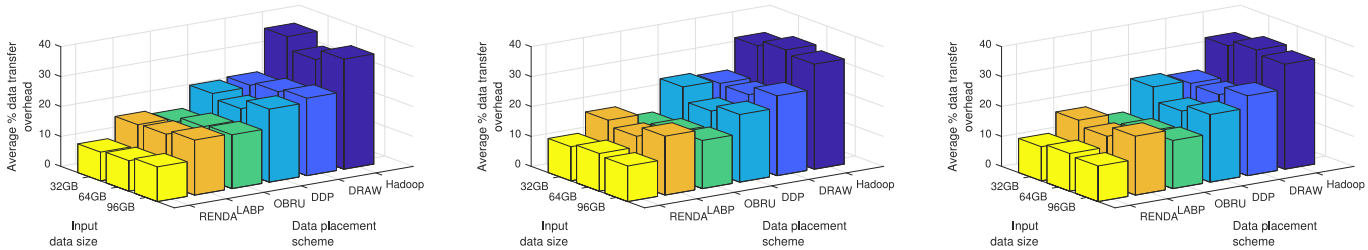
(a) WordCount (b) Grep (c) Sort

Fig. 12. Comparison of data placement schemes with respect to percentage of non-local task executions.

Fig. 12 shows the results for the percentage of non-local task executions across different input data sizes and Fig. 13 shows the results for the percentage of data transfer overhead across different input data sizes. In the default Hadoop scheme, the data are distributed irrespective of the computing capacity of nodes and network delay. This is the main reason for the default Hadoop placement scheme's poor performance, leading to the highest percentage of non-local task executions and data transfer overhead compared to the rival schemes, as evident from Figs. 12 and 13, respectively. On the contrary, DRAW [21] and DDP [20] performs marginally better compared to default Hadoop placement. The potential justification for the improved performance of DDP [20] over the DRAW [21] is its partially informed data distribution scheme. The DDP [20] obtains the computing capacity information of VNs from the sample task execution. On the contrary, DRAW [21] is highly application-specific, and in the absence of significant interest locality and data grouping, it performs similarly to that of the default Hadoop scheme. Contrary to DDP and DRAW, recent data placement schemes LABP [4] and OBRU [24] perform better. The LABP improves the local task executions and reduces the data transfer overhead at the cost of an increase in the number of replicas of data blocks. However, data blocks replication increases the data blocks distribution time and thereby increases the job completion time. On the contrary, OBRU benefits from its overlapped execution strategy of multiple MR phases such as Map, Shuffle, and Reduce. However, OBRU heavily relies on the sampler to collect the data distribution information from partitions before running the MR jobs. As evident from Figs. 12 and 13, the RENDA consistently outperforms the rival schemes irrespective of the underlying MR applications and input data size with a reduction in the percentage of non-local executions and data transfer overhead. The possible explanation of RENDA's improved performance is it's careful on-the-fly dispatching of data to VNs just in time before the currently executing data exhaust.

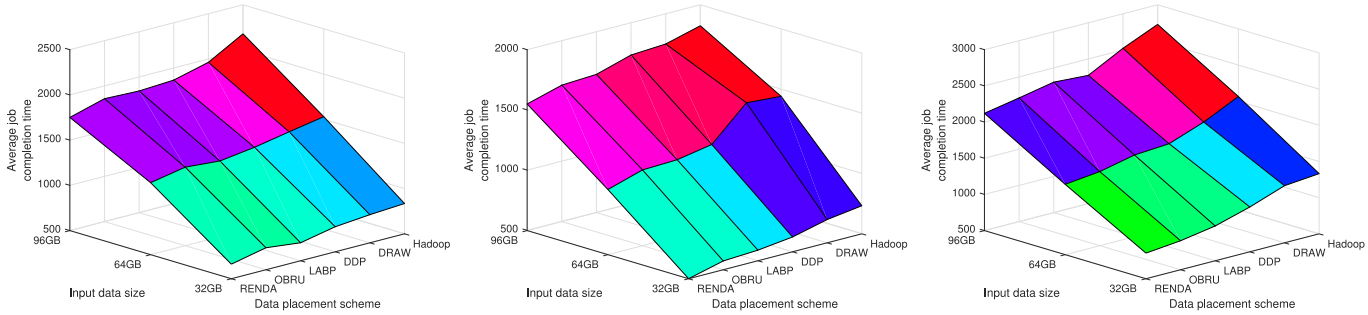
Fig. 14 shows the performance comparison in terms of overall average job completion time taken by various schemes across different MR applications and input data size. Fig. 14 shows that the default Hadoop scheme performs poorly and takes the maximum job completion time followed by DRAW, DDP, LABP, OBRU, and the RENDA. The DRAW, DDP, and LABP schemes have the sequential execution compulsion for the job execution, whereby the data processing stage follows the data distribution stage. Hence, unless the data distribution is completed, the data processing can not start. In several instances, few straggling nodes or faulted networking hold up the data distribution stage for unexpectedly longer duration resulting in the delay of overall job completion time. However, LABP benefits from its locality-awareness data block distribution strategy, which reduces the percentage of non-local task executions and decreases job completion time. On the contrary, OBRU performs better than DRAW, DDP, and LABP as it efficiently handles the imbalanced workloads during the reduce phase to efficiently utilize the cloud resources, which marginally improves the performance of the MR jobs. RENDA overcomes the compulsion as mentioned above and overlaps the data distribution and data processing stages by carefully allocating data in several installments based on the nodes' real-time computing capacity estimation. As evident from Figs. 14a, 14b, and 14c, RENDA shows the improved performance over the rival schemes with reduced overall job completion time regardless of the underlying MR applications and input data size.

