# Accepted Manuscript

A data locality based scheduler to enhance MapReduce performance in heterogeneous environments

Nenavath Srinivas Naik, Atul Negi, Tapas Bapu B.R., R. Anitha

PII: S0167-739X(18)30837-9
DOI: https://doi.org/10.1016/j.future.2018.07.043
Reference: FUTURE 4361

To appear in: *Future Generation Computer Systems*

Received date : 6 April 2018
Revised date : 1 July 2018
Accepted date : 18 July 2018

Please cite this article as: N.S. Naik, A. Negi, T.B.B.R. T.B.B.R, R. Anitha, A data locality based scheduler to enhance MapReduce performance in heterogeneous environments, *Future Generation Computer Systems* (2018), https://doi.org/10.1016/j.future.2018.07.043

# A Data Locality based Scheduler to Enhance MapReduce Performance in Heterogeneous Environments

Nenavath Srinivas Naik[a,*], Atul Negi[a], Tapas Bapu B R[b], R. Anitha[b]

[a]*School of Computer and Information Sciences, University of Hyderabad, Hyderabad 500046, India.*
[b]*S.A.Engineering College, Chennai, India.*

**Abstract**

MapReduce is an essential framework for distributed storage and parallel processing for large-scale data-intensive jobs proposed in recent times. Hadoop default scheduler assumes homogeneous environment. This assumption of homogeneity does not work at all times in practice and limits the performance of MapReduce. Data locality is essentially moving computation closer (faster access) to the input data. Fundamentally, MapReduce does not always look into the heterogeneity from a data locality perspective. Improving data locality for MapReduce framework is an important issue to improve the performance of large-scale Hadoop clusters.

This paper proposes a novel data locality based scheduler which allocates input data blocks to the nodes based on their processing capacity. Also schedules *map* and *reduce* tasks to the nodes based on their computing ability in the heterogeneous Hadoop cluster. We evaluate proposed scheduler using different workloads from Hi-Bench benchmark suite. The experimental results prove that our proposed scheduler enhances the MapReduce performance in heterogeneous environments. Minimizes job execution time, and also improves data locality for different parameters as compared to the Hadoop default scheduler, Matchmaking scheduler and Delay scheduler respectively.

*Keywords:* MapReduce, Data Locality, Task Scheduler, Heterogeneous Environments

*Corresponding author
*Email address:* srinuphdcs@gmail.com (Nenavath Srinivas Naik)

## 1. Introduction

Distributed and Parallel processing is one of the best intelligent ways to store and compute Big Data [2]. An enormous quantity of data is produced every day from web authoring, digital media, and scientific instruments, etc. One of the significant challenges to the software and research community is efficiently storing, querying, analyzing, forecasting and using of this Big Data. MapReduce was first developed at Google by Jeffrey Dean and Sanjay Ghemawat [1]. MapReduce is one of the significant computational frameworks for large-scale data processing and analysis on massive clusters of commodity machines. Apache Hadoop is now the open-source implementation of the MapReduce framework that was created by Doug Cutting [5].

MapReduce follows master-slave architecture [3]. A single master node monitors the status of the slave nodes in a cluster. The master node divides the input files into multiple *map* tasks, and then schedules both *map* and *reduce* tasks to slave nodes in a cluster. A task assigned to slave nodes has two phases of processing: *Map* and *Reduce*. From the *Map* phase, intermediate (*key*, *value*) pairs are generated and transferred to the *Reduce* phase as input. The *Reduce* phase sorts the intermediate results by matched *key* and then merges them as one final output.

The homogeneity assumption is that all the nodes in the cluster will have the same processing capacity. This assumption can, as a matter of fact, degrade the performance of MapReduce framework because there is certain diversity in the hardware. In the current day scenario, financially constrained entities like Universities and Colleges would like to have a cluster with a mix of legacy hardware with newer ones. Advancement of hardware technology is another practical reason for heterogeneous clusters to increase, as hardware sourced at different times in technology cycles can be brought together in a better way. Therefore working with heterogeneous clusters would be a major goal to increase the scope of MapReduce [31]. Additionally, issues of data locality, straggler tasks, also are known to affect the MapReduce performance [5]. These problems have been underestimated by researchers in most of the conventional MapReduce scheduling algorithms.

The aim is to design an efficient scheduler, which is responsible for making decisions on which task to be executed at what time and on which machine. The most common objective of scheduling is to minimize the job completion

time by appropriately allocating the tasks to the processors. An improper scheduling of jobs or tasks will not utilize the actual potential of the systems present in the cluster. One of Hadoop's primary principles is "moving computation towards data is cheaper than moving data towards computation" [12]. Cross-switch network traffic is one of the obstructions in data-intensive computing which can be reduced by proper data locality, which in turn is characterized as the distance between the input data node and task-allotted node [13]. The data locality issue has got much consideration from the research community, and schedulers which enhance data locality have been proposed in the extant literature and also implemented practically [9].

In the MapReduce model, the job completion time in each node depends on the complete execution of workload assigned to that node [28]. To balance the workload in a cluster, Hadoop distributes the data blocks to different nodes by taking into account the availability of disk space. Such a data distribution methodology might be proficient for a homogeneous environment, where nodes are identical [30]. If workload distributes evenly in the heterogeneous nodes, then different processing capacity of nodes will complete their execution at different times. Therefore, to minimize the job execution time we have to distribute the workload among nodes depending on the processing capacity of nodes in the cluster.

Improving data locality for MapReduce is favorable in following ways:

1. Minimizes job execution time as data transfer time limits are the ones which are going to occupy most of the total job execution time.
2. Reduces the cumulative data center network traffic since fewer *map* and *reduce* tasks need to fetch data tenuously.

Data distribution might not be effective in a homogeneous environment because all the nodes in a cluster will have the same hardware configurations [25]. However, in a heterogeneous environment, nodes with high computation can complete computing local data quicker than nodes with low computation [21]. For this situation, the overhead of moving unprocessed data from slow processing node to fast processing node is high if the volume of data transfer is high. These issues motivate us to develop dynamic data locality based scheduler which minimizes the quantity of data movements among slow and fast processing nodes and thereby enhances the MapReduce performance in heterogeneous environments.

The key contributions of this research paper are therefore:

1. Proposal of a Data distribution method that dynamically distributes input data to the nodes depending on their processing capacity in the cluster.
2. Development of a data locality based scheduler that schedules *map* and *reduce* tasks to different nodes in a heterogeneous cluster by their processing capacity. (i.e. Nodes with fast processing capacity will be assigned more tasks than slower ones).
3. Comparison of our proposed scheduling approach with the state-of-the-art schedulers using heterogeneous workloads from Hi-Bench benchmark suite in heterogeneous Hadoop cluster.
4. Proposal of scheduling approach for reducing data movement activities in a cluster by improving data locality rate and also minimizing the job execution time. Thus, the proposed approach enhances MapReduce performance in heterogeneous environments.

The rest of this paper is structured as follows. Section 2 illustrates the background of the MapReduce framework. Section 3 presents the related work. Section 4 introduces proposed novel scheduling approach for data locality in MapReduce framework. Section 5 analyses the performance of the proposed scheduling approach. Finally, we conclude the paper by providing an outline for future research work in Section 6.

## 2. Background

This section illustrates a brief overview of the MapReduce framework and concisely address about heterogeneous environments.

### *2.1. Overview of MapReduce*

The primary objective of the MapReduce framework is to distribute and parallelize job execution on several nodes in a cluster for processing [10]. A MapReduce application which needs to be executed is named to be a job. The input file for a job will be residing on a distributed file system until the cluster gets divided into equal-sized blocks [27]. A job can be split into a series of tasks. Every data block is first processed by a *Map* function, which produces an output as intermediate $(key, value)$ pairs and then a *Reduce* function produces the final output.

Hadoop framework contains two components [2]: 1. HDFS (Hadoop Distributed File System), which stores the input data and 2. MapReduce engine,

which processes the data blocks stored in different nodes of a cluster. HDFS contains a *NameNode* and *DataNodes* in a cluster. *NameNode* is a master node which contains the meta-data information of the data block locations in a cluster. *DataNodes* are slave nodes which store the data blocks within a cluster. MapReduce contains a *JobTracker* and multiple *TaskTrackers*. *JobTracker* deals with job scheduling and tasks to available *TaskTrackers* within a cluster depending on the slot availability. *TaskTrackers* process the *map* and *reduce* tasks on the corresponding nodes in the cluster. For clarity, we have modified the MapReduce workflow and shown it in the Fig. 1. The MapReduce contains the following stages [5] when scheduling a job from the master node to the slave nodes.

