




# A Novel Data Management Scheme in Cloud for Micromachines

Gurwinder Singh <sup>1</sup> , Rathinaraja Jeyaraj <sup>2,3</sup> , Anil Sharma <sup>4</sup> and Anand Paul <sup>3,\*</sup> 

<sup>1</sup> Department of Computer Science and Applications, Sikh National College, Banga 144505, India; gurwinder.11@gmail.com

<sup>2</sup> Department of Computer and Information Sciences, University of Houston-Victoria, Victoria, TX 77901, USA; jrathinaraja@gmail.com

<sup>3</sup> School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Republic of Korea

<sup>4</sup> School of Computer Applications, Lovely Professional University, Punjab 144001, India; anil.19656@lpu.co.in

\* Correspondence: paul.editor@gmail.com

**Abstract:** In cyber-physical systems (CPS), micromachines are typically deployed across a wide range of applications, including smart industry, smart healthcare, and smart cities. Providing on-premises resources for the storage and processing of huge data collected by such CPS applications is crucial. The cloud provides scalable storage and computation resources, typically through a cluster of virtual machines (VMs) with big data tools such as Hadoop MapReduce. In such a distributed environment, job latency and makespan are highly affected by excessive non-local executions due to various heterogeneities (hardware, VM, performance, and workload level). Existing approaches handle one or more of these heterogeneities; however, they do not account for the varying performance of storage disks. In this paper, we propose a prediction-based method for placing data blocks in virtual clusters to minimize the number of non-local executions. This is accomplished by applying a linear regression algorithm to determine the performance of disk storage on each physical machine hosting a virtual cluster. This allows us to place data blocks and execute map tasks where the data blocks are located. Furthermore, map tasks are scheduled based on VM performance to reduce job latency and makespan. We simulated our ideas and compared them with the existing schedulers in the Hadoop framework. The results show that the proposed method improves MapReduce performance in terms of job latency and makespan by minimizing non-local executions compared to other methods taken for evaluation.

**Keywords:** cyber-physical system; data block placement; data locality; MapReduce scheduling



**Citation:** Singh, G.; Jeyaraj, R.; Sharma, A.; Paul, A. A Novel Data Management Scheme in Cloud for Micromachines. *Electronics* **2023**, *12*, 3807. <https://doi.org/10.3390/electronics12183807>

Academic Editor: Gyu Myoung Lee

Received: 25 July 2023

Revised: 31 August 2023

Accepted: 4 September 2023

Published: 8 September 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

A micromachine is a miniature version of a traditional device, typically on a micrometer or millimeter scale. Micromachines have a wide range of applications in CPS, as shown in Figure 1, including aerospace, automotive, healthcare, industry, consumer electronics, etc. A microsensor, for example, can measure physical, chemical, or biological properties such as temperature, pressure, or chemical composition. Because data is generated rapidly and in large quantities, for example, transportation management [1], it is not possible to store, analyze, and make decisions at the physical layer. Therefore, data is transferred via networking devices in the communication layer and processed by cloud-influenced data processing platforms [2], such as dew, edge, mist, and fog computing layer through stream/query processing using distributed processing tools such as Storm, S4, Kafka, etc. However, these layers are not capable of storing and batch processing such massive amounts of data. Thus, the collected data are stored in the cloud data center (CDC) and processed using the Hadoop framework [3], which provides Hadoop Distributed File Systems (HDFS) and MapReduce to store and process large datasets. Today, HDFS and MapReduce are offered as services [4] from the cloud on a pay-per-use basis on a cluster of

VMs. However, heterogeneity in cloud execution environments [5] inhibits MapReduce performance significantly. Heterogeneity occurs in different ways in cloud virtual execution environments [6]: hardware heterogeneity, VM heterogeneity, performance heterogeneity, and workload heterogeneity. In recent work [5], one or more of these heterogeneities have been exploited to improve the job latency and makespan for a batch of workloads. As Hadoop 2.0 allows users to customize the map/reduce task container size for each job in a batch, schedule problems become more complicated. In particular, HDFS data block placement and locality aware map task scheduling play a crucial role in improving MapReduce job scheduler performance. Placement of data blocks in HDFS is carried out as follows. Data collected from the CPS environment are transferred over the Internet and stored on the cluster by HDFS in blocks of a predefined size. These data blocks are replicated [7] to ensure fault tolerance by taking advantage of rack awareness in the physical cluster. In contrast, when a cluster of VMs is hosted on different physical machines (PMs) across the CDC, ensuring fault tolerance with data replication for HDFS is not possible. Additionally, it has a significant impact on the execution of locality based map tasks. When all VMs are hosted within a rack, the network cost to perform non-local execution is negligible. There is, however, no guarantee of rack awareness [7]. In this case, once the data blocks are placed in the VMs, it is rarely expected to move them around the cluster again to perform non-local execution, which involves additional network costs. Therefore, distributing VMs across the CDC guarantees rack awareness with additional network costs for non-local executions. Hence, it is important to minimize the number of non-local executions when the VMs in the virtual cluster are distributed in the physical cluster.

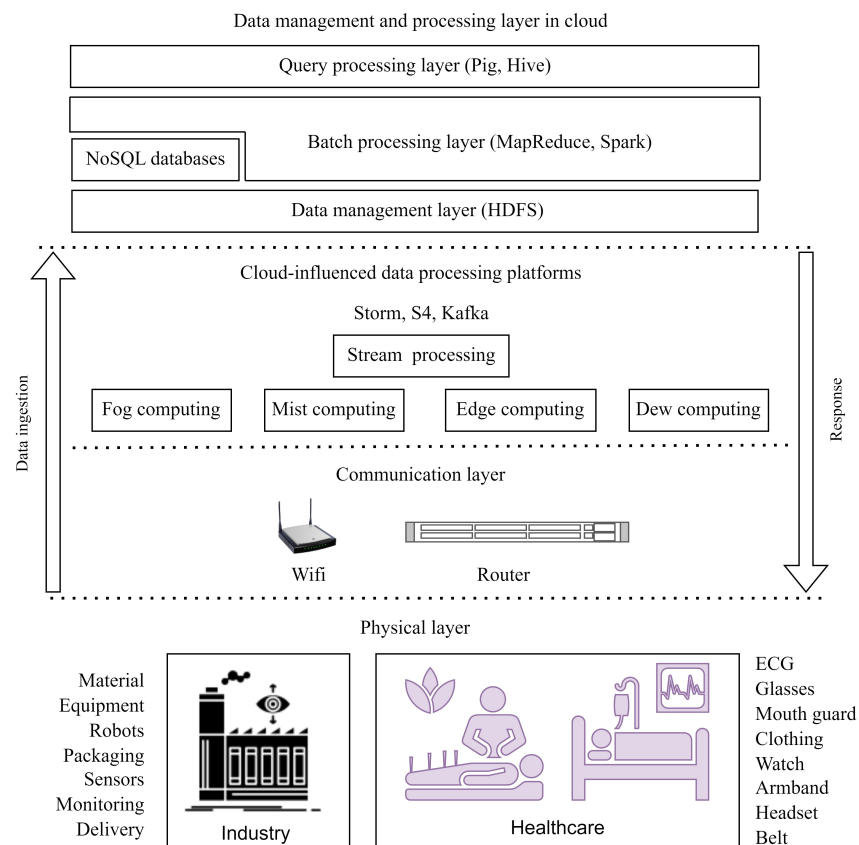


Figure 1. Big data management for CPS applications.

Second, open source MapReduce schedulers [8,9] are mostly driven by resource availability in the virtual cluster. Due to the highly heterogeneous capacity and performance of PMs in the CDC, VMs also exhibit heterogeneous performance. It is possible for the storage of a set of VMs to be allocated to the same hard disk drive (HDD), even if multiple

HDDs are available on the same PM. In the cloud system, there is no option by default to allocate storage space from specific HDDs to VMs. In the worst-case scenario, VMs share a single HDD. The result is varying latency for map tasks due to disk IO contention/races among VMs hosted on the PM. Due to this, data local execution is severely affected since disk IO is mostly used to serve data for other tasks from other VMs. At this moment, performing non-local execution obviously consumes additional network bandwidth and time. In addition, the nature and resource requirements of map tasks in MapReduce for different jobs are also highly heterogeneous. Sometimes, a map task could be CPU or memory intensive and the container size for map tasks from different jobs could also differ. Therefore, it is necessary to exploit the dynamic performance of VMs based on the map task requirements of different jobs to achieve data local execution.

Moreover, disk IO becomes a bottleneck when Hadoop/non-Hadoop VMs share a single HDD among them. In general, big data batch processing (MapReduce and Spark) jobs tend to be more disk-intensive, requiring constant disk bandwidth to bring input data blocks for map task execution. However, each VM hosted on a PM shares the disk bandwidth for different tasks. This causes map task latency to increase. Therefore, data blocks must be stored based on HDD performance, so map tasks receive input data blocks seamlessly. Moreover, as VMs in the hired virtual cluster could be of a different flavor, the number of data blocks processed per unit of time varies greatly. Hence, it is important to understand the capacity and processing performance of different VMs before loading data blocks. To achieve this, we first determine the performance of individual disks on PMs, then choose the most suitable disk for storing data blocks. Based on disk performance, the number of data blocks to be stored in each VM is calculated. Therefore, understanding the performance of each VM for map tasks of different jobs could help schedule map/reduce tasks in the right VM. The reason for this is that map tasks require data to be localized as much as possible without switching them around the cluster. In contrast, reduce tasks require minimal network bandwidth consumption. This can improve MapReduce job latency and makespan when MapReduce jobs are periodically submitted as a batch.