Table 5 shows the percentage reduction obtained in non-local execution of tasks by RENDA with respect to the default Hadoop, DRAW, DDP, LABU, and OBRU. The



(a) WordCount (b) Grep (c) Sort

Fig. 13. Comparison of data placement schemes with respect to percentage of data transfer overhead.



(a) WordCount (b) Grep (c) Sort

Fig. 14. Comparison of data placement schemes with respect to average job completion time.

performance comparison results are obtained for three MR applications, such as WordCount, Grep, and Sort. The RENDA achieves as much as a 28 percent reduction in non-local execution compared to the default Hadoop scheme. The RENDA also performs superior compared to the DRAW, DDP, LABU, and OBRU with as much as 20.5, 13.0, 12.2, and 10.6 percent reduction in non-local task executions. The results of Table 5 shows that the RENDA is robust and performs consistently well across the diversified MR applications and workload sizes. Table 6 shows the percentage reduction obtained in average job completion time by the RENDA with respect to the default Hadoop, DRAW, DDP, LABU, and OBRU. On the line of the results obtained in Table 5, RENDA shows as much as 29.0, 23.3, 13.9, 12.6, and 11.8 percent reduction in average job completion time compared to the default Hadoop scheme, DRAW, DDP, LABU, and OBRU, respectively.

The load-balanced data distribution is one of the crucial aspects to improve the job completion time. The RENDA carries out multi-installment on-th-fly data blocks allocation in proportion to the computing capacity of nodes and network characteristics. The experiment was carried out to validate the performance of RENDA for subsequent data blocks distribution. Except the master node VN_4, data blocks' processing time of all slave VNs is recorded for at leaast ten tasks of a given MR application. Additionally, the performance is assessed across different input sizes, such as 32 GB, 64 GB, and 96 GB. The corresponding results for the WordCount, Grep, and Sort are shown in Figs. 15a, 15b, and 15c, respectively. Despite heterogeneous computing capacities of nodes shown in Fig. 9, the data placement by RENDA has resulted in uniform average data processing time, as shown in Fig. 15. The uniform data processing time of nodes confirms the balanced distribution of workloads to the nodes. The slower nodes received less data and faster nodes received more data. The maximum standard deviation of data processing time of nodes is 5.4 sec, 8.6 sec, and 11.2 sec for input data size 32 GB, 64 GB, and 96 GB, respectively across the three MR applications.

6.3 Limitations and Future Directions

The RENDA ensures the efficient data placement of periodic workloads in heterogeneous clouds and improves cloud resource utilization by reducing the overall job completion time. The efficiency of RENDA is validated on a in-house cloud of a 16 VMs, which is one of the limitations. In the future, we starve to validate the scalability of RENDA on a cloud data center comprised of hundreds of VMs. Under the abstraction of the MR paradigm, RENDA relies on the Weighted Moving Average (WMA) to estimate the remaining time. The WMA performs reasonably well, provided a sufficient number of past m observations are considered, which is challenging to ascertain theoretically in a production cluster. However, sub-optimal m can be obtained experimentally, which provides an improved estimation of the remaining time. The online estimation of remaining time and prefetch of data requires additional computation power compared to the straight forward default Hadoop data placement. However, the time complexity to estimate remaining time is a unit time as described in Theorem 1.