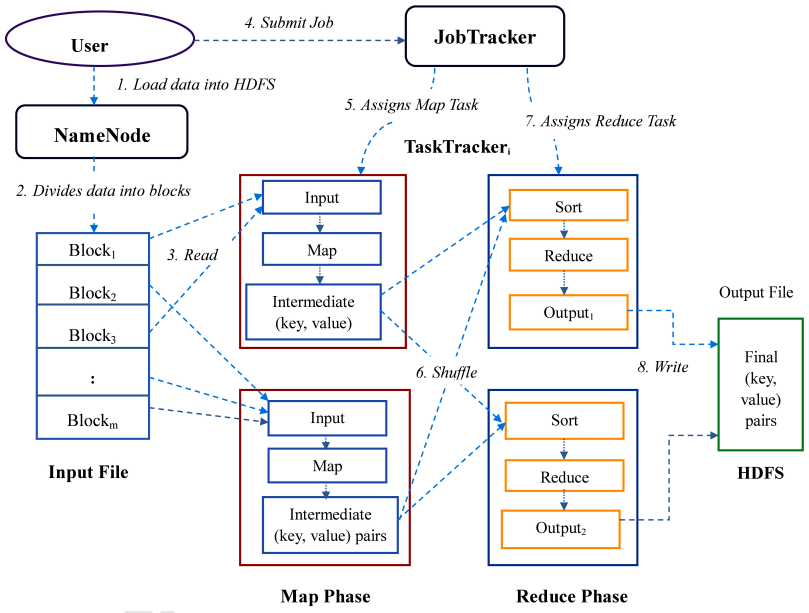


Figure 1: MapReduce workflow [5]

1. User submits input data to the *NameNode*
2. The *NameNode* divides the data into $m$ blocks of the same size. $r$ duplicates of every block are produced for fault tolerance. ($r$ is the replication factor).
3. The master node picks up the idle slave nodes to assigns *map* tasks and *reduce* tasks.

4. User submits a job to the *JobTracker* for processing corresponding data blocks.
5. A slave node that is executing a *map* task parses the data block and assigns each (*key*, *value*) pair to the *Map* function. The intermediate (*key*, *value*) pairs are buffered in memory at corresponding nodes.
6. The buffered (*key*, *value*) pairs are written to data residing nodes at fixed intervals and divided into $R$ sections by means of a (configurable) partition function (default is $hash(intermediate\text{key}) mod R$). The identical (*key*, *value*) pairs go to same partition. When the *map* task completes, the slave node sends the location information of partition to the master node.
7. The node which contains *Reducer* function reads the data using remote procedure calls. It sorts and groups the data by *intermediatekey* so that all values of the same *key* are grouped. It is called shuffling of the task.
8. The master node produces the final output after execution of all *map* and *reduce* tasks.

### *2.2. Heterogeneous Environments*

Hadoop was initially aimed for homogeneous cluster environments, but, now it is commonly used in various heterogeneous environments [11]. Heterogeneity is categorized as below which is increasing in both workloads and cluster configuration.

1. In Heterogeneous environments, each node in the cluster would have different physical parameters such as data storage and processing units.
2. MapReduce jobs can be heterogeneous on various task features such as data and computation requirements.

However, current Hadoop schedulers are not correctly adapted for heterogeneous systems [26]. This research is originally motivated by addressing the scheduling challenges arising due to increase in heterogeneity of distributed systems. The heterogeneous system introduces new scheduling challenges, which directly affect the system performance.

## 3. Related Work

A considerable amount of research work has been conducted to expand the data locality approach for better effectiveness. We, thereby, briefly dis-

cuss Data Locality based MapReduce Schedulers such as MapReduce default scheduler, Matchmaking scheduler, and state-of-the-art Delay scheduler. These schedulers have been developed to enhance data locality of the MapReduce framework in heterogeneous environments, but they do have certain limitations.

### *3.1. Hadoop default Scheduler*

The Hadoop default FIFO (i.e. first-in-first-out) [2] scheduler has already considered data locality as its main functionality. This scheduler will always pick the first job and schedule its *map* tasks that are local to the data. In cases when a job may not have any *map* task neighborhood to a data node, stand out of its non-nearby *map* undertakings will be allocated to the node at once generally assigned on the same rack, or onto another rack in the cluster.

#### *3.1.1. Limitations of Hadoop default Scheduler*

1. Default scheduler takes after the strict FIFO job order for task assignment. This scheduling rule hurts the data locality because a different job's local tasks cannot be allotted to the slave node until the primary job schedules all its *map* tasks.
2. Default scheduler does not work better in Heterogeneous environments.
3. Response time and data locality rate are minimal for smaller jobs as compared to larger jobs.

### *3.2. Matchmaking Scheduler*

Matchmaking Scheduler [7] tries to enhance data locality for *map* tasks. The main thought of this algorithm is to provide all nodes a fair and plausible chance to take hold of local tasks in advance, so that some non-local tasks are allocated to any other nodes. If none of the local tasks were found for a node then, it would not have a heartbeat signal received to that particular interval. During one heartbeat interval, all other nodes that have free *Map* slots will probably send their heartbeat signals to the master node, and possibly receive local task details. For example, suppose, if a local task cannot be found for a node during the second time in a row, only then a non-local task is assigned. The nodes use a locality marker that marks their status immediately while processing. If a local task cannot be assigned to a node, then depending on the value of the node's locality marker, the node will either be in a wait state for a heartbeat interval or receive a non-local task to continue.

*3.2.1. Limitations of Matchmaking Scheduler*

1. One of the problems of the Matchmaking algorithm is the fact that it does not consider rack locality into account.
2. The fact that it does not need to configure any parameters but it might make the algorithm rigid because even if it can be a hassle to tune a parameter properly, a proper configuration might lead to better results.

*3.3. Delay Scheduler*

Zaharia et al. [6] have proposed the delay scheduling algorithm in Fair Scheduler to enhance data locality in the MapReduce framework. Delay scheduler halts several job request when allocating *map* tasks to a $TaskTracker$. If the first job does not have a local *map* task, the scheduler can delay it to wait and allocate a different job's local *map* tasks. Here, the delay time $D$ is an essential factor. As a matter of course, it is fixed at 1.5 times heartbeat interval of the slave node. When a job is postponed for more than $D$ time units, then the scheduler can allocate the job's non-local *map* tasks. To get a better execution of the delay scheduler, we need to pick an appropriate $D$ value. If the value is too large, then there's a risk of job starvation, which in turn can influence the performance overall. Despite, the small $D$ value cannot guarantee the data locality. Delay scheduler solves other locality issues, for example, head-of-line scheduling. It happens when small jobs are improbably allocated to a node without the processing of data associated with it.

*3.3.1. Limitations of Delay Scheduler*

1. Despite the fact that this strategy can enhance data locality, yet it needs to give up some utilization while waiting. It has to wait for more time to allocate a local task or discard the locality once losing many CPU cycles.
2. This algorithm is optimized for small tasks like those that are usually encountered for example in Facebook clusters; so it is not appropriate for jobs that take more execution time. It can bring job execution degradation if it has delayed tasks.
3. Wait times used in delay scheduling must be manually configured for every particular system that uses this scheduling method.
4. Another limitation of this algorithm is that it does not consider data locality for *reduce* tasks.

To overcome some of the limitations of the above Hadoop schedulers, we have developed our scheduling approach for data locality under heterogeneous environments.

## 4. Proposed Data Locality based MapReduce Scheduler in Heterogeneous Environments

In this section, we propose a scheduling approach that solves the problem of data distribution and task scheduling in MapReduce framework depending on the node processing capacity, so as to improve the MapReduce performance in heterogeneous environments.

In a heterogeneous environment, the processing capabilities of nodes may differ altogether. A node with high processing capacity can complete execution of local data quicker than a node with slow processing capacity. A way to enhance the MapReduce performance in heterogeneous computing environments is to minimize the volume of data transfer between fast and slow processing nodes. To manage the data load in a heterogeneous environment, we propose a dynamic data distribution approach. This approach aims to divide the data dynamically depending upon the processing capacity of the nodes in the cluster.

### *4.1. Proposed Map Task Data Locality based Scheduler*

The proposed scheduling algorithm first dynamically divides the input data into various data blocks depending on the node processing capacity in the cluster by the dynamic block partitioner. Then, the proposed algorithm assigns these data blocks to different nodes in the cluster. A node with high processing capacity will get more data blocks compared to the low processing node.

We calculate the total number of data blocks for the input data using (1).

$$TDB \leftarrow DS/BS \tag{1}$$

Here, $TDB$ is the total number of data blocks for input data, $DS$ is Data size of input data, and $BS$ is Default block size.

We estimate the processing capacity of each $Node\ i$ in the heterogeneous Hadoop cluster using (2).

$$NPC(i) \leftarrow PDN(i) + ATE(i) \tag{2}$$

Here, $NPC(i)$ is the Node processing capacity of $i^{th}$ node, $PDN(i)$ is Performance of $i^{th}$ node, and $ATE(i)$ is Average task execution time in $i^{th}$ node.