Our major contributions from this paper are summarized below.

1. In MapReduce, it is very important to minimize non-local executions for map tasks in order to minimize latency and makespan. Due to co-located VM interference, map tasks fail to receive their required data blocks on time when HDD contention is high in a PM. At this point, it is impossible to avoid non-local execution. In order to overcome this problem, we predict the IO performance of every HDD in PMs in the CDC before loading data blocks from the CPS environment. Because IO contention is directly correlated with disk IO performance, linear regression is used to predict disk IO performance to place data blocks. This minimizes non-local executions for map tasks, which ultimately minimizes job latency and makespan for a batch of MapReduce jobs.
2. Furthermore, the performance of VMs that host Hadoop MapReduce is also impacted by co-located VMs that interfere with resource sharing. Since map tasks from different jobs have different resource requirements, it is important to allocate map tasks to the right VM. Consequently, varying performance Hadoop VMs are observed and ranked for scheduling map tasks to minimize job latency.

The remainder of the paper is organized as follows. In Section 2, we give a brief overview of Hadoop MapReduce and the motivations behind our work. Section 3 discusses the related works on data local execution for map tasks and the dynamic performance of VMs. Subsequently, the proposed methodologies are justified and explained in Section 4. The results and analysis of our proposed method are presented in Section 5. Finally, Section 6 includes concluding remarks.

## 2. Hadoop MapReduce Background

In this section, we offer a brief overview of Hadoop's framework for managing big data. The Hadoop framework consists of two major components: HDFS and MapReduce. The HDFS manages huge data across multiple servers and feeds it to MapReduce for

processing. MapReduce jobs consist of two phases: map and reduce, as shown in Figure 2. A map phase consists of a set of map tasks. A map task processes a set of data blocks from HDFS and produces an arbitrary size of intermediate output. A set of reduce tasks is launched during the reduce phase, depending on how much parallelism we intend to extract. Reduce tasks collect and consolidate portions of the map tasks' output. Prior to executing the reduce function, the reduce tasks perform a series of steps (merge, sort, group). From the group function, the reduce function receives a list of inputs and writes the output to HDFS. As part of this execution sequence, the output of map tasks is moved to reduce tasks through the process of "shuffle/copy". It is possible to start the reduce phase simultaneously with the map phase. However, the reduce function in the reduce phase can be executed only after the map phase is finished. Physical resources for MapReduce jobs are allocated by a component called YARN, which comprises two modules: resource manager (RM) and node manager (NM). RM distributes clustered server resources (CPU, RAM, storage) between different frameworks (such as Hadoop and Spark). The NM manages local resources on the server according to the size of each map and reduce task.

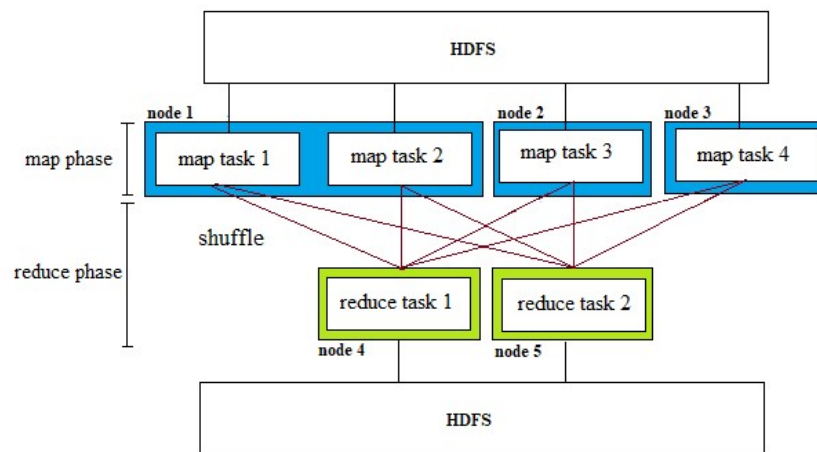


Figure 2. MapReduce Phases.

### 3. Related Works

With the increasing use of cloud-based data processing services to support CPS applications, managing virtual resources to improve MapReduce job latency and makespan has become difficult. Therefore, big data processing with tools offered by the cloud is increasingly becoming a research hotspot. Different block placement and MapReduce job/tasks scheduling play a vital role in improving job latency and makespan. Over a decade, there have been many data block placement algorithms proposed to improve makespan and novel scheduling algorithms [10] to exploit dynamic performance to improve job latency in a cloud environment. In this section, we discuss some of the significant previous works on data placement and MapReduce job scheduling exploiting heterogeneity in cloud environments.

Distributed file systems, such as Google File System [11] and HDFS [12], divide input datasets into fixed-size blocks. The Modulo-based data block placement algorithm [13] is introduced for block placement to minimize energy consumption and improve makespan in a virtual cluster environment. It is performed in terms of CPU-intensive, IO-intensive, and interactive jobs. Roulette wheel scheme (RWS)-based data block placement and heuristic-based MapReduce job scheduler (HMJS) are proposed in [5] to enhance data local execution for map tasks, job latency, and makespan. In this work, the authors consider the computing capacity of VMs to place data blocks but not the heterogeneous performance of VMs. In addition, disk IO performance largely affects map phase latency, even though CPU and memory are available for map tasks. To handle this, constrained two-dimensional bin packing is proposed to place heterogeneous map/reduce tasks to minimize job latency and makespan. In [14], the authors propose a workflow level-based data block placement



in a cloud environment to optimize the data sharing among multiple jobs in a batch. To manage geo-distributed workloads, the authors in [15] place the data based on the computing capacity of nodes in the CDC. These data blocks are typically stored based on rack awareness in case of a physical cluster. This leads to a varying number of data blocks on each physical machine in the cluster, as shown in Figure 2. However, in a cloud virtual environment, there is no rack awareness [16] as VMs might be hosted in the same PM or different PMs within a rack. This results in an unequal number of data blocks in each VM, regardless of its performance and location. A detailed description of different replication strategies in distributed file systems on the cloud is mentioned in [17,18].

To improve data local execution in a heterogeneous environment, a sampling-based randomized algorithm is proposed in [19]. In this work, the authors estimate the workload arrival at runtime for a set of candidate nodes in the cluster. The node that is estimated with the smallest workload in the future is chosen to place data blocks. A dynamic data replication strategy is proposed in [20] to minimize network bandwidth consumption and latency. The authors consider a set of characteristics, such as number of replicas, dependency between datasets, and the storage capacity to decide the lifetime of a replica. A similar approach is applied in [7] to retain the data blocks that are most wanted. Interestingly, the authors in [21] place data blocks based on the availability of cache memory in the server. A location-aware data block placement strategy for HDFS is devised in [22], by identifying virtual nodes' processing capacities in the cloud. To take advantage of data local executions, data locality aware scheduling is performed in [23]. A similar approach is carried out in [24], with the focus of minimizing the total data transfer cost at the time of non-local execution. In an IoT environment, data blocks located on different decentralized nodes are processed and the results are combined. To achieve this, the authors in [25] proposed an edge-enabled block replica strategy that stores in-place, partition-based, and multi-homing block replicas on respective edge nodes. To estimate the workers' and tasks' future resource consumption, a Kernel Density Estimation and Fuzzy FCA techniques are used in [26] to cluster data partitions. Fuzzy FCA is also used to exclude partitions and jobs that require less resources, which will reduce needless migrations. In another work [27], the authors place data blocks based on the evaluation indicators, such as node's disk space capacity, memory and CPU utilization, rack load rate, and network distance between racks.

In a production environment, a batch of jobs is periodically executed to extract insight from huge data in physical/virtual clusters at different times. The nature of jobs would not change much and reveal more information about workload behavior. Fair scheduler [8] equally shares the underlying resources to all the jobs in the batch, resulting in an equal chance for each job. However, if a job is idle and waiting for the resources, the resources allocated to that job are held idle until the job completion. To provide resources based on the job's requirements, capacity scheduler [9] is introduced to define the resources required for each job in the batch. Like fair scheduler, capacity scheduler also holds resources unused when the job is waiting for other resources. In addition, both fair and capacity schedulers do not consider the heterogeneity in the underlying hardware resources. To adopt various dynamic parameters in big data applications to improve energy efficiency, workload analysis [28] is conducted to select the optimal configuration and system parameters. They used micro-benchmarks and real-world applications to demonstrate the idea proposed. In this study, a variety of processing elements, as well as system and Hadoop configuration parameters, are tested. These metrics emphasize the performance of the MapReduce scheduler. Identifying the right combination of these parameters is a challenging task that cannot be performed at the time of execution. Therefore, Metric Important Analysis (ensemble learning) was carried out by Zhibin Yu et al. [29] using MIA-based Kiviat Plot (MKP) and Benchmark Similarity Matrix (BSM). This produces more insight than traditional-based dendrograms to understand job behavior by using three different benchmarks: iBench, CloudRank-D, and SZTS.