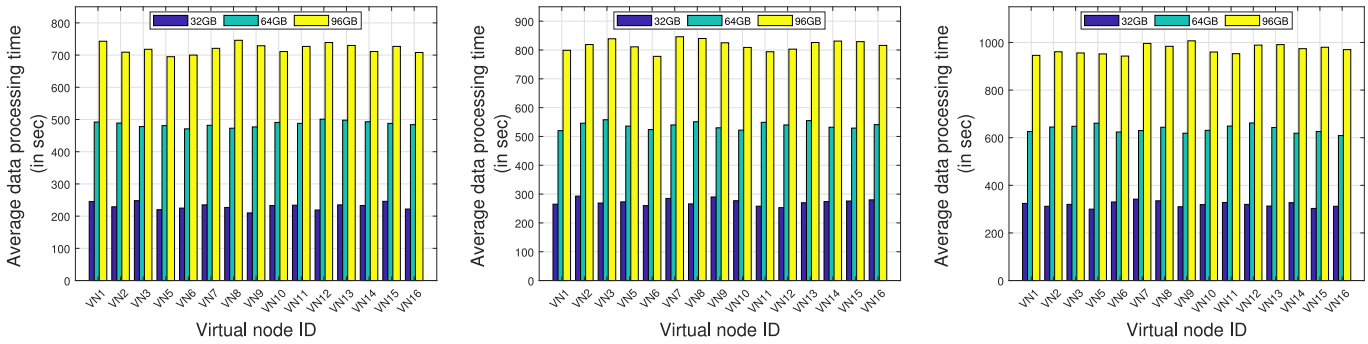
RENDA improves cloud performance through efficient data placement, and it is not designed as the overall resource manager. The recent cloud architectures such as YARN, BORG, and Fuxi act as resource managers and address broader issues such as workload partitioning, job scheduling, and resource allocation. On the contrary, RENDA solely focuses on efficient partitioning of the

TABLE 5
The Evaluation Results of RENDA With Respect to the Percentage Reduction in Non-Local Execution of Tasks

	RENDA		
	WordCount	Grep	Sort
Hadoop	28.0%	25.9%	25.5%
DRAW	19.5%	20.5%	18.3%
DDP	11.9%	13.0%	12.4%
LABP	10.4%	12.2%	10.9%
OBRU	9.1%	9.7%	10.6%

TABLE 6
The Evaluation Results of RENDA With Respect to the Average Reduction in Job Completion Time

	RENDA		
	WordCount	Grep	Sort
Hadoop	18.6%	25.0%	29.0%
DRAW	13.5%	22.7%	23.2%
DDP	9.3%	13.1%	13.9%
LABP	10.6%	12.6%	12.3%
OBRU	8.7%	11.8%	10.5%



(a) WordCount (b) Grep (c) Sort

Fig. 15. Evaluation of load balance distribution of RENDA with respect to the data processing time.

periodic workloads and distributing them to nodes considering the dominant factors of cloud, such as network delay and concurrency. RENDA benefits the existing architectures, and it can be used as an external plug-in to address the data placement issue efficiently.

One interesting immediate extension to the problem posed in this paper is to validate using the DLT model. Although the DLT model attempts to derive optimal data sizes under constant network and processing speeds, when applied in our problem context, the processing time derived using DLT model could serve as a useful estimate on time performance, which could also give an estimate on the amount of resources to be deployed.

7 Conclusion

In this paper, a Resource and Network-aware Data Placement Algorithm (RENDA) is proposed. At the center of the RENDA is the fine-tuned overlapped execution of the data distribution stage and data processing stage, which overcomes the sequential execution compulsion of MR jobs and improves the cloud performance in terms of job completion time. The performance of RENDA is largely dependent on the estimation of the remaining time of the nodes and subsequent data blocks distribution. RENDA employs Weighted Moving Average with $m = 7$ past observations that give an improved estimation of the remaining time for Grep and Sort applications. Moreover, RENDA allocates the subsequent data blocks to nodes utilizing relative performance, which ensures timely data availability and streamlines the data distribution and processing stages. The RENDA is implemented on 16 nodes heterogeneous virtual Hadoop cloud and subsequently evaluated on three periodical batch applications. The experimental results show that the RENDA reduces the data transfer overhead up to 28 percent, thereby reducing the average job completion time up to 16 percent with an average speedup of 27 percent over default Hadoop policy. To be specific, RENDA consistently outperforms the existing state-of-the-art alternatives.