We will find the node processing capacity (NPC) of all nodes in the cluster by considering the following: 1. Average task execution time of a particular job in that node of the cluster and 2. We combine the available memory (RAM) and CPU of that node at regular intervals.

We will find the average task execution (ATE) time by executing a few tasks of a particular job on that node. We have added $ATE$ to the $NPC$ because the job will execute at different times for heterogeneous nodes in the cluster and this helps in identifying the $NPC$ of a node. We can determine the percentage of Memory and CPU usage in every node of the heterogeneous cluster. After that, we will calculate the free Memory and CPU usage in every node in the cluster using (3) and (4).

$$FreeCPU(i) \leftarrow ((100 - UsedCPU(i)) \times CPU(i))/100 \tag{3}$$

$$FreeMem(i) \leftarrow ((100 - UsedMem(i)) \times Mem(i))/100 \tag{4}$$

Here, $CPU(i)$ is Processing capacity of $i^{th}$ node in the cluster, $Mem(i)$ is RAM capacity of $i^{th}$ node in the cluster, $UsedCPU(i)$ is Percentage of CPU used in $i^{th}$ node of the cluster, and $UsedMem(i)$ is Percentage of Memory used in $i^{th}$ node of the cluster.

We will find the performance of $i^{th}$ data node by obtaining free available CPU and memory from that node as in (5)

$$PDN(i) \leftarrow FreeCPU(i) + FreeMem(i) \tag{5}$$

To identify the fast and slow processing nodes accurately, we use $K$-means clustering algorithm to divide the nodes into $K$ clusters by considering dynamic features such as:

1. Average task execution time
2. The amount of free Memory available
3. The amount of free CPU available
4. Disk space left on each node

Each time when the job runs, $K$-means algorithm classifies the nodes in the cluster depending on the node processing capacity.

We calculate the total node processing capacity of all heterogeneous nodes present in the cluster using (6).

$$TNPC \leftarrow \sum_{i=1}^{n} NPC(i) \tag{6}$$

Here, $TNPC$ is Total node processing capacity of the heterogeneous cluster, and $n$ is the number of data nodes in the cluster.

We estimate the number of data blocks to be assigned for each $DataNode$ $i$ in the cluster as in (7).

$$NB(i) \leftarrow TDB \times (NPC(i)/TNPC) \tag{7}$$

Here, $NB(i)$ is Number of data blocks to be assigned for $i^{th}$ $DataNode$.

We will assign the number of data blocks to $i^{th}$ $DataNode$ in a heterogeneous environment depending on the processing capacity of the nodes as in (8)

$$DN(i) \leftarrow NB(i) \tag{8}$$

Here, $DN(i)$ is $i^{th}$ data node in the heterogeneous cluster.

Let us consider a MapReduce job with its input data in a cluster. The proposed scheduling approach will dynamically divide the input data and distribute the data blocks to the nodes according to the processing capacity of nodes in the cluster, so that all the nodes in the cluster will process their local data blocks within the same amount of time.

We will arrange the data nodes according to their node processing capacity in a heterogeneous Hadoop cluster.

We will first check if there are any $Map$ slots available on the fast processing node. The fast processing data node has the maximum processing capacity within the Hadoop cluster. If $Map$ slots are available, then we will assign the $map$ tasks ($MT$) to the fast processing node in the cluster as in (9).

$$FPN(i) \leftarrow MT \tag{9}$$

Here, $FPN(i)$ is $i^{th}$ fast processing data node in the heterogeneous cluster.

Suppose, if $Map$ slots are not available on that node, then wait until it satisfies the node delay threshold as in (10). After the time period, assign

*map* task to the subsequent fast processing node in the cluster. We have taken the $NDT$ (Node Delay Threshold) as 4.5 seconds because if the $NDT$ value is larger than that, then job starvations may occur affecting, in turn, the MapReduce performance. On the other hand, if $NDT$ value is smaller, then it cannot guarantee the data locality.

$$WaitNode \leqslant NDT(i) \tag{10}$$

The execution flow of our proposed scheduler contains the following stages as shown in Fig. 2.

1. User submits the input data to the $NameNode$. The Dynamic block partitioner present in the $NameNode$, divides the input data according to the processing capacity of nodes in the cluster. Dynamic block partitioner gets $NPC$ of nodes from meta-data information and this information is updated periodically.
2. The Dynamic block partitioner then assigns the input data blocks to the corresponding data nodes in the cluster depending on their processing capacity.
3. User submits a job to the $JobTracker$, which in turn selects a job from the job running queue.
4. Further, the $JobTracker$ will gets the information from the meta-data and then assigns tasks ($T$) to the $TaskTrackers$ depending on the number of slots ($Map$ or $Reduce$) available on the fast processing node in a heterogeneous Hadoop cluster.

We consider the following cases of our proposed scheduler for Fig 2.

*Case 1:.* if we want to assign any new task, then we check for the slot's availability of fast processing node. In this case, $Node_3$ has the maximum node processing capacity of 8 units, and 1 $Map$ slot is available among 4 $Map$ slots (since 3 *map* tasks are running on 3 $Map$ slots). Therefore, we can assign any new task ($T$) of $Job_1$ or $Job_2$ to $Node_3$.

*Case 2:.* the $Node_2$ has 2 $Map$ slots, and all are executing the tasks. Suppose any new task of $Job_2$ or $Job_1$ wants to run on $Node_2$ it has to wait for a threshold period so that the slots can be free. Suppose, if slots are available, then we give priority to $Job_2$ and assign its tasks instead of $Job_1$ because $Node_2$ has data blocks of $Job_2$.

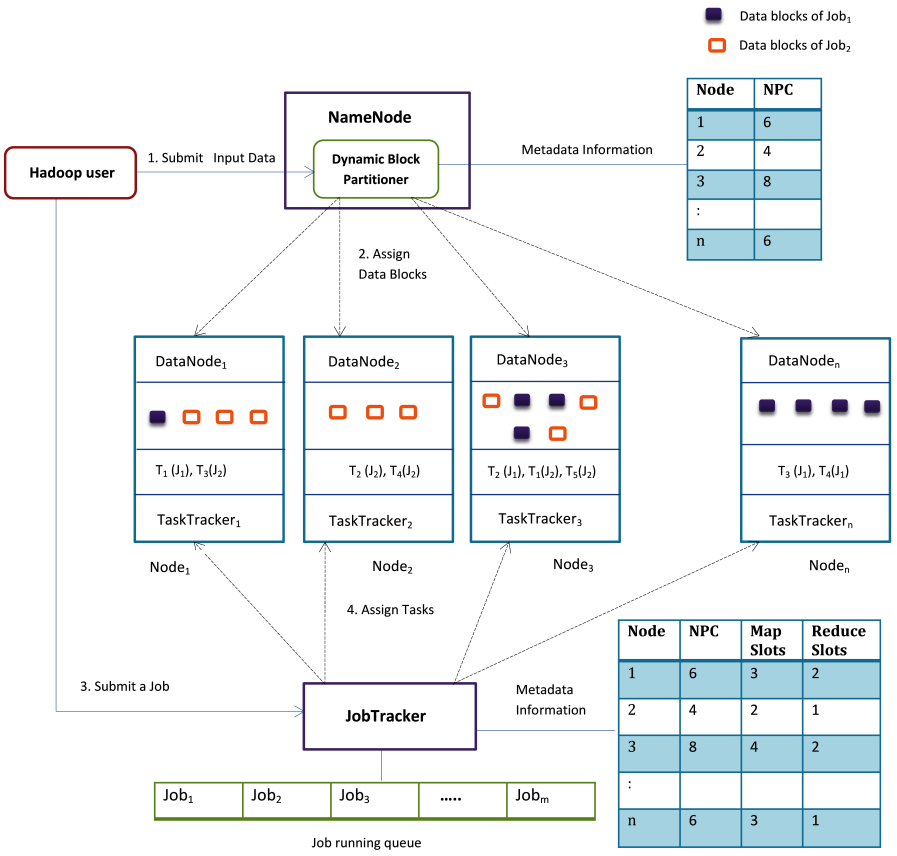


Figure 2: Execution flow of Proposed Scheduler

*Case 3:*. considering $Node_1$ has 3 $Map$ slots, if any task of $Job_1$ wants to run on $Node_1$ it has to wait because it has no corresponding data block. If any task of $Job_2$ intends to run on $Node_1$, then we can assign a task to it because it has one free $Map$ slot and also a corresponding data block.

We present a scheduler which dynamically distributes data blocks and schedules *map* tasks to a node which has maximum processing capacity in the heterogeneous cluster as shown in algorithm 1.