Heterogeneity at different levels of the cloud environment is considered before data block distribution in the virtual cluster by using a framework, called MRA++ [30]. This

method uses some sample map tasks to gather information on the capacity and performance of individual nodes in the cluster. If a node is underperforming, it does not attract compute-intensive tasks. Thus, stragglers are avoided. Balancing data load among the nodes in the Hadoop cluster is very difficult to determine in a heterogeneous environment as the performance of individual nodes varies significantly. A novel data placement technique was proposed by Vrushali Ubarhande et al. [31] to minimize makespan in a heterogeneous cloud environment. Computing performance is determined for each virtual node using some heuristics, and data blocks are placed accordingly. The authors claim that data locality and makespan are compared to classical methods. To balance optimal load across the virtual cluster, a topology-aware heuristic algorithm [32] was designed to minimize non-local execution for map tasks and minimize global data access during the shuffle phase. Here, the computation cost was minimized up to 32.2% compared to classical MapReduce schedulers.

In light of these related works, the following observations are made. To improve the performance of data local execution, existing works rely mostly on establishing the processing performance or capacity of the VM. However, they do not consider disk IO performance in a PM. Moreover, the performance of VMs fluctuates due to co-located VM resource consumption interference. Therefore, in this paper, we propose the following works to increase the data local executions, thereby minimizing job latency and makespan in a cloud virtualized environment. As

1. disk IO load is directly correlated to the overall HDD performance, we employ simple linear regression algorithm to predict HDD performance based on the number of IO operations performed over time.
2. VM performance varies due to the co-located VM's interference, map tasks are scheduled based on the VM performance in terms of CPU and disk IO.

#### 4. Proposed Methods

In this section, performance of disk IO is predicted and used for placing the data blocks to ensure data location. This helps MapReduce tasks execute without waiting for disk access when memory and CPU are available. In particular, map tasks benefit from this approach by minimizing the number of non-local executions. In addition, to improve map task performance further, map tasks are placed based on the heterogeneous performance of VMs in different PMs.

##### 4.1. Predicting Disk IO Performance to Place Data Blocks Using Regression

###### 4.1.1. Problem definition

A numerical predictive problem is defined as follows: given a number of read + write access of a disk, performance of an HDD is predicted. According to the performance of each HDD, a set of data blocks is distributed on the cluster VMs.

###### 4.1.2. Problem formulation

Generally, the Hadoop framework is offered as a service on a cluster of VMs, hosted on different PMs with different configurations in a CDC, as shown in Figure 3. A CDC consists of  $t$  racks (Rack1, Rack2, ..., Rack $t$ ), each with  $u$  PMs (PM1, PM2, ..., PM $u$ ). These PMs host Hadoop and non-Hadoop VMs. In Figure 3, a cluster of  $v$  Hadoop VMs (VM1, VM2, ..., VM $v$ ), highlighted in shade, is assumed. HDFS services such as name node (NN), secondary name node (SNN), and data node (DN) are run on different VMs. YARN components such as RM and NM are also serving to the MapReduce jobs. These Hadoop VMs in the virtual cluster are distributed across the CDC based on resource availability in PMs, which are generally of varying capacities and performance. Therefore, heterogeneity in the physical environment is unavoidable, as it causes varying performance for the same task at different runs. Sometimes, a set of VMs might be allocated with the same HDD on a PM even though multiple HDDs are available. In this case, though CPU and memory capacity are available for launching map tasks, due to IO contention by co-located

VMs on the same HDD, it takes time to bring data blocks into memory for map tasks. In Figure 4, disk IO consumption every 5 s is shown for four cases on a wordcount job to understand how map tasks consume disk bandwidth: HDFS access with interference, HDFS access without interference, local file system (LFS) access with interference, and LFS access without interference. From Figure 4, it is evident that the map task latency of a wordcount job increased by up to 50% due to co-located VM interference when compared to map task execution with no interference. Because of co-located VM interference, the map task is held idle for a significant amount of time while waiting for disk IO. As a result, job latency increases and resources are underutilized. Figure 5 shows that when co-located VM interference increases, CPU and network resources are mostly unused, while disk IO is interfered with by co-located VMs. This indicates that the hired resources are not properly utilized, and job latency deteriorates. This influences overall job latency and the number of local map tasks executed. Therefore, it is important to consider HDD performance before distributing data blocks.

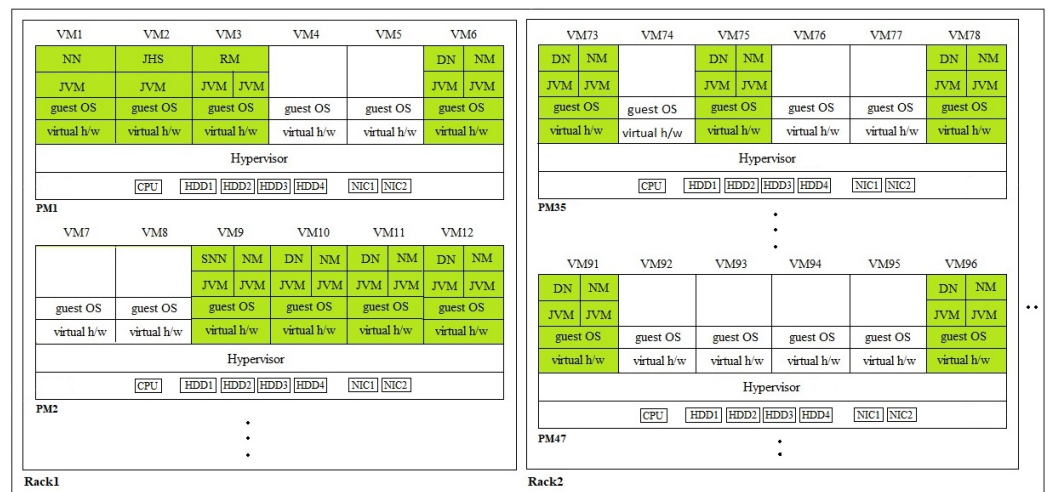


Figure 3. Hadoop virtual cluster.

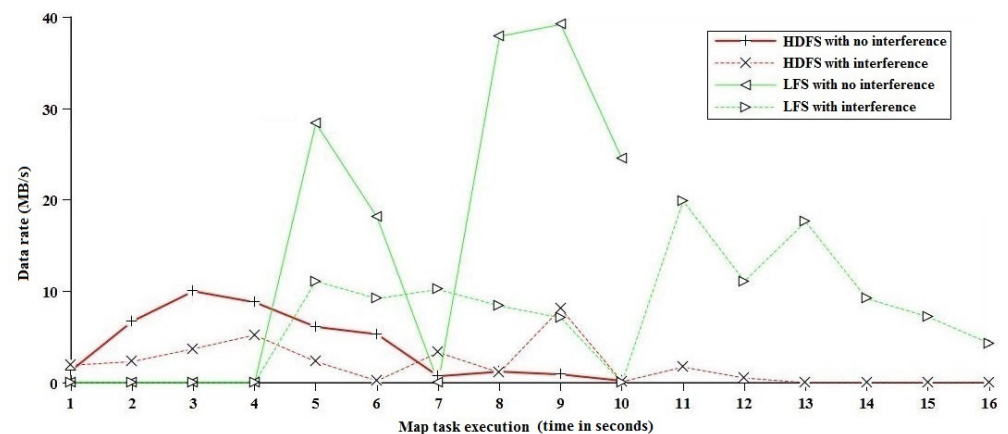


Figure 4. Disk IO consumption for map task in wordcount job during co-located VM interference.

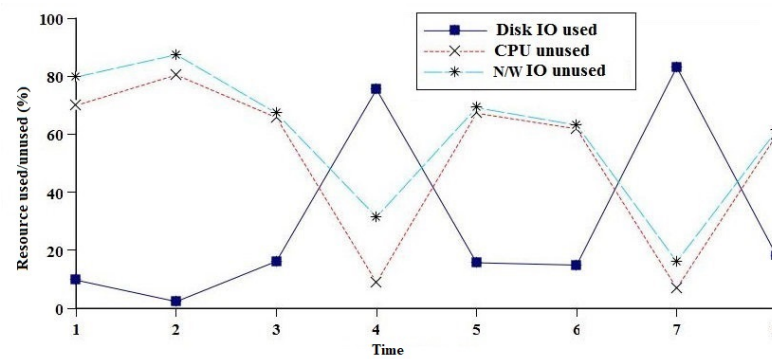


Figure 5. Unused CPU and network resources due to disk IO contention in each PM.