Acknowledgments

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan under Grant 109-2221-E-182-014.

References

[1] W. Yang, G. Wang, K.-K. R. Choo, and S. Chen, "HEPart: A balanced hypergraph partitioning algorithm for big data applications," *Future Gener. Comput. Syst.*, vol. 83, pp. 250–268, 2018.

[2] P. P. Nghiem and S. M. Figueira, "Towards efficient resource provisioning in MapReduce," *J. Parallel Distrib. Comput.*, vol. 95, pp. 29–41, 2016.

[3] Y. Xun, J. Zhang, X. Qin, and X. Zhao, "FiDoop-DP: Data partitioning in frequent itemset mining on hadoop clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 101–114, Jan. 2017.

[4] M. Bae, S. Yeo, G. Park, and S. Oh, "Novel data-placement scheme for improving the data locality of hadoop in heterogeneous environments," *Concurrency Computation: Pract. Experience*, 2020, Art. no. e5752.

[5] I. Mavridis and H. Karatza, "Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark," *J. Syst. Softw.*, vol. 125, pp. 133–151, 2017.

[6] F. Xu, F. Liu, and H. Jin, "Heterogeneity and interference-aware virtual machine provisioning for predictable performance in the cloud," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2470–2483, Aug. 2016.

[7] A. T. Haghighat and M. Shajari, "Service integrity assurance for distributed computation outsourcing," *IEEE Trans. Serv. Comput.*, vol. 13, no. 6, pp. 1166–1179, Nov./Dec. 2020.

[8] B. Hou and F. Chen, "GDS-LC: A latency-and cost-aware client caching scheme for cloud storage," *ACM Trans. Storage (TOS)*, vol. 13, no. 4, 2017, Art. no. 40.

[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] A. Redondi, M. Cesana, M. Tagliasacchi, I. Filippini, G. Dán, and V. Fodor, "Cooperative image analysis in visual sensor networks," *Ad Hoc Netw.*, vol. 28, pp. 38–51, 2015.

[11] G. N. Iyer, B. Veeravalli, and S. G. Krishnamoorthy, "On handling large-scale polynomial multiplications in compute cloud environments using divisible load paradigm," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 48, no. 1, pp. 820–831, Jan. 2012.

[12] D. Millot and C. Parrot, "Optimization of the processing of data streams on roughly characterized distributed resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1415–1429, May 2015.

[13] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, May 2003.

[14] X. Wang and B. Veeravalli, "Performance characterization on handling large-scale partitionable workloads on heterogeneous networked compute platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2925–2938, Oct. 2017.

[15] X. Wang, B. Veeravalli, and M. Haiming, "On the design of a time, resource and energy efficient multi-installment large-scale workload scheduling strategy for network-based compute platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1120–1133, May 2019.

[16] Z. Tang, X. Zhang, K. Li, and K. Li, "An intermediate data placement algorithm for load balancing in spark computing environment," *Future Gener. Comput. Syst.*, vol. 78, pp. 287–301, 2018.

[17] C. Guerrero, I. Lera, and C. Juiz, "Migration-aware genetic optimization for MapReduce scheduling and replica placement in hadoop," *J. Grid Comput.*, vol. 16, pp. 265–284, 2018.