**Algorithm 1** Proposed Algorithm for Map Task Data Locality

**Input:** Set of Unscheduled $MapTask$s (UMT).

**Output:** A $MapTask$ $MT \in UMT$ that can be scheduled on a fast processing node.

1: **for** each $DataNode$ $i$ in the heterogeneous cluster **do**
2: Calculate the total no. of Data blocks for input data
3: $TDB \leftarrow DS/BS$
4: Calculate the Node processing capacity of $i$
5: $NPC(i) \leftarrow PDN(i) + ATE(i)$
6: Calculate the Total Node processing capacity of the *cluster*
7: $TNPC \leftarrow \sum_{i=1}^{n} NPC(i)$
8: Calculate the no. of Data blocks for $i$ in the cluster
9: $NB(i) \leftarrow TDB \times (NPC(i)/TNPC)$
10: Assign the no. of Data blocks to $i$ in the cluster
11: $DN(i) \leftarrow NB(i)$
12: Run $K - means$ algorithm to classify the nodes into $K$ clusters depending on their processing capacity
13: Arrange the Fast Processing Nodes (FPN) in the cluster in order
14: **if** $Map$ slots available on $FPN$ **then**
15: Assign $map$ tasks to $FPN$
16: $FPN(i) \leftarrow MT$
17: **else**
18: $WaitNode \leqslant NodeDelayThreshold$
19: $FPN(i) \leftarrow MT$
20: **end if**
21: **end for**

We will make few assumptions such as:

1. Each node in the cluster is given a value $N$ which represents the relative processing capacity of that particular node.
2. While assigning a value $N$, we will take the following parameters such as average task execution time, free Memory and CPU usage of that

node in the cluster.

For example, if $Node_1$ has $N$ value of 10 and another $Node_2$ has $N$ value of 5, it signifies that $Node_1$ is capable to process data two times faster than $Node_2$. Data distribution mechanism in Hadoop might not be effective in a homogeneous environment, where every node has the same processing and disk capacity. Suppose, if we have homogeneous Hadoop cluster environment, then the data block assignment to the nodes will be as shown in Table 1.

Table 1: Default Data block assignment in a Homogeneous environment

| Rack | Node | NPC | Number of Data Blocks assigned |
|---|---|---|---|
| 1 | 1 | 10 | 10 |
| | 2 | 10 | 10 |
| | 3 | 10 | 10 |
| | 4 | 10 | 10 |
| | 5 | 10 | 10 |
| 2 | 6 | 10 | 10 |
| | 7 | 10 | 10 |
| | 8 | 10 | 10 |
| | 9 | 10 | 10 |
| | 10 | 10 | 10 |

For better understanding of the algorithm 1, we consider an example for data allocation. We have an input data of size 12.8 GB and block of size 128 MB. The total number of data blocks for that input data is 100 blocks ($12800MB/128MB = 100$), which is to be distributed among the heterogeneous nodes of the cluster. Suppose, if we assume that node processing capacity of $Node_1$ is 6 units and the total node processing capacity of all the nodes in the cluster is 50 units, then the number of data blocks that is to be assigned to $Node_1$ is 12 Blocks ($100 \times 6/50 = 12$). As $Node_4$ and $Node_8$ are fast processing data nodes, as their node processing capacity is 8 units. Thus the number of data blocks assigned is 16 blocks each ($100 \times 8/50 = 16$). $Node_3$ and $Node_{10}$ are the slow processing nodes, as their node processing capacity is 2 units compared to other nodes in the cluster and assigned 4 data blocks each ($100 \times 2/50 = 4$). Accordingly, the number of data blocks is allocated to each node in the cluster depending on their processing capacity as shown in Table 2.

Table 2: Data block assignment of Proposed scheduler in a Heterogeneous environment

| Rack | Node | NPC | Number of Data Blocks assigned |
|---|---|---|---|
| 1 | 1 | 6 | 12 |
| | 2 | 4 | 8 |
| | 3 | 2 | 4 |
| | 4 | 8 | 16 |
| | 5 | 4 | 8 |
| 2 | 6 | 6 | 12 |
| | 7 | 4 | 8 |
| | 8 | 8 | 16 |
| | 9 | 6 | 12 |
| | 10 | 2 | 4 |

*4.2. Proposed Reduce Task Data Locality based Scheduler*

In Hadoop, the *Reduce* phase does not start until the $Map$ phase produces all the intermediate data. To get the final output of a job, it has to wait for all the *map* tasks to finish. Therefore, after *map* task data locality scheduler completes its execution, *reduce* task data locality scheduler starts its execution.

We arrange the nodes such that it has *Reduce* function and maximum partition of intermediate $(key, value)$ pairs in the cluster in order as in (11).

$$RN(i) \leftarrow RN_1, RN_2, ....., RN_m \tag{11}$$

Here, $RN(i)$ is $i^{th}$ Node with *Reduce* function (Reducer node) in the heterogeneous cluster, and $m$ is the number of nodes with *Reduce* function in the cluster.

Arrange the Fast processing nodes in the cluster in order. Initially, for every node we check if both reducer node and fast processing node are same in the Hadoop cluster.

If both the nodes are same in the cluster, then continue otherwise assign intermediate data of the reducer nodes to the fast processing nodes as in (12).

$$FPN(i) \leftarrow RN(i) \tag{12}$$

First, check if any *Reduce* slots are available on the fast processing data

node. If available then assign *reduce* tasks ($RT$) to the $FPN(i)$ in the cluster as shown in (13).

$$FPN(i) \leftarrow RT \tag{13}$$

Otherwise, wait for *reduce* task assignment until it satisfies the $NDT$ (Node Delay Threshold) as in (14). If it satisfies, then assign the *reduce* tasks to the fast processing node as in (15).

$$WaitNode \leqslant NDT \tag{14}$$

$$FPN(i) \leftarrow RT \tag{15}$$

We present a scheduler which relies on data locality and schedules *reduce* tasks to a node which has maximum processing capacity in the heterogeneous cluster as shown in algorithm 2.

## 5. Performance Evaluation

In this section, the proposed data locality based scheduler performance is assessed by a series of experimental results and analysis. We evaluate our scheduling approach using metrics such as:

1. Job execution time: it is the most basic measurable standard for a good scheduler because the purpose of developing a scheduling algorithm is to minimize the execution time of a job.
2. Data locality: it is measured by the total number of tasks run locally in the data residing node.

We implement our scheduling method by doing a modification to the Hadoop framework. We first present the local test-bed before discussing our experimental results.

### *5.1. Experimental Environment*

We performed the experiments on a heterogeneous Hadoop cluster whose configuration is shown in Table 3. Our local test-bed contains one master node ($NameNode$ and $JobTracker$) and six slave nodes ($DataNodes$ and $TaskTrackers$). We measure the heterogeneity in the cluster by having various CPU types, memory size and disk space on each node. All nodes in the cluster were connected with a Gigabit Ethernet switch and ran on Ubuntu

14.04 operating system. In our experiments, we deployed and configured Hadoop 1.2.1 stable version with a block size of 128 MB and JDK version 8. We maintain three replicas for each data block in this cluster to increase the availability of the data. Hadoop1 is the widespread production today, and it is the version for which most of the Hadoop application ecosystem has been developed. In future, we will work on Hadoop2/YARN.

**Algorithm 2 Proposed Algorithm for Reduce Task Data Locality**

**Input:** Set of Unscheduled Reduce Tasks (URT).

**Output:** A ReduceTask $RT \in URT$ that can be scheduled on a fast processing node.

1: **for** each *DataNode* $i$ in the heterogeneous cluster **do**
2: Find Reducer Nodes that has maximum partition of Intermediate $(key, value)$ pairs in order
3: $RN(i) \leftarrow RN_1, RN_2, ....., RN_m$
4: Find the Fast processing nodes in the cluster in order
5: **if** $RN_i = FPN_i$ **then**
6: Continue;
7: **else**
8: $FPN_i \leftarrow RN_i$
9: **end if**
10: **if** *Reduce* slots available on $FPN$ **then**
11: Assign *reduce* tasks to $FPN$
12: $FPN(i) \leftarrow RT_i$
13: **else**
14: $WaitNode \leqslant NodeDelayThreshold$
15: $FPN(i) \leftarrow RT_i$
16: **end if**
17: **end for**

*5.2. Workload Description (Benchmarks)*

We evaluated our proposed scheduler with Hi-Bench benchmark suite [8] as it is a novel, comprehensive, realistic and widely-used benchmark aimed at

Hadoop. It has different benchmarks like: Micro benchmarks (*WordCount*, *Sort*, and *TeraSort*), Machine learning benchmarks (*BayesianClassification*, *K-meansClustering*) and Web search benchmarks (*NutchIndexing*, *PageRank*). We have taken the default data sizes of all these workloads of Hi-Bench benchmark suite. *Sort* and *WordCount* has 60GB and *TeraSort* has 1TB as input data. *BayesianClassification* and *K-meansClustering* has 63GB and 132GB respectively. NutchIndexing and *PageRank* has 8.4GB and 3.63GB respectively.