Once the big data is uploaded onto HDFS, it divides the input file into  $n$  equal sized blocks (along with replication, in total  $N = 3 * n$  number of blocks) and identifies the target VM for storing the data block. Here, in general, the target VM is chosen based on the storage availability. In the proposed approach, we predict the disk IO performance based on the disk access requests to the HDDs to decide the target disk for loading data blocks. The idea is the number of disk IO access (read + write) is directly proportional to the amount of disk IO bandwidth consumed most of the time by the Hadoop VMs. Therefore, based on the disk access, we can predict the performance of a Hadoop VM. Because applications running in VMs exhibit similar behavior to disk access, we use a simple linear regression algorithm, which is a supervised learning model that helps with numerical prediction. It takes an input variable (independent variable) and produces an approximate/estimated output (dependent variable), as shown in Equation (1).

$$IOP_i^k = y = mx + c \tag{1}$$

Here,  $i$ ,  $k$ , and  $x$  indicate PM, disk number, and input value, respectively.  $IOP_i^k$  indicates  $k$ th disk in  $i$ th PM. In our case,  $x$  is defined as “number of disk IO access (read + write)” and  $y$  is the performance of every five seconds.  $m$  and  $c$  are the parameters to be estimated. While reading/writing on an HDD, some data are transferred back and forth by consuming disk bandwidth. Even though we cannot predict the behaviour of applications running inside non-Hadoop VMs, based on bandwidth consumption (in %) over time, it is possible to approximate the performance of a VM as the output variable. This is calculated by averaging the disk IO consumption rate every five seconds. By recording thousands of such samples, we build a linear model using regression. This is applied to each HDD on the PMs in the data centre. Despite many Hadoop VMs being hosted on the same HDD, data blocks are stored on the drive, which gives high performance.

In general, VMs are not migrated very frequently. Therefore, predicting IO performance of a VM using this method is useful for most of the time. By default, three copies of a data block are distributed across the virtual cluster. What if three copies are stored in three different VMs hosted on the same PM? Multiple copies of the same block are used to achieve fault tolerance, but there is no way to ensure rack awareness to achieve fault tolerance in a virtual cluster. Therefore, after predicting the IO performance of an individual disk, it is guaranteed that no copies of a block are created in VMs hosted on the same physical machine. Moreover, if disk IO performance is predicted to be high, then the number of blocks present in that VM is high, as given in Equation (2). In general, the number of data blocks (DB) stored in each VM is the same. However, in the proposed approach, the number of data blocks available on a disk for a VM is directly proportional to the performance of disk IO on that PM. If the performance is the same in all HDDs, then an equal number of data blocks is stored in each VM.

$$\forall_{i,j,k} (DB)_{i,j}^k = \left[ (IOP)_i^k * e \right] \tag{2}$$

Here,  $i, j$ , and  $k$  indicate PM, VM, and disk number, respectively;  $e$  denotes the number of data blocks possible on the respective HDD based on the available storage. Algorithm 1 provides a summary of the proposed prediction-based block placement strategy. Figure 6 shows the overall flow of the proposed approaches. Initially, big data are uploaded onto the virtual cluster hired by dividing the data into blocks. At this point, the service provider assists the data-loading phase by providing the performance of HDDs associated with VMs in the cluster to improve data local execution. Then, while executing a batch of jobs, the map tasks in each job are scheduled based on the varying performance of VMs to minimize job latency and makespan.

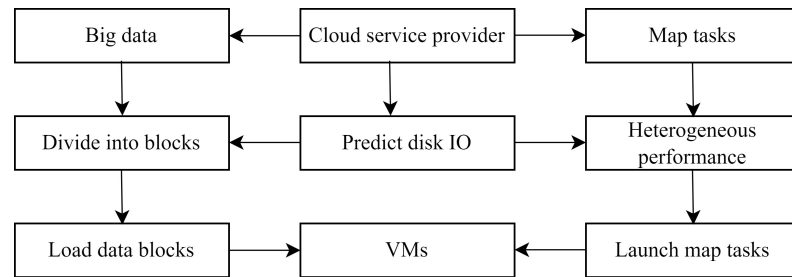


Figure 6. Prediction-based data block placement.

Whenever a virtual cluster is scaled up based on demand, the number of VMs and their type (different configurations and capacities, such as micro, min, etc.) are chosen based on the requirements. Users can redistribute blocks from heavily loaded machines to lightly loaded machines to balance the cluster load. As a result, the data blocks are moved in accordance with the disk IO performance of the PM that hosts the new VMs.

**Algorithm 1:** Prediction based block placement at the data loading stage

```

Input :  $N$  data blocks,  $IOP$ 
Output :  $DB$ 
for  $w = 1 \dots N$  do
  for  $j = 1 \dots |VM|$  do
    for  $k = 1 \dots |HDD|$  do
      if  $DB_{i,j}^k \leq DB_{i,j}^e$  then
         $DB_{i,j}^k = w$ 
         $DB_{i,j}^k ++$ 
        continue  $w$ 
      end
    end
  end
end
  
```

4.2. Scheduling Map Tasks Based on the Heterogeneous Performance of VMs

Once the data blocks are loaded based on the performance of disk IO, MapReduce jobs can be launched on them. Each map task has three options as the replication factor is 3, by default. Therefore, a map task can be executed in one of the three VMs in which a specific block has been stored. To decide this, the dynamic performance of each VM is computed, based on which map tasks are launched. This idea might seem arbitrary while dealing with with one job. However, when a batch of jobs is executed, map tasks from different jobs might be configured with different resource capacity. Therefore, choosing the right VM for executing a map task is very important.

Typically, Hadoop VMs are co-located with non-Hadoop VMs, which affects the performance of the Hadoop VMs by sharing the IO resources of the underlying PM. Even though the performance of the disk is high, there could be a chance that CPU and memory



are tightly shared by co-locating VMs. Therefore, it is also important to observe that data local execution is minimized due to the unavailability of resources. Therefore, we dynamically monitor the performance of each Hadoop VM. If the performance of a VM is not significant, then the frequently accessed data blocks from that VM are migrated to another VM, which results in high performance. This way, there is a chance to increase the number of local executions, thereby improving the latency and makespan for a batch of MapReduce jobs. CPU performance of  $j$ th VM in the  $i$ th PM ( $VM_{ij}^{CPU}$ ) is calculated by finding the PM with maximum CPU frequency ( $CPU\_F$ ) among all the PMs in which Hadoop VMs have been hosted, as given in Equation (3).

$$VM_{ij}^{CPU} = \frac{VM_{ij}^{CPU\_F}}{\max(\forall PM_i^{CPU\_F})} \tag{3}$$

Equation (4) calculates the Disk IO performance of  $j$ th VM in  $i$ th PM ( $VM_{ij}^{DIO}$ ) considering a current disk bandwidth ( $CB$ ) rate of  $j$ th VM in  $i$ th PM ( $VM_{ij}^{CB}$ ) over the disk bandwidth ( $B$ ) of  $k$ th disk in  $i$ th PM ( $PM_{ik}^B$ ).

$$\forall i, j \quad VM_{ij}^{DIO} = \forall k, \frac{\sum VM_{ij}^{CB}}{PM_{ik}^B} \tag{4}$$

Map and reduce tasks from different jobs have different resource requirements. Map tasks demand more CPU and storage access, while reduce tasks need CPU and network bandwidth. Therefore, to launch map tasks in VMs, it should have seamless disk bandwidth while the job begins. In addition, it should have seamless network bandwidth while moving map outputs to reduce nodes where reduce tasks are running. To find the virtual node that is suitable for running map tasks, we calculate the influence of  $j$ th VM in  $i$ th PM for map ( $VM_{ij}^{MI}$ ) by considering the latency of the last  $z$  map tasks ( $ML$ ) executed in  $j$ th VM, using Equation (5).

$$\forall i, j, VM_{ij}^{MI} = \min \left( \forall z, \frac{ML_{jz}}{\sum_{m=1}^z ML_{jm}} \right) \tag{5}$$

Typically, overall performance of a VM is calculated regardless of the task’s type. Using Equation (6), we find map task performance ( $MP$ ) in each VM ( $VM_{ij}^{MP}$ ) based on CPU frequency and Disk IO bandwidth of respective VM hosted in each PM.

$$\forall i, j, VM_{ij}^{MP} = VM_{ij}^{CPU} \times (1 - VM_{ij}^{DIO}) \times (1 - VM_{ij}^{MI}) \tag{6}$$

Finally, VMs are sorted in descending order, using Equation (7), based on its performance to launch map tasks, or else reduce tasks could be launched in place of map tasks.

$$rank = sort(VM_{ij}^{MP}) \tag{7}$$

Algorithm 2 describes how map tasks are scheduled for a job. After calculating the map performance rank for VMs, task scheduler schedules map tasks. For the initial scheduling of map tasks, only the top 10% of map nodes are selected from the rank list. If no container is possible and no data locality is possible, the remaining 90% of nodes in the rank list are used for scheduling map tasks. However, the performance of these 90% of nodes would be relatively inferior to that of the first 10%. Non-local execution is preferred when locality cannot be achieved on any node in the rank list. This means that map tasks are performed on nodes where the data is not locally available. This approach may result in higher data transfer costs or reduced performance, but it allows tasks to be executed even when data locality cannot be achieved. When a virtual cluster is expanded by adding additional VMs with varying capacity and performance, the existing rank list becomes obsolete since the new VMs might perform better and be included in the top 10% or lower. It is therefore

essential to reevaluate Equations (3)–(7). By comparing the ranking of newly added VMs with the ranking of existing VMs, high performing VMs can be ranked relatively. Due to the dynamic nature of performance throughout the execution, it is helpful to launch map tasks on the appropriate VMs. As map task scheduler calculates dynamic performance and is highly heuristic, its computational time is constant for every VM.