[18] M. Wasi-ur Rahman , N. S. Islam, X. Lu, and D. K. D. Panda, "A comprehensive study of MapReduce over lustre for intermediate data placement and shuffle strategies on HPC clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 633–646, Mar. 2017.
[19] 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.
[20] C.-W. Lee, K.-Y. Hsieh, S.-Y. Hsieh, and H.-C. Hsiao, "A dynamic data placement strategy for hadoop in heterogeneous environments," *Big Data Res.*, vol. 1, pp. 14–22, 2014.
[21] J. Wang, Q. Xiao, J. Yin, and P. Shang, "DRAW: A new datagRouping-AWare data placement scheme for data intensive applications with interest locality," *IEEE Trans. Magn.*, vol. 49, no. 6, pp. 2514–2520, Jun. 2013.
[22] B. Wang, J. Jiang, and G. Yang, "ActCap: Accelerating MapReduce on heterogeneous clusters with capability-aware data placement," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 1328–1336.
[23] H. Zheng, Z. Wan, and J. Wu, "Optimizing MapReduce framework through joint scheduling of overlapping phases," in *Proc. 25th Int. Conf. Comput. Commun. Netw.*, 2016, pp. 1–9.
[24] J. Li, J. Wang, B. Lyu, J. Wu, and X. Yang, "An improved algorithm for optimizing MapReduce based on locality and overlapping," *Tsinghua Sci. Technol.*, vol. 23, no. 6, pp. 744–753, 2018.
[25] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li, "Diamond sketch: Accurate per-flow measurement for big streaming data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2650–2662, Dec. 2019.
[26] Y. Li, X. Tang, W. Cai, J. Tong, X. Liu, and G. Wang, "Resource-efficient index shard replication in large scale search engines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2820–2835, Dec. 2019.
[27] M. Washha, A. Qaroush, M. Mezghani, and F. Sedes, "Unsupervised collective-based framework for dynamic retraining of supervised real-time spam tweets detection model," *Expert Syst. Appl.*, vol. 135, pp. 129–152, 2019.
[28] R. da Rosa Righi, E. Correa, M. M. Gomes, and C. A. da Costa, "Enhancing performance of iot applications with load prediction and cloud elasticity," *Future Gener. Comput. Syst.*, vol. 109, pp. 689–701, 2020.
[29] B. L. Dalmazo, J. P. Vilela, and M. Curado, "Online traffic prediction in the cloud," *Int. J. Netw. Manage.*, vol. 26, no. 4, pp. 269–285, 2016.
[30] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.
[31] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. 8th USENIX Conf. Operating Syst. Des. Implementation*, 2008, pp. 29–42.

Hiren Kumar Thakkar (Member, IEEE) received the MTech degree from IIIT Bhubaneswar, India, in 2012 and the PhD degree from the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan, in 2018. From 2019 to 2021, he was an assistant professor with the Department of Computer Science Engineering, Bennett University, India. He is currently an assistant professor with the Department of Computer Science and Engineering, SRM University, Andhra Pradesh, India. He has reviewed papers for several reputed journals including the *IEEE Transactions on Parallel and Distributed Systems*. His research interests include biomedical big data analysis, cloud resource management and optimization, and machine learning.

Prasan Kumar Sahoo (Senior Member, IEEE) received the BSc degree (Hons.) in physics and the MSc degree in mathematics Utkal University, Bhubaneswar, India, in 1987 and 1994, respectively, the MTech degree in computer science from the Indian Institute of Technology Kharagpur, India, in 2000, the first PhD degree in mathematics from Utkal University in 2002, and the second PhD degree in computer science and information engineering from the National Central University, Taiwan, in 2009. He is currently a professor with the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. He was an adjunct associate researcher with the Department of Cardiology and an adjunct researcher with the Division of Colon and Rectum Cancer, Chang Gung Memorial Hospital, Linkou, Taiwan. He is currently an adjunct researcher with the Department of Neurology, Chang Gung Memorial Hospital, Linkou, Taiwan. From 2007 to 2011, he was an associate professor with the Department of Information Management, Vanung University, Taiwan. He was also a visiting associate professor with the Department of Computer Science, Universite Claude Bernard Lyon 1, Villeurbanne, France. His current research interests include artificial intelligence, big data analytic, cloud computing, and IoT. He is an editorial board member of the *Journal of Network and Computer Applications*, *International Journal of Vehicle Information*, and *Communication Systems* and was the lead guest editor of the special issue of *Electronics* journal. He was the program committee member of several IEEE and ACM conferences.

Bharadwaj Veeravalli (Senior Member, IEEE) received the BSc degree in physics from Madurai-Kamaraj University, India, in 1987, the master's degree in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1991, and the PhD degree from the Department of Aerospace Engineering, Indian Institute of Science, in 1994. He is currently a tenured associate professor with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) Division, National University of Singapore, Singapore. His main research interests include cloud/grid/cluster computing, which include big data processing, analytics, and resource allocation, scheduling in parallel and distributed systems, cybersecurity, and multimedia computing. He is one of the earliest researchers in the field of divisible load theory. He is currently on the editorial board of the *IEEE Transactions on Parallel and Distributed Systems* as an associate editor. He is a senior member of the IEEE and the IEEE-CS. He was the recipient of Gold Medals for his bachelor's degree overall performance and outstanding PhD thesis (IISc, Bangalore, India), in 1987 and 1994, respectively.