Table 3: Hadoop evaluation environment

| **Node Type** | **Hardware Configuration** | **Hadoop Configuration** |
|---|---|---|
| Master node | Intel Xeon CPU L5520 @ 2.27 GHz * 8, 16 GB RAM, 1000 GB Disk space | 8 *map* slots and 5 *reduce* slots |
| Slave node 1 | Intel core i5- 4570T CPU @ 2.90 GHz * 4, 4 GB RAM, 1000 GB Disk space | 4 *map* slots and 2 *reduce* slots |
| Slave node 2 | Intel core 2 duo CPU E7500 @ 2.93 GHz * 2, 4 GB RAM, 500 GB Disk space | 2 *map* slots and 1 *reduce* slot |
| Slave node 3 | Intel Pentium D CPU @ 3.00 GHz * 2, 3 GB RAM, 500 GB Disk space | 2 *map* slots and 1 *reduce* slot |
| Slave node 4 | Intel core 2 duo Processor P8400 @ 2.26 GHz * 2, 3 GB RAM, 250 GB Disk space | 2 *map* slots and 1 *reduce* slot |
| Slave node 5 | Intel core i5- 4570T CPU @ 2.90 GHz * 4, 4 GB RAM, 1000 GB Disk space | 4 *map* slots and 2 *reduce* slots |
| Slave node 6 | Intel core i5- 2500 CPU @ 3.30 GHz * 4, 8 GB RAM, 500 GB Disk space | 4 *map* slots and 2 *reduce* slots |

*5.3. Performance Analysis of the Proposed Scheduler*

To evaluate our proposed data locality based scheduler, we compared its performance with the Hadoop default scheduling algorithm, Matchmaking

scheduler, and Delay scheduler. Comparison with Delay scheduler is more appropriate as it is a state-of-the-art scheduler and is widely used in the Hadoop framework. We set up a small local Hadoop cluster of 7 nodes, and it can be scaled to medium size cluster. We performed five runs in each of the experiment for evaluating proposed scheduler in heterogeneous environments.

*5.3.1. Micro Benchmarks*

The *WordCount*, *Sort*, and *TeraSort* workloads are widely used in the Hadoop research community. Both the *Sort* and *WordCount* programs are illustrative of a large subsection of practical jobs MapReduce.

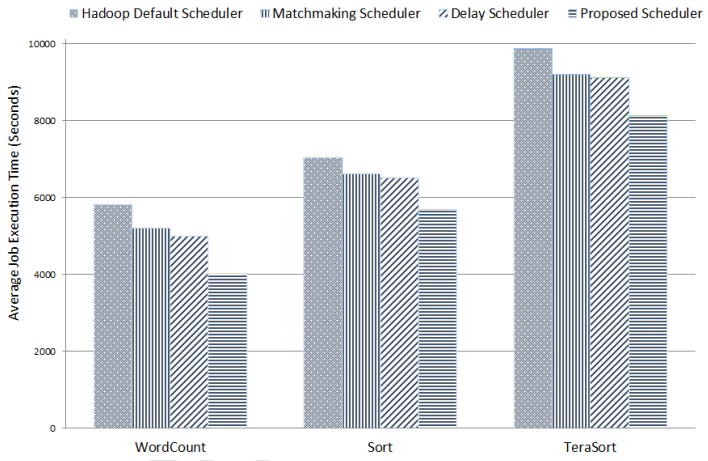


Figure 3: Job execution time for Micro Benchmarks

The *WordCount* counts the occurrence of words from input data, which are generated using RandomTextWriter. On an average, the proposed scheduler finishes jobs 19.5% faster than Delay scheduler, 22.8% faster than Matchmaking scheduler and 30.8% faster than Hadoop default scheduler as shown in Fig. 3.

The *Sort* workload sorts the text data, which is produced with RandomWriter. On an average, the proposed scheduler finishes jobs 14.3% faster than Delay scheduler, 15.6% faster than Matchmaking scheduler and 18.9% faster than Hadoop default scheduler as shown in Fig. 3.

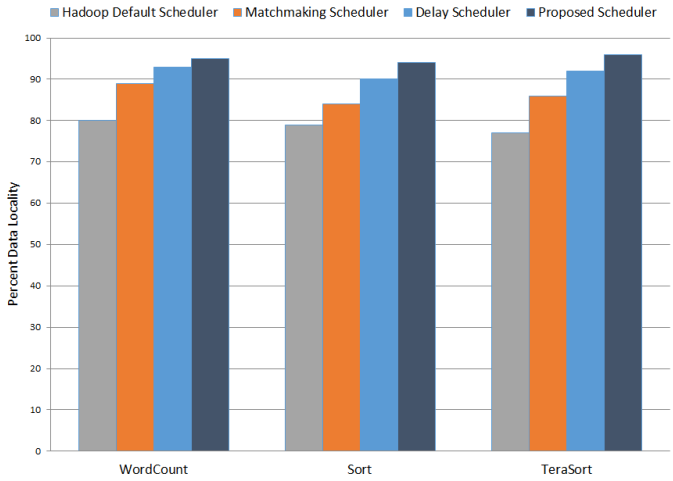


Figure 4: Data Locality for Micro Benchmarks

The *TeraSort* workload categorizes 10 billion 100-byte records produced by the TeraGen program. On an average, the proposed scheduler finishes jobs 10.9% faster than Delay scheduler, 11.8% faster than Matchmaking scheduler and 17.8% faster than Hadoop default scheduler as presented in Fig. 3.

We observe that our proposed data locality scheduler shows significantly higher data locality rate than Hadoop default scheduler, Matchmaking scheduler and Delay scheduler for all of the workloads from Hi-Bench benchmark suite. Our proposed scheduler has data local tasks to at most 95%, 94% and 96% for *WordCount*, *Sort* and *TeraSort* benchmarks respectively as shown in Fig. 4.

*5.3.2. Web Search Benchmarks*

The *NutchIndexing* and *PageRank* benchmarks are comprised in Hi-Bench as they are representatives of large-scale search indexing systems.

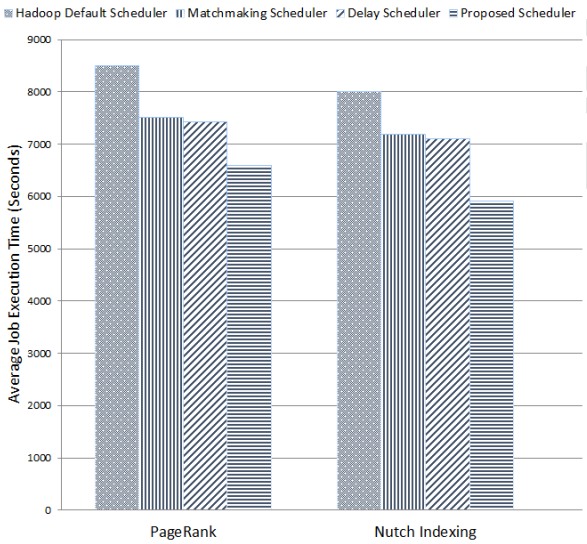


Figure 5: Job execution time of Web Search Benchmarks

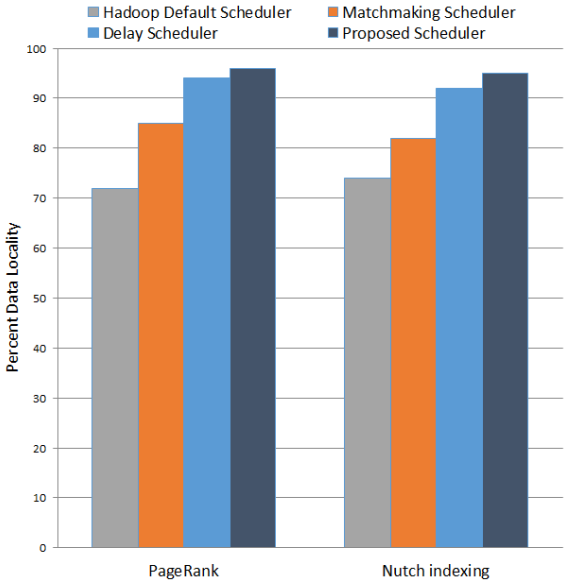


Figure 6: Data Locality for Web Search Benchmarks

The *PageRank* workload is an open source implementation of the page-rank algorithm. On an average, the proposed scheduler finishes jobs 11.2% faster than Delay scheduler, 12.2% faster than Matchmaking scheduler and 22.5% faster than Hadoop default scheduler as shown in Fig. 5.

The *NutchIndexing* benchmark is the indexing sub-system of Nutch. On an average, the proposed scheduler finishes jobs 16.6% faster than Delay scheduler, 17.8% faster than Matchmaking scheduler and 29.8% faster than Hadoop default scheduler as shown in Fig. 5.