---

**Algorithm 2:** Heterogeneous performance aware map task scheduling

---

```

Input : MapReduce jobs, rank
Output :  $VM_{ij} \leftarrow$  map tasks
 $m \leftarrow$  number of map tasks a job
 $M_{np}$  –  $p$ th map task of  $n$ th job
 $\forall p, M_{np} \leftarrow 0$ 
 $C_n$ —number of completed map tasks of  $n$ th job
 $C_n \leftarrow 0$ 
while  $C_n \leq m$  do
  Get a map task ( $p$ ) from  $n$ th job
  Get top 10% VMs from rank
  while until 10% nodes do
    if containers && data locality then
       $map_{np} = 1$ 
      Launch map task
       $C_n ++$ 
    end
  end
  if  $map_{np} = 1$  then
    Get remaining 90% VMs from rank
    while until 90% of nodes do
      if containers && data locality then
         $map_{np} = 1$ 
        Launch map task
         $C_n ++$ 
      end
    end
  else
    if perform non-local execution then
       $map_{np} = 1$ 
      Launch map task
       $C_n ++$ 
    else
      add  $map_{np}$  into task queue
    end
  end
end

```

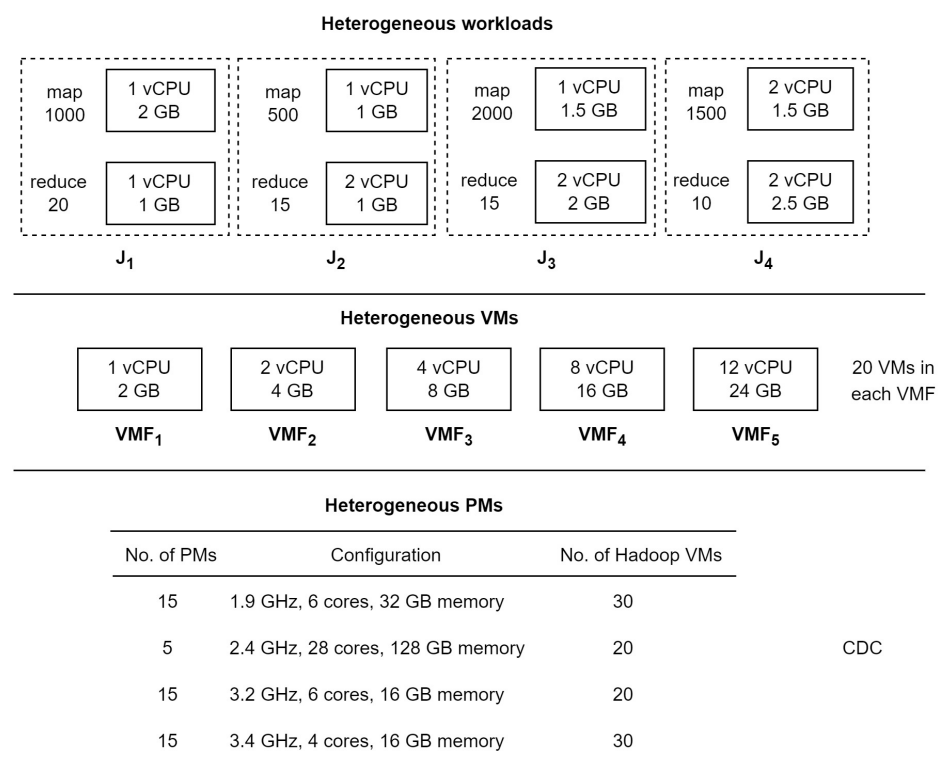
---

## 5. Results and Analysis

### 5.1. Experimental Setup

We simulate our ideas on an Ubuntu server with a 12-core CPU (hyper-threaded), 64 GB memory,  $4 \times 1$  TB HDDs, and a maximum disk bandwidth rate of 100 MB. We compare the proposed ideas with classical Fair scheduler [8], Rowlett Wheel Scheme (RWS)-based job schedulers, and Heuristics-based MapReduce Job Schedulers (HMJS) [5], based on quality-of-service parameters, such as non-local execution, job latency, and makespan, using Hadoop 2.7.0. We chose these works because they demonstrate how our proposed method performs compared to a classical scheduler without any heterogeneities and one that does. We also assume workload size and VM configuration parameters, as described below. VMs and PMs in CDC are assumed to be highly heterogeneous, as shown in Figure 7. Five different VM Flavors (VMFs) with different configurations (CPU, memory) are considered in the simulation, namely,  $VMF_1$  (1 vCPU, 2 GB),  $VMF_2$  (2 vCPU, 4 GB),  $VMF_3$  (4 vCPU,

4 GB),  $VMF_4$  (8 vCPU, 16 GB), and  $VMF_5$  (12 vCPU, 24 GB). Twenty VMs in each VMF are considered in the virtual cluster deployed in different PMs (shown in Figure 7) in random across CDC using KVM hypervisor. We have used KVM hypervisor and considered the following network parameters: ethernet cables with a maximum bandwidth of 250 MHz and supporting 10 Gbps data transfer speeds, and network switches with a capacity of 1 Gbps. The amount of network bandwidth consumed is not defined in numbers, instead it can be assumed with job latency and makespan, as we focus on disk IO access in different HDDs and do not restrict the simulation environment related to network-related parameters. We simulate disk IO interference based on normal distribution, as if co-located VMs perform random executions. The range for normal distribution is defined by the minimum and maximum disk IO access rate in each PM (as given in Section 5.2). The frequency of disk IO interference is also random, as we cannot expect the interference on certain patterns in real-time.



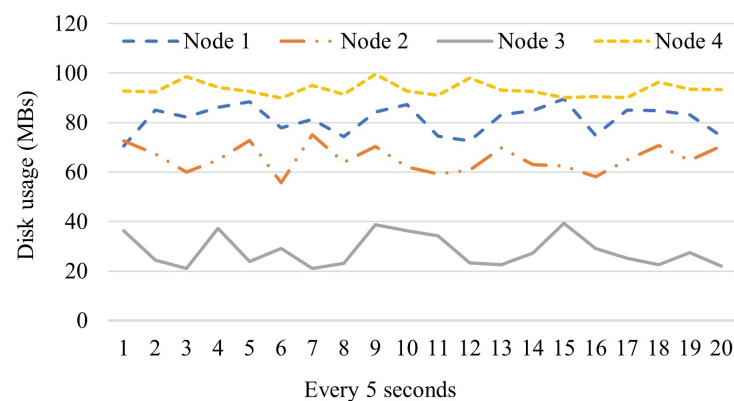
**Figure 7.** Experimental setup in the simulation.

Workloads for the experiment considered are wordcount, wordmean, wordmedian, and kmean to process datasets of sizes 128 GB, 64 GB, 256 GB, and 192 GB, respectively, totaling 640 GB. These datasets can be downloaded from [33]. Wordcount calculates the frequency of word occurrences in a file. The average length of words is calculated by wordmean job. The median length of words in a file is computed by the wordmedian job. The kmean job finds clusters in the given input data file. For all datasets, the input block size is 128 MB, and the replication factor is 3. The number of map/reduce tasks and their resource requirements is given in Table 1. One map task is assigned to a block, such that the number of map tasks for each job is 1000, 500, 2000, and 1500, respectively. Map task latency and reduce task latency are also fixed and included in the table. These latencies are approximated and taken from our lab experiments.

**Table 1.** Resource requirements of each job.

MapReduce Job	No. of Map Tasks	No. of Reduce Tasks	vCPU		Memory		Map Task Latency	Reduce Task Latency
			Map	Reduce	Map	Reduce		
wordcount ( $J_1$ )	1000	20	1	1	2	1	21	39
wordmean ( $J_2$ )	500	15	1	2	1	1	18	33
wordmedian ( $J_3$ )	2000	15	1	2	1.5	2	15	30
kmean ( $J_4$ )	1500	10	2	2	1.5	2.5	21	60

In the field of big data analytics and distributed computing, the datasets listed in Table 1 are usually used as benchmarks. These datasets and the MapReduce jobs represent a range of characteristics and properties commonly encountered in real-world applications. Inherently, they contain varying distributions of data, which can influence resource utilization. For example, a skewed distribution in the dataset can lead to resource imbalances within the virtual cluster. In addition, MapReduce jobs can differ in their resource consumption depending on the nature of the datasets. The frequency and intensity of input/output operations involved in processing the datasets should be considered. In some cases, datasets may require frequent disk access, resulting in higher disk IO resource utilization. It may be necessary to use more computational resources for certain datasets that involve complex computations or algorithms. For example, wordcount, wordmean, and wordmedian MapReduce jobs are highly IO intensive, as the size of map task output is greater than the size of its input. Therefore, disk IO access for HDFS operation is high throughout the execution, as discussed in Figure 8. In addition, reduce tasks tend to read massive input from map tasks, which require more disk IO bandwidth to distribute the output. Secondly, the kmean job involves massive computational resources as reduce task performs clustering the data points. In this job also, map tasks produce the output bigger than the size of its input. This diversity allows for a comprehensive assessment of the proposed method’s effectiveness across different data types and distributions.



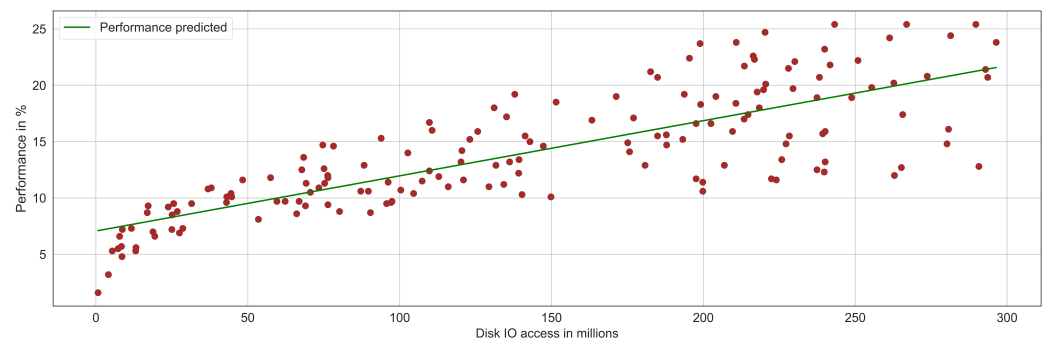
**Figure 8.** IO access pattern in different hard disk drives.

5.2. Prediction Based Block Placement

In general, the classical scheduler places data blocks across VMs evenly, but our proposed method places data blocks based on the performance of HDDs. To predict the performance of HDDs, we use the trace of disk IO access recorded every five seconds. Consider three physical machines (nodes), each with one HDD. Assume the first node hosts five VMs, the second node hosts three VMs, and the third node hosts one VM. IO-intensive applications such as web applications are run in a VM to record disk IO access behavior. Disk access on Node 1 fluctuates between 70 MB/s and 89 MB/s because of hosting five VMs. Each VM races against each other to hold disk access, such that disk IO is constantly delivering data to running tasks. If many map tasks are launched on Node 1, then bringing data blocks of a job that requires more IO access into memory takes more time. Moreover, data blocks might be in different sectors/disks on the HDD. Similarly, on Node 2, disk

access ranges between 55 MB/s and 74 MB/s as three VMs are hosted on this machine. Node 3 disk access varies between 20 MB/s and 39 MB/s. However, the disk bandwidth consumption may go down to the starting range mentioned above. Figure 8 shows the recorded disk IO access for every 5 s.

The idea here is to predict the performance of an HDD based on the estimated disk IO access required for a MapReduce job. As VMs reside on specific PMs and execute programs that exhibit similar behavior, it is possible to generalize its impact on the HDD based on the number of disk IO access and the performance. Hence, we collected 1152 samples in total from all the three nodes mentioned above, which are split into 80:20 for training and testing. In particular, we have to record samples on each HDD in the server that hosts Hadoop VMs. Then, we build a regressor model for each HDD, such that it is used for predicting its performance given input (estimated number of IO access). The approximate number of disk IO access for a particular MapReduce job can be obtained by running that job on a single block. Figure 9 shows the regressor model devised on 160 samples collected on a HDD attached to Node 3. The scatter plot displays disk IO access (in millions) and performance (in %) as coordinates. We can see various numbers of disk IO access ranging from 0 to 300 millions to showcase the performance of a HDD in different degrees.



**Figure 9.** Training phase for Node 3.

To fit a linear model on the dataset collected, values for the parameters  $m$  and  $c$  must be approximated from the dataset. Mean squared error (MSE) is the cost function typically used in regression. Parameter values that minimize the cost function, i.e., using gradient, are the values of interest. The MSE on the training set for the above model is 9.7, with the resultant model  $y = 0.049x + 7.069$ . Here, the model parameters  $m$  and  $c$  obtain 0.049 and 7.069, respectively. The model error rate would change based on the HDD performance for a given disk IO access, which depends on the particular PM running co-located non-Hadoop VMs, hypervisor, and the host OS. The predicted accuracy is displayed in Figure 10 with the training set. The MSE with the testing set is 13.5, which shows that the better predicted accuracy is coherent with the training set. Figure 11 shows the residual plot of the predicted accuracy. It can be observed that the model has performed well closed to the observed values by minimizing the deviations with the predicted disk IO performance below 15 MB/s. The variation (residuals above to 3) that is seen with higher disk IO performance can be explained with a higher number of samples obtained in the same range. However, it is good enough and helpful in placing data blocks. In addition, collecting more samples for the varying disk IO performance can help to estimate the model even better. As a final argument, including a higher number of features related to disk access to infer its performance is possible. However, the simple linear model with the number of disk IO access abstract all the lower level features related to HDD.



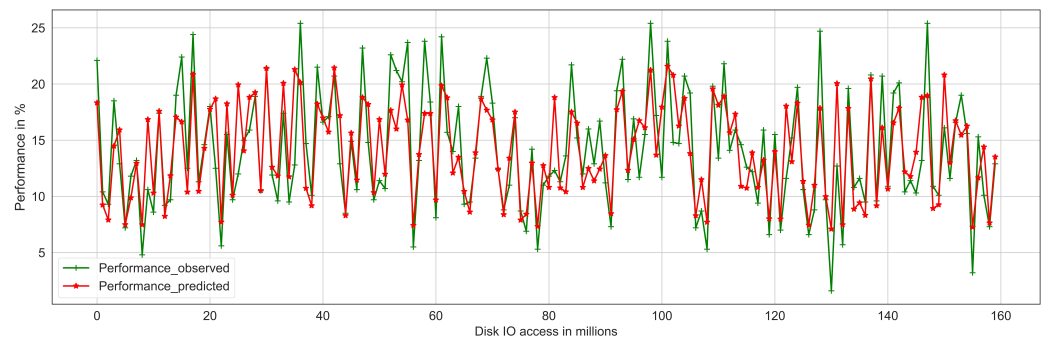


Figure 10. Predicted and observed values.

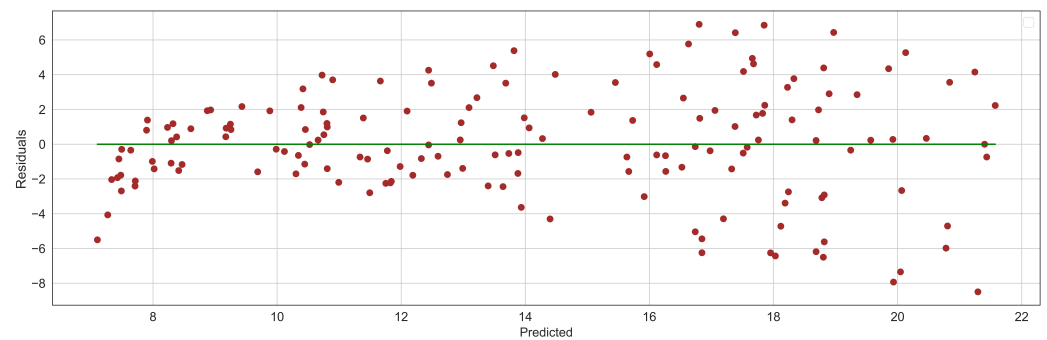


Figure 11. Residual plot.

If a VM is attached with a virtual HDD of 100 GB, and the HDFS block size is 128 MB, then the VM is capable of storing 400 ( $e$ ) blocks of different jobs. However, based on Equation (2), if the performance of the physical HDD attached with that VM ( $(IOP)_i^k$ ) is 85%, then  $(DB)_{i,j}^k$  gets  $\lfloor 0.85 * 400 \rfloor = 340$ . This means that storing 340 blocks in the respective VM will benefit while launching a batch of jobs. However, there is a chance of keeping the leftover storage unused. In this case, 15 GB is unused here. In another case, if  $(IOP)_i^k$  is 10%, then 90 of GB memory will be unused. As users pay for only the amount of storage used, cloud service providers can charge the user accordingly. In the worst-case scenario, 400 blocks of data are already stored in a VM. Unfortunately, disk performance falls relative to the other VMs in the same PM. In order to maintain scheduler performance afterward, the data blocks may be redistributed around the virtual cluster. Another scenario is that cloud service providers can migrate VMs to a PM that provides better HDD performance if a VM encounters prolonged HDD performance drops in a PM.