Our proposed scheduler has data local tasks to at most 96% and 95% for *PageRank* and *NutchIndexing* benchmarks respectively as shown in Fig. 6.

### *5.3.3. Machine Learning Benchmarks*

The *BayesianClassification* and *K-meansClustering* implementations are contained in Mahout. They are included in Hi-Bench because of the significant additional uses of large-scale machine learning.

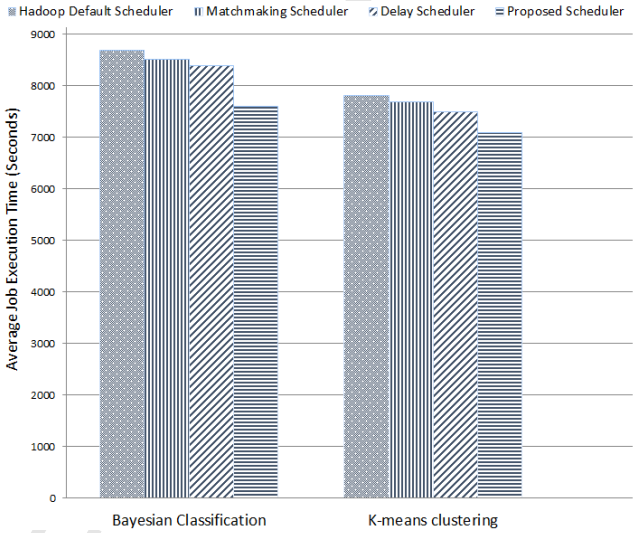


Figure 7: Job execution time for Machine Learning Benchmarks

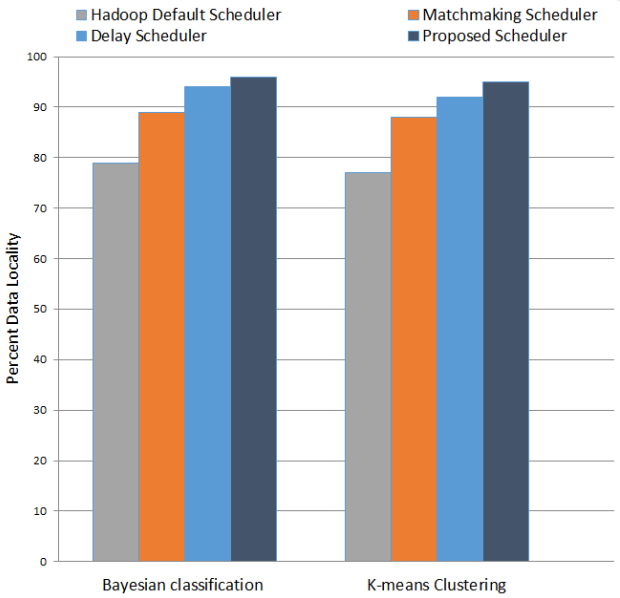


Figure 8: Data Locality for Machine Learning Benchmarks

The *BayesianClassification* benchmark implements the trainer part of Naive. On an average, the proposed scheduler finishes jobs 9.4% faster than Delay scheduler, 10.6% faster than Matchmaking scheduler and 12.5% faster than Hadoop default scheduler as shown in Fig. 7.

The *K-meansClustering* benchmark implements K-means. On an average, the proposed scheduler finishes jobs 5.3% faster than Delay scheduler, 7.6% faster than Matchmaking scheduler and 9.1% faster than Hadoop default scheduler as shown in Fig. 7.

Our proposed scheduler has data local tasks to at most 96% and 95% for *BayesianClassification* and *K-meansClustering* benchmarks respectively as shown in Fig. 8.

### *5.3.4. Hive Benchmarks*

In practice, Hive benchmarks are used more widely than hand-coded MapReduce programs for big data query and analysis applications. One motivation of our optimization work is to benefit from big data query and analysis systems. Therefore, we also evaluate the impact of performance improvement for the Hive benchmarks. In this experiment, we use Hive as the

big data query and analysis system and run the Hive benchmarks(*Scan*, *Join* and *Aggregation*) on the Hadoop default scheduler, Matchmaking scheduler, Delay scheduler, and proposed scheduler respectively.

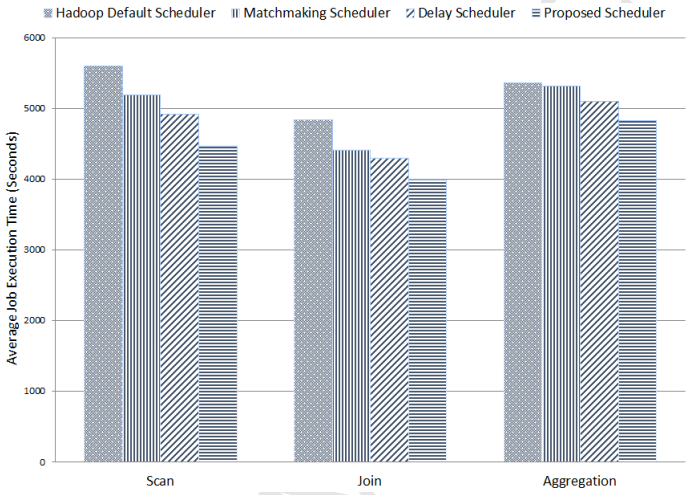


Figure 9: Job execution time of Hive Benchmarks

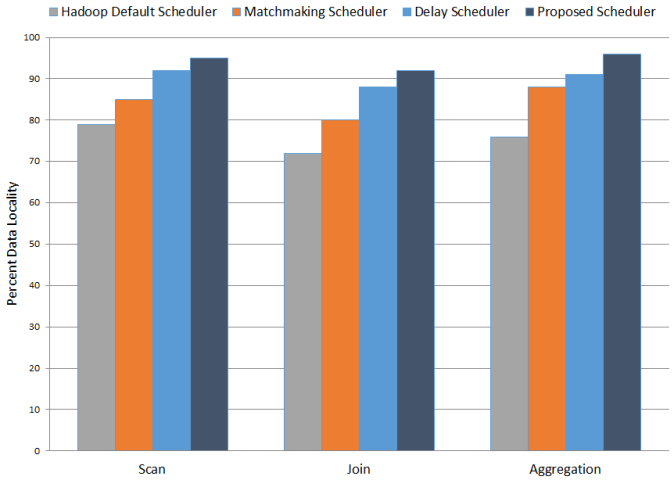


Figure 10: Data Locality for Hive Benchmarks

On an average, the proposed scheduler finishes jobs 10.3% faster than Delay scheduler, 14.1% faster than Matchmaking scheduler and 25.6% faster than Hadoop default scheduler by running *Scan* benchmark as shown in Fig. 9.

On an average, the proposed scheduler finishes jobs 7.5% faster than Delay scheduler, 9.4% faster than Matchmaking scheduler and 21% faster than Hadoop default scheduler by running *Join* benchmark as shown in Fig. 9.

On an average, the proposed scheduler finishes jobs 5.7% faster than Delay scheduler, 9.3% faster than Matchmaking scheduler and 11% faster than Hadoop default scheduler by running *Aggregation* benchmark as shown in Fig. 9.

Our proposed scheduler has data local tasks to at most 95%, 92% and 96% for *Scan*, *Join* and *Aggregation* benchmarks respectively as shown in Fig. 10.

Table 4: Average job execution time (Seconds) for different workloads of Hi-Bench benchmark suite

| **Hi-Bench benchmark suite** | **Average job execution time (Seconds)** | | | |
|---|---|---|---|---|
| | **Hadoop scheduler** | **Matchmaking scheduler** | **Delay scheduler** | **Proposed Data Locality scheduler** |
| WordCount | 5800 | 5200 | 4987 | 4012 |
| Sort | 7012 | 6690 | 6498 | 5682 |
| TeraSort | 9870 | 9200 | 9101 | 8108 |
| PageRank | 8500 | 7509 | 7419 | 6586 |
| Nutch Indexing | 7998 | 7194 | 7094 | 5911 |
| Bayesian Classification | 8700 | 8517 | 8403 | 7609 |
| K-means Clustering | 7821 | 7690 | 7501 | 7103 |
| Scan | 5600 | 5193 | 4916 | 4458 |
| Join | 4832 | 4408 | 4293 | 3990 |
| Aggregation | 5356 | 5319 | 5101 | 4824 |

In all of these different workloads from Hi-Bench benchmark suite [8], proposed data locality based scheduler accomplished the best and consistent results regarding the minimum average job execution time and higher data locality rate as compared to the Hadoop default scheduler, Matchmaking scheduler and Delay scheduler in heterogeneous environments as presented in Table 4 and Table 5.