### 5.3. Scheduling Map Tasks Based on Heterogeneous Performance

The heterogeneous performance of VMs causes varying latency of the same task in different VMs. If a map task execution lasts for a long time, then the latency of the map phase is extended. As a batch of MapReduce jobs is executed periodically, data blocks could be shifted to a VM that performs better. Therefore, the number of map tasks completed per unit of time by a high-performing VM is higher than a VM with large capacity but varying performance. Therefore, VMs are dynamically monitored for varying performance, and data blocks are moved from one VM to another VM. We compared our approach with existing schedulers: Fair scheduler [8], RWS, and HMJS [5] based on the configuration mentioned in the Experimental setup. Figure 12 shows the number of non-local executions achieved with different schedulers for a batch of four different workloads. Prediction-based scheduler outperformed classical fair scheduler by 76% on average considering all four workloads. On average, our proposed approach minimizes the number of non-local executions by 72% compared to the RWS-based scheduler. Minimizing the number of non-local executions leads to a reduction in job latency, as shown in Figure 13. It is observed that

job latency is decreased up to 49% on average while using our proposed method over the classical fair scheduler. This is because the fair scheduler partitions the resources for all the jobs, so several local executions are very low, resulting in an increase in job latency (not the map task latency). Similarly, on average, a 40% improvement is observed while using the prediction-based approach compared to the RWS-based scheduler, because the RWS-based scheduler focuses on the computing capacity of VMs, while the prediction-based approach considers disk IO performance to minimize map phase latency.

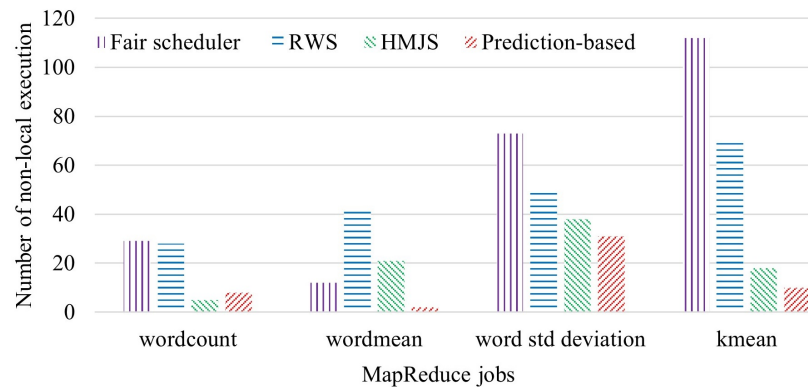


Figure 12. Number of non-local execution with different schedulers for different jobs.

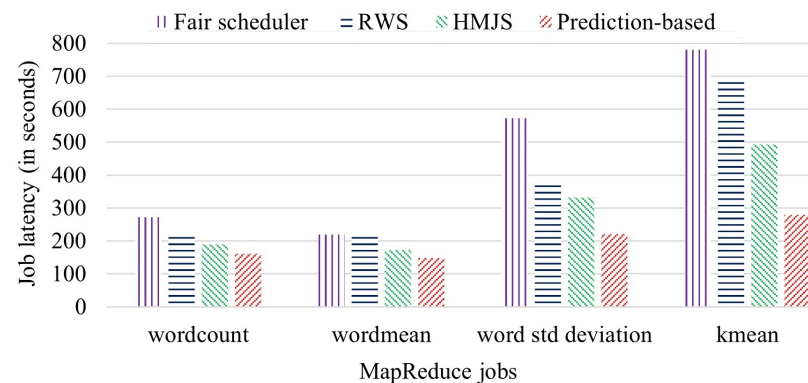
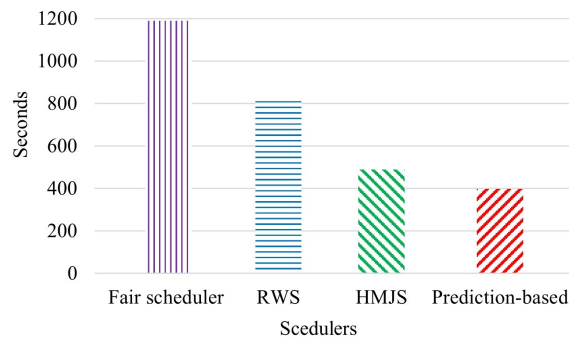


Figure 13. MapReduce job latency with different schedulers.

Although HMJS considers heterogeneous performance to minimize job latency, the prediction-based scheduler takes into account the number of disks available on each physical machine and its performance over time. Thus, the prediction-based scheduler only outperforms HMJS by 26%. Finally, to emphasize the effectiveness of our proposed method, we also compared the makespan of each scheduler for a batch of jobs. The prediction-based scheduler improved the makespan by up to 66% compared to the traditional fair scheduler, as shown in Figure 14. Similarly, it outperformed the RWS-based approach by 52%. It is significant to note that the improvement of our proposed approach is 19% in comparison to HMJS. HMJS uses a bin packing approach that randomly fits the most optimal combination of containers in each VM. In contrast, our proposed method takes a different approach to loading data blocks in advance to minimize map phase latency. When bin packing is focused, data local execution is compromised to obtain the most optimal combination of containers. Therefore, the number of non-local executions is steadily increasing while working with a bin backing-based approach in HMJS. In our proposed approach, we break this limitation by loading data blocks depending on the performance of the disk on physical machines where VMs have been hosted. Moreover, disk IO performance could be limited over time due to co-located VMs’ interference. Hence, we periodically observed the change in disk IO performance and moved data blocks accordingly depending on disk IO persistence. This significantly shortened the makespan when compared to HMJS. In

addition to considering HDD performance for placing data blocks, cloud service providers can monitor and balance workloads on PMs indirectly to manage HDD performance for data block placement by planning and distributing workloads.



**Figure 14.** Makespan for different schedulers.

#### 5.4. Rank Calculation for Launching Map Task

As discussed in Section 4.2, to schedule map tasks in all the MapReduce jobs, as displayed in Table 2, the performance score ( $MP$ ) of each VM in the virtual cluster is calculated based on disk IO ( $DIO$ ),  $CPU$ , and map task influence ( $MI$ ). Based on  $MP$ , a rank list is prepared to choose the right VM for launching map tasks by adhering to the data locality; rank list is then sorted in ascending order to choose the top 10% of VMs in the first priority. Ten VMs in the virtual cluster running in different PMs are considered for understanding the flow of heterogeneous performance calculation. The parameters in Table 2, such as  $VM_{ij}^{CPU}$ ,  $VM_{ij}^{DIO}$ ,  $VM_{ij}^{MI}$ ,  $VM_{ij}^{MP}$ , and rank display the values for all 10 different VMs. These values are calculated based on Equations (3)–(7). There are various scenarios of all these parameters that constitute rank 1 to 10. For instance, when  $VM_{ij}^{CPU}$ ,  $VM_{ij}^{DIO}$ , and  $VM_{ij}^{MI}$  hold values of 0.9, 0.95, and 0.1, respectively, the value for  $VM_{ij}^{MP}$  is high. Therefore, VM(10) is preferred for executing map tasks on the first priority. When more than one VM holds the same rank, based on features such as data locality, etc., appropriate VM is chosen for executing map tasks. In contrast, on the opposite side, when  $VM_{ij}^{CPU}$ ,  $VM_{ij}^{DIO}$ , and  $VM_{ij}^{MI}$  holds values of 0.65, 0.64, and 0.65, respectively, the value for  $VM_{ij}^{MP}$  is very low, which is 0.22. Therefore, VM(6) is least preferred for launching map tasks.

Initially, the top 10% of the high performing VMs are chosen. As per the example given in Table 2, only VM(10) is considered for launching map tasks. When a virtual cluster contains 100 s of VMs, then there could be many VMs that could come under the top 10%. After successfully launching map tasks on these VMs, based on rank in the table, the remaining tasks are executed. One important point to remember is that lower rank holder VMs are not having many map tasks for execution, as its performance is less significant. In contrast, the high performing VMs that hold top rank are considered for executing a higher number of map tasks. Hence, based on the heterogeneous performance of each VM, the respective number of map tasks can be executed. When this idea is combined with predictive-based data block placement, the performance of MapReduce can be improved.

**Table 2.** Rank calculation for different VMs.

$VM(j)$	$VM_{ij}^{CPU}$	$VM_{ij}^{DIO}$	$VM_{ij}^{MI}$	$VM_{ij}^{MP}$	Rank
1	0.8	0.68	0.5	0.34	7
2	0.9	0.91	0.2	0.73	2
3	0.7	0.55	0.6	0.22	9
4	0.85	0.82	0.4	0.49	4
5	0.75	0.73	0.45	0.40	6
6	0.65	0.64	0.65	0.22	10
7	0.95	0.86	0.25	0.65	3
8	0.8	0.77	0.4	0.46	5
9	0.7	0.59	0.55	0.27	8
10	0.9	0.95	0.1	0.86	1

## 6. Conclusions

Without cloud services, it is impossible for CPS applications to use big data processing frameworks, as it is expensive to set up on-premises. Hadoop is one of the most efficient processing tools for crunching large volumes of data. It is also offered as a cloud service on a cluster of VMs hosted on a cluster of physical servers in a CDC. In this case, heterogeneity in physical servers and VM performance is unavoidable. It adversely affects disk read performance for map tasks in MapReduce and the latency of map tasks. To handle these issues, we distribute the data blocks based on the performance of disk IO and dynamically exploit the heterogeneous performance of VMs to minimize job latency and makespan. We simulated our ideas on Hadoop 2.7 and compared them to the classical fair scheduler, RWS-based scheduler, and HMJS. Based on the results, our proposed scheduler outperformed existing schedulers for makespan by up to 66%, 52%, and 19%, respectively. The performance of our predictions can be affected by cloud scheduler decisions in the event of frequent virtual machine thrashing or dynamic resource sharing, although predictions play a vital role in our work. Therefore, considering the heuristics of the cloud resource scheduler in addition to the prediction for the placement of blocks could be an advantage.