Table 5: Data Locality for different workloads of Hi-Bench benchmark suite

| **Hi-Bench benchmark suite** | **Data locality rate (%)** | | | |
|---|---|---|---|---|
| | **Hadoop scheduler** | **Matchmaking scheduler** | **Delay scheduler** | **Proposed Data Locality scheduler** |
| WordCount | 80 | 89 | 93 | 95 |
| Sort | 79 | 84 | 90 | 94 |
| TeraSort | 77 | 86 | 92 | 96 |
| PageRank | 72 | 85 | 94 | 96 |
| Nutch Indexing | 74 | 82 | 92 | 95 |
| Bayesian Classification | 79 | 89 | 94 | 96 |
| K-means Clustering | 77 | 88 | 92 | 95 |
| Scan | 79 | 85 | 92 | 95 |
| Join | 72 | 80 | 88 | 92 |
| Aggregation | 76 | 88 | 91 | 96 |

*5.3.5. Measuring Data Locality for Different Replication Factor and Block Size*

Theoretically, Replication factor increases the possibility of achieving better data locality. If we use more replicas, it means that Hadoop can always find a close data node for computation, resulting in less communication overhead. The impact of different replication factors on Hadoop default scheduler, Matchmaking scheduler, Delay scheduler and the Proposed data locality based scheduler are shown in Fig. 11 for different workloads from Hi-Bench benchmark suite [8]. The percentage of local tasks increases for all the configurations if we increase the number of replicas per block. The increase of replication factor yields substantial data locality improvement for proposed

scheduler as compared to Delay scheduler, Matchmaking and Hadoop default schedulers, but more storage space is also required for increased replicas. So, we have selected the best replication factor as 3 for our local test-bed cluster to balance the storage usage and it achieves the best possible data locality.

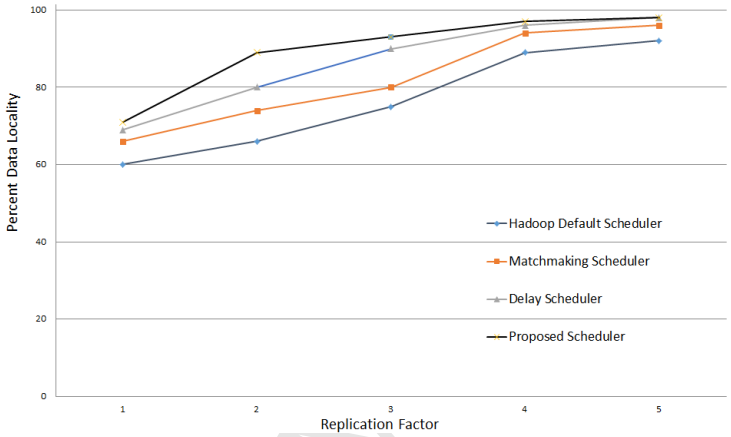


Figure 11: Varied replication factor for Hi-Bench benchmarks

Table 6: Data Locality for Different Replication Factors

| **Replication Factor** | **Data locality rate (%)** | | | |
|---|---|---|---|---|
| | **Hadoop scheduler** | **Matchmaking scheduler** | **Delay scheduler** | **Proposed Data Locality scheduler** |
| 1 | 60 | 66 | 69 | 71 |
| 2 | 66 | 74 | 80 | 89 |
| 3 | 75 | 80 | 90 | 93 |
| 4 | 89 | 94 | 96 | 97 |
| 5 | 92 | 96 | 98 | 98 |

We conduct set of experiments for different data block size by executing dissimilar workloads from Hi-Bench benchmark suite [8]. We compare our proposed scheduler with Hadoop default scheduler, Matchmaking scheduler and Delay scheduler for different block size as 64MB, 128MB, and 256 MB

as shown in Fig. 12. The proposed scheduler has improved the percentage of data locality with the increase of block size because the data transmission time becomes longer when block size increases.

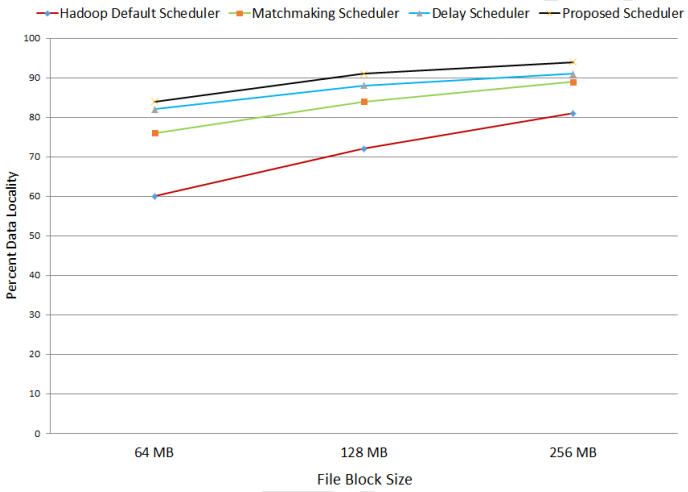


Figure 12: Comparison of different block size for Hi-Bench benchmarks

Table 7: Data Locality for Different Block size

| **Block Size** | **Data locality rate (%)** | | | |
|---|---|---|---|---|
| | **Hadoop scheduler** | **Matchmaking scheduler** | **Delay scheduler** | **Proposed Data Locality scheduler** |
| 64 MB | 60 | 76 | 82 | 84 |
| 128 MB | 72 | 84 | 88 | 91 |
| 256 MB | 81 | 89 | 91 | 94 |

Our proposed data locality based scheduler achieved the consistent results regarding the data locality rate in heterogeneous environments compared to the Hadoop default scheduler, Matchmaking scheduler and Delay scheduler as presented in Table 6 and Table 7. Proposed scheduler minimizes the execution time of jobs by achieving maximum data locality. Thus, it improves the performance of the MapReduce framework in heterogeneous environments.

## 6. Conclusion and Future work

This paper proposed a novel data locality based scheduling algorithm which enhances the MapReduce framework performance in heterogeneous Hadoop cluster. Proposed scheduler dynamically divides the input data and assigns the data blocks according to the node processing capacity. It also schedules the *map* and *reduce* tasks according to the processing capacity of nodes in the heterogeneous Hadoop cluster. Experimental results prove that our proposed data locality based scheduler performs significantly better regarding minimum average job execution time and data locality rate as compared to the Hadoop default scheduler, Matchmaking scheduler and state of the art Delay scheduler by running different workloads from Hi-bench benchmark suite on a local heterogeneous Hadoop cluster.

As part of our future research work, we like to perform an extensive and comprehensive experiment on a large heterogeneous Hadoop cluster to enhance the MapReduce performance.

## References

[1] Dean, J. and Ghemawat, S. 'Mapreduce: Simplified data processing on large clusters', *Communications of the ACM*, Vol. 51, pp.107–113 (2008).

[2] Jiang, D., Ooi, B.C., Shi, L. and Wu, S. 'The performance of mapreduce: an in-depth study', *Proc. VLDB Endow.*, pp.472–483 (2010).

[3] Dean, J. and Ghemawat, S. 'Mapreduce: a flexible data processing tool', *Communications of the ACM*, pp.72–77 (2010).

[4] Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R. and Stoica, I. 'Improving MapReduce performance in heterogeneous environments', *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, pp.29–42, Berkeley, USA (2008).

[5] Tan, J., Meng, X. and Zhang, L. 'Delay tails in mapreduce scheduling', *SIGMETRICS Perform. Eval. Rev.*, pp.5–16 (2012).

[6] Zaharia, M., Borthakur, D., Sarma, J.S., Elmeleegy, K., Shenker, S. and Stoica, I. 'Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling,' *Proceedings of the 5th European Conference on Computer Systems*, vol.14, pp.265–278 (2010).

[7] He, C., Lu, Y. and Swanson, D. 'Matchmaking: A New MapReduce Scheduling Technique,' *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp.40–47 (2011).

[8] Shengsheng, H., Jie, H., Jinquan, D., Tao, X. and Huang, B. 'The HiBench benchmark suite: Characterization of the MapReduce-based data analysis', *IEEE 26th International Conference on Data Engineering Workshops*, pp.41–51, Long Beach, CA (2010).

[9] Rasooli, A. and Down, D.G. 'A Hybrid Scheduling Approach for Scalable Heterogeneous Hadoop Systems,' *Proceeding of the 5th Workshop on Many-Task Computing on Grids and Supercomputers*, pp.1284–1291 (2012).

[10] Guo, Z. and Fox, G. 'Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization', *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp.714–716, Washington, DC, USA (2012).

[11] Tian, C., Zhou, H., He, Y. and Zha, L. 'A dynamic MapReduce scheduler for heterogeneous workloads', *Eighth International Conference on Grid and Cooperative Computing*, pp.218–224, Lanzhou, Gansu (2009).

[12] Zhenhua, G., Fox, G. and Zhou, M. 'Investigation of Data Locality in MapReduce,' *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp.419–426 (2012).

[13] Chunguang Wang, W., Qingbo, W., Yusong, T., Wenzhu, W. and Quanyuan, W. 'Locality Based Data Partitioning in MapReduce,' *IEEE 16th International Conference on Computational Science and Engineering (CSE)*, pp.1310–1317 (2013).

[14] Rasooli, A. and Down, D.G. 'Guidelines for Selecting Hadoop Schedulers Based on System Heterogeneity', *Journal of Grid Computing*, pp.1–26 (2014).

[15] Romsaiyud, W. and Premchaiswadi, W. 'An adaptive machine learning on Map-Reduce framework for improving performance of large-scale data analysis on EC2,' *11th International Conference on ICT and Knowledge Engineering*, pp.1–7 (2013).

[16] Rasooli, A. and Down, D.G. 'COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems', *Journal of Future Generation Computer Systems*, pp.1–15 (2014).

[17] Tang, Z., Jiang, L., Zhou, J., Li, K. 'A self-adaptive scheduling algorithm for reduce start time,' *Future Generation Computer Systems*, vol. 43, pp.51–60 (2015).

[18] Sun, M., Zhuang, H., Li, C., Lu, K. and Zhou, X. 'Scheduling algorithm based on prefetching in MapReduce clusters,' *Applied Soft Computing*, vol. 38, pp.1109–1118 (2016).

[19] Tiwari, N., Sarkar, S., Bellur, U. and Indrawan, M. 'Classification Framework of MapReduce Scheduling Algorithms,' *ACM Computing Survey*, vol. 47, number 3, pp.49:1–49:38 (2015).

[20] Jin, J., Luo, J., Song, A., Dong, F., and Xiong, R. 'BAR: An Efficient Data Locality Driven Task Scheduling Algorithm for Cloud Computing,' *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp.295–304, Newport Beach, CA, (2011).

[21] Ching-Hsien, H., Kenn, D., Slagter, Yeh-Ching, C. 'Locality and loading aware virtual machine mapping techniques for optimizing communications in MapReduce applications,' *Future Generation Computer Systems*, pp.43–54, vol. 53, (2015).

[22] Kenn, S., Ching-Hsien, H., and Yeh-Ching, C. 'An Adaptive and Memory Efficient Sampling Mechanism for Partitioning in MapReduce,' *International Journal of Parallel Programming*, pp.489-507, vol. 43, number 3, (June 2015).

[23] Shabeera, T. P., Madhu Kumar, S. D. 'Optimising virtual machine allocation in MapReduce cloud for improved data locality,' *International Journal of Big Data Intelligence (IJBDI)*, pp.2–8, vol. 2, number 1, (2015).

[24] Song, K., and Hongwei, Lu. 'Efficient Querying Distributed Big-XML Data using MapReduce,' *International Journal of Grid and High Performance Computing*, pp.70–79, vol. 8, number 3 (July 2016).

[25] Wang, W., Zhu, K., Ying, L., Tan, J., and Zhang, L. 'Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality,' *IEEE/ACM Transactions on Networking*, pp.190–203, vol. 24, number 1, (2016).

[26] Elhoseny H., Elhoseny M., Riad A.M., Hassanien A.E., 'A Framework for Big Data Analysis in Smart Cities', *International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2018)*, vol. 723, (2018). (DOI: https://doi.org/10.1007/978-3-319-74690-6-40)

[27] Mohamed Elhoseny, Abdulaziz Shehab and Xiaohui Yuan, 'Optimizing Robot Path in Dynamic Environments Using Genetic Algorithm and Bezier Curve', *Journal of Intelligent & Fuzzy Systems*, vol. 33, no. 4, pp.2305–2316, (2017). (DOI: 10.3233/JIFS-17348)

[28] Mohamed Elhoseny, Alaa Tharwat, Aboul Ella Hassanien, 'Bezier Curve Based Path Planning in a Dynamic Field using Modified Genetic Algorithm', *Journal of Computational Science*, (2017). (https://doi.org/10.1016/j.jocs.2017.08.004)

[29] Mohamed Elhoseny, Ahmed Abdelaziz, Ahmed Salama, AM Riad, Arun Kumar Sangaiah, Khan Muhammad, 'A Hybrid Model of Internet of Things and Cloud Computing to Manage Big Data in Health Services Applications', *Future Generation Computer Systems*, (Accepted March 2018), (In Press)

[30] Alaa Tharwat, Mohamed Elhoseny, AboulElla Hassanien, Thomas Gabel, and N. Arun kumar, 'Intelligent Bezir Curve-based Path Planning Model Using Chaotic Particle Swarm Optimization Algorithm,' *Cluster Computing*, pp.1–22, (March 2018). (DOI: 10.1007/s10586-018-2360-3)

[31] Ali Asghar Rahmani Hosseinabadi, Javad Vahidi, Behzad Saemi, Arun Kumar Sangaiah, Mohamed Elhoseny, 'Extended Genetic Algorithm for solving open-shop scheduling problem', *Soft Computing*, (April 2018) (https://doi.org/10.1007/s00500-018-3177-y)

R.Anitha, received UG degree, B.Com. From the University of Madras, Chennai, India, in 2000. PG degree, Master of Computer Applications (MCA) from the University of Madras, Chennai, India, in 2003. Another PG Degree, Master of Philosophy (M.Phil.) in Computer Science from Vinakya Mission, Chennai, India, in 2009. Another PG Degree, Master of Technology (M.Tech) in Computer Science from SRM University, Chennai, India, in 2013. She has been working with Department of Computer Applications (MCA), S.A. Engineering College, Chennai, India. She got 2 international publications. Research interest includes Wireless Sensor Networks, Mobile Adhoc Networks & Bigdata.

TAPAS BAPU B R obtained his Ph.D. in Electronics and Communication Engineering department from St. Peters University Chennai -54. He is currently working as Professor, Faculty of Electronics and Communication Engineering, S. A. Engineering College, Chennai -77. He has a total teaching experience of 20.6 years in engineering colleges alone.

He has published 17 plus international journals which includes 3 papers in Web of Science and 5 in Scopus indexing. He is reviewers for few international journals in Springer and Inderscience publications.

His area of research is Wireless Sensor Networks, Network Security and Image Processing. He is also interested in Digital Electronics, Microprocessor and Microcontroller, Analog and Digital Communication, Linear Integrated Circuits, and Control Systems.

Completed B.E. (ECE) in the year 1997 from National Engineering College, Kovilpatti, Tamilnadu, India. Affiliated to Manonmaniam Sundaranar University.

Completed M.E (Applied Electronics) in the year 2004 from Hindustan College of Engineering, Chennai. Affiliated to Anna University, Chennai.

Nenavath Srinivas Naik received his Ph.D. and M.Tech in Computer Science from School of Computer and Information Sciences, University of Hyderabad, Hyderabad, India in 2017 and 2010 respectively. Received his B.E. in Computer Science and Engineering from Osmania University, Hyderabad, India in 2005. Currently, he is working as Senior Assistant Professor in Department of Computer Science and Engineering, Madanapalle Institute of Technology and Science (MITS), Madanapalle, Andhra Pradesh, India. His research interests focus on Big Data, Cloud Computing, Parallel and Distributed Computing. He has published papers in several international journals/conferences. He presented his research work as poster in reputed international conferences like 29$^{th}$ IEEE IPDPS 2015, 21$^{st}$ IEEE ADCOM 2015 and 23$^{rd}$ IEEE HIPC 2016.

Dr. Atul Negi (Senior Member IEEE, Life Member IUPRAI) is presently working as Professor in School of Computer and Information Sciences, University of Hyderabad, India. He has reviewed papers for several international journals like IEEE Transactions on SMC, Pattern Recognition, Pattern Recognition Letters, Image and Vision Computing, Computers and Security amongst others. He has served on the technical program committee of several conferences such as IEEE SMC, and reviewed papers for many other international conferences such as ICDAR etc. He has research interests in fields of Grid Computing, Pattern Recognition, Optical Character Recognition, and Soft Computing. He held position of Director, Prestige Institute of Engineering and Science, Indore. He worked as Scientific Officer for DRDO Project COMRADES, IISc. He worked as an Investigator for several funded projects funded by ISRO, Ministry of Communications and IT.

The highlights of our research work are the following:

1. Proposal of a Data distribution method that dynamically distributes input data to the nodes depending on their processing capacity in the cluster.

2. Development of a data locality based scheduler that schedules *map* and *reduce* tasks to different nodes in a heterogeneous cluster by their processing capacity. (i.e. Nodes with fast processing capacity will be assigned more tasks than slower ones).

3. Comparison of our proposed scheduling approach with the state-of-the-art schedulers using heterogeneous workloads from Hi-Bench benchmark suite in heterogeneous Hadoop cluster.

4. Proposal of scheduling approach for reducing data movement activities in a cluster by improving data locality rate and minimizing the job execution time. Thus, the proposed approach enhances MapReduce performance in heterogeneous environments.