**Author Contributions:** Conceptualization, G.S. and R.J.; methodology, G.S.; software, R.J.; validation, A.S., A.P. and G.S.; formal analysis, R.J.; investigation, A.P.; resources, A.S.; data curation, G.S.; writing—original draft preparation, G.S.; writing—review and editing, G.S.; visualization, R.J.; supervision, A.S. and A.P.; project administration, A.P.; funding acquisition, A.P. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the National Research Foundation of Korea (Grant No. 2020R1A2C1012196), and in part by the School of Computer Science and Engineering, Ministry of Education, Kyungpook National University, South Korea, through the BK21 Four Project, AI-Driven Convergence Software Education Research Program, under Grant 4199990214394.

**Data Availability Statement:** Simulation of this research is carried out by a program developed by our team. As a result, the data generated by the program is not available for online publishing separately.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

- Rathore, M.M.U.; Shah, S.A.; Awad, A.; Shukla, D.; Vimal, S.; Paul, A. A cyber-physical system and graph-based approach for transportation management in smart cities. *Sustainability* **2021**, *13*, 7606. [[CrossRef](#)]
- Jeyaraj, R.; Balasubramaniam, A.; Kumara, A.M.A.; Guizani, N.; Paul, A. Resource Management in Cloud and Cloud-Influenced Technologies for Internet of Things Applications. *ACM Comput. Surv.* **2022**, *55*, 1–35. [[CrossRef](#)]
- Jeffrey Dean, S.G. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* **2008**, *51*, 2140–2144.

4. Guo, Y.; Rao, J.; Jiang, C. Moving Hadoop into the Cloud with Flexible Slot Management and Speculative Execution. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *28*, 798–812. [CrossRef]
5. Jeyaraj, R.; Ananthanarayana, V.S.; Paul, A. Improving MapReduce scheduler for heterogeneous workloads in a heterogeneous environment. *Concurr. Comput. Pract. Exp.* **2020**, *32*. [CrossRef]
6. Jeyaraj, R.; Ananthanarayana, V.S.; Paul, A. Dynamic ranking-based MapReduce job scheduler to exploit heterogeneous performance in a virtualized environment. *J. Supercomput.* **2019**, *75*, 7520–7549.
7. Xiong, R.; Du, Y.; Jin, J.; Luo, J. HaDaap: A hotness-aware data placement strategy for improving storage efficiency in heterogeneous Hadoop clusters. *Concurr. Comput. Pract. Exp.* **2018**, *30*. [CrossRef]
8. Hadoop MapReduce Fair Scheduler. Available online: <https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/FairScheduler.html> (accessed on 20 May 2023).
9. Hadoop MapReduce Capacity Scheduler. Available online: <https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html> (accessed on 20 May 2023).
10. Hashem, I.A.T.; Anuar, N.B.; Marjani, M.; Ahmed, E.; Chiroma, H.; Firdaus, A.; Abdullah, M.T.; Alotaibi, F.; Mahmoud Ali, W.K.; Yaqoob, I.; et al. MapReduce scheduling algorithms: A review. *J. Supercomput.* **2020**, *76*, 4915–4945. [CrossRef]
11. Ghemawat, S.; Gobioff, H.; Leung, S.T. The Google file system. *ACM USA* **2003**, *37*, 29–43.
12. Hadoop Distributed File System (HDFS). Available online: [https://hadoop.apache.org/docs/r1.2.1/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html) (accessed on 20 May 2023).
13. Song, J.; He, H.Y.; Wang, Z.; Yu, G.; Pierson, J.M. Modulo Based Data Placement Algorithm for Energy Consumption Optimization of MapReduce System. *J. Grid Comput.* **2018**, *16*, 409–424. [CrossRef]
14. Derouiche, R.; Brahmi, Z. A cooperative agents-based workflow-level distributed data placement strategy for scientific cloud workflows. In Proceedings of the 2nd International Conference on Digital Tools & Uses Congress, Virtual, 15–17 October 2020.
15. Li, C.; Liu, J.; Li, W.; Luo, Y. Adaptive priority-based data placement and multi-task scheduling in geo-distributed cloud systems. *Knowl.-Based Syst.* **2021**, *224*, 107050. [CrossRef]
16. Du, Y.; Xiong, R.; Jin, J.; Luo, J. A Cost-Efficient Data Placement Algorithm with High Reliability in Hadoop. In Proceedings of the Fifth International Conference on Advanced Cloud and Big Data (CBD), Shanghai, China, 13–16 August 2017; pp. 100–105.
17. Shakarami, A.; Ghobaei-Arani, M.; Shahidinejad, A.; Masdari, M.; Shakarami, H. Data replication schemes in cloud computing: A survey. *Clust. Comput.* **2021**, *24*, 2545–2579. [CrossRef]
18. Sabaghian, K.; Khamforoosh, K.; Ghaderzadeh, A. Data Replication and Placement Strategies in Distributed Systems: A State of the Art Survey. *Wirel. Pers. Commun.* **2023**, *129*, 2419–2453. [CrossRef]
19. Wang, T.; Wang, J.; Nguyen, S.N.; Yang, Z.; Mi, N.; Sheng, B. EA2S2: An efficient application-aware storage system for big data processing in heterogeneous clusters. In Proceedings of the 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017.
20. Bouhouch, L.; Zbakh, M.; Tadonki, C. Dynamic data replication and placement strategy in geographically distributed data centers. *Concurr. Comput. Pract.* **2022**, *35*. [CrossRef]
21. Ahmadi, A.; Daliri, M.; Goharshady, A.K.; Pavlogiannis, A. Efficient approximations for cache-conscious data placement. In Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, 13–17 June 2022; pp. 857–871.
22. Xu, H.; Liu, W.; Shu, G.; Li, J. LDBAS: Location-aware data block allocation strategy for HDFS-based applications in the cloud. *KSII Trans. Internet Inf. Syst.* **2018**, *12*, 204–226.
23. Gandomi, A.; Reshadi, M.; Movaghar, A.; Khademzadeh, A. HybSMRP: A hybrid scheduling algorithm in Hadoop MapReduce framework. *J. Big Data* **2019**, *6*, 106. [CrossRef]
24. Jin, J.; An, Q.; Zhou, W.; Tang, J.; Xiong, R. DynDL: Scheduling data-locality-aware tasks with dynamic data transfer cost for multicore-server-based big data clusters. *Appl. Sci.* **2018**, *8*, 2216. [CrossRef]
25. Qureshi, N.M.F.; Siddiqui, I.F.; Unar, M.A.; Uqaili, M.A.; Nam, C.S.; Shin, D.R.; Kim, J.; Bashir, A.K.; Abbas, A. An Aggregate MapReduce Data Block Placement Strategy for Wireless IoT Edge Nodes in Smart Grid. *Wirel. Pers. Commun.* **2019**, *106*, 2225–2236. [CrossRef]
26. Sellami, M.; Mezni, H.; Hacid, M.S.; Gammoudi, M.M. Clustering-based data placement in cloud computing: A predictive approach. *Clust. Comput.* **2021**, *24*, 3311–3336. [CrossRef]
27. He, Q.; Zhang, F.; Bian, G.; Zhang, W.; Li, Z.; Yu, Z.; Feng, H. File block multi-replica management technology in cloud storage. *Clust. Comput.* **2023**. [CrossRef]
28. Malik, M.; Neshatpour, K.; Rafatirad, S.; Homayoun, H. Hadoop workloads characterization for performance and energy efficiency optimizations on microservers. *IEEE Trans. Multi-Scale Comput. Syst.* **2018**, *4*, 355–368. [CrossRef]
29. Yu, Z.; Xiong, W.; Eeckhout, L.; Bei, Z.; Mendelson, A.; Xu, C. MIA: Metric importance analysis for big data workload characterization. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 1371–1384. [CrossRef]
30. Anjos, J.C.S.; Carrera, I.; Kolberg, W.; Tibola, A.L.; Arantes, L.B.; Geyer, C.R. MRA++: Scheduling and data placement on MapReduce for heterogeneous environments. *Future Gener. Comput. Syst.* **2015**, *42*, 22–35. [CrossRef]
31. Ubarhande, V.; Popescu, A.M.; González-Vélez, H. Novel Data-Distribution Technique for Hadoop in Heterogeneous Cloud Environments. In Proceedings of the Ninth International Conference on Complex, Intelligent, and Software Intensive Systems, Santa Catarina, Brazil, 8–10 July 2015; pp. 217–224.



32. Chen, W.; Paik, I.; Li, Z. Tology-Aware Optimal Data Placement Algorithm for Network Traffic Optimization. *IEEE Trans. Comput.* **2016**, *65*, 2603–2617. [[CrossRef](#)]
33. PUMA. Purdue University. Available online: <https://engineering.purdue.edu/~puma/datasets.htm> (accessed on 20 May 2023).

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.