Data Locality and VM Interference Aware Mitigation of Data Skew in Hadoop Leveraging Modern Portfolio Theory

Seyed Morteza Nabavinejad
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
mnabavi@ce.sharif.edu

Maziar Goudarzi
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
goudarzi@sharif.edu

ABSTRACT

Data skew, which is the result of uneven distribution of data among tasks in big data processing frameworks such as MapReduce, causes significant variation in the execution time of tasks and makes their placement on computing resources more challenging. Moreover, with the proliferation of big data processing in the cloud, the interference among virtual machines co-located on the same physical machine exacerbates the aforementioned variation. To tackle this challenge, we propose Locality and Interference aware Portfolio-based Task Assignment (LIPTA) approach. LIPTA leverages the modern portfolio theory to mitigate the variation in execution time of tasks while considering the interference of virtual machines and locality of input data. It selects and assigns groups of tasks (the portfolio) to each machine such that variation of their total execution time is reduced due to portfolio effect. Experimental results using real-world workload logs demonstrate the effectiveness of our LIPTA approach. It can reduce the total execution time of workloads by up to 46.7% compared with several variation-oblivious approaches.

CCS CONCEPTS

• **General and reference → Performance**; • **Computer systems organization →** *Cloud computing*;

KEYWORDS

cloud computing, big data, data skew, portfolio theory

ACM Reference Format:
Seyed Morteza Nabavinejad and Maziar Goudarzi. 2019. Data Locality and VM Interference Aware Mitigation of Data Skew in Hadoop Leveraging Modern Portfolio Theory. In *Proceedings of ACM SAC Conference (SAC'18).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data skew is an inevitable feature of jobs in big data processing frameworks such as MapReduce [5]. It can dramatically change the execution time of tasks and create significant variation among them, which makes the resource allocation and task assignment of big data jobs an arduous challenge. It also has detrimental impact on the current proposed performance estimation [2, 12, 17, 27, 28, 35] and resource allocation and task scheduling [21, 22, 26, 29, 36] approaches for big data jobs that are data skew-oblivious. These approaches profile the tasks prior to allocation/scheduling to predict the execution time of tasks (and not their distribution) or their resource demand. Changing the input data affects the data skew pattern and its impact on tasks, and hence, makes the profiling mechanism of these approaches less effective. Both Map and Reduce tasks suffer from this phenomenon in MapReduce applications and a large body of research has tried to tackle this issue [4, 10, 19]. Most of the currently proposed approaches address data skew in Reduce tasks by modifying the partitioning phase of MapReduce applications [4, 13, 23, 33, 38]. However, only a few approaches try to tackle the data skew problem in Map tasks [10]. The reason is that applications have control over the input data of Reduce tasks. They can partition the key-value pairs of intermediate data (which is the input of Reduce phase) and determine the input size and input distribution of Reduce tasks. But for Map tasks, usually, the input data (data chunks or data blocks) are already copied to the HDFS and application has no control over them.

Moreover, with the explosive growth of big data processing, using the cloud to provide the computing and storage resources for big data applications is on the increase. Users deploy their applications either on public clouds that offer dedicated services for big data processing such as Amazon EMR [1] or on private clouds. Processing big data on the cloud has various advantages such as VM migration [20] which can facilitate resource management. But, it also introduces challenges such as VM interference [24] that can exacerbate the variation of tasks' execution time caused by data skew.

Previous approaches such as FlexSlot [10] address the data skew in Map tasks by changing the resources of their corresponding slots rather than task assignment. FlexSlot first identifies the stragglers, tasks that are slower than average, and then increases the resources of their slots to improve their performance. The low performance of a task might be the result of data skew or the interference among VMs. It also uses the speculative execution to further mitigate data skew. While these mechanisms can improve the performance of applications, they impose resource overhead and extra monetary cost due to concurrent execution of several instances of one task. Furthermore, when FlexSlot decides to increase the slot's resources of a machine due to VM interference, it does not readjust its resources after the VM interference is resolved, which can lead to resource under-utilization.

To tackle the execution time variation of MapReduce applications, as well as the interference among co-located VMs in the cloud, we propose our Locality and Interference Aware Portfolio-based Task Assignment (LIPTA) approach. LIPTA considers an environment where several MapReduce jobs share the same VM cluster and execution time of tasks of each job has its own normal distribution.

SAC'18, April 9-13, 2018, Pau,France
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

To have the distribution of tasks' execution time, our approach uses profiling. However, unlike other approaches, it does not rely on the exact value of execution time of tasks, but only needs mean and standard deviation of tasks' execution time. It leverages the portfolio effect introduced in section 2.2 to reduce the execution time of tasks assigned to each machine. Furthermore, It takes into account both placement of input blocks on machines (data locality) and VM interference when assigning tasks to VMs. Moreover, LIPTA divides tasks into several subsets of tasks and assign each subset in one interval, instead of assigning all the tasks at the beginning. By this, LIPTA is able to adapt itself to changes that happen during the course of time such as VM interference. The overall flow of LIPTA is depicted in Figure 1

In this study, we discuss the MapReduce applications because the real world workloads that we have belongs to a Hadoop cluster that executes such applications. However, the proposed approach can be generalized to any other programming paradigm that suffers data skew or VM interference, and the data locality affects the execution time of the tasks.

The rest of the paper is organized as follows: in section 2 we discuss the motivation behind our study and introduce the required mathematical background. Our proposed approach is introduced in section 3. We evaluate our approach and present results of experiments in section 4 and summarize the relevant previous works in section 5. Finally, we conclude the paper and outline directions for future research in section 6.

2 MOTIVATION AND BACKGROUND

In this section, we first present a motivational example to show the wide variety of tasks' execution time in MapReduce applications using the Hadoop log dataset. After that, we discuss Modern Portfolio Theory and explain how we are going to use it to mitigate the variation of tasks' execution time.

2.1 Motivational Example

To show how data skew affects the execution time of Map tasks in MapReduce applications, we employ the Hadoop log dataset [31]. Depicting the execution time of Map tasks for several jobs of these logs reveals a significant variation in them. Figure 2 demonstrates this variation. jobs are identified by their ID in logs. We see that for all 10 jobs, there is a wide gap between minimum and maximum execution time of tasks and the variation is also very high; note that the horizontal lines of each box in Figure 2 show the 25th, 50th, and 75th percentile of dots (execution time of tasks).

2.2 Mathematical Background

Modern Portfolio Theory (MPT) developed by Harry Markowitz [25] is a theory of finance. Using this theory, an investor can form his or her portfolio by selecting proportions of different assets based on individual risk of each asset in order to reduce the risk of the portfolio for a certain expected return. Return of each asset is modeled as a normal distribution where the risk of the asset is the standard deviation of the distribution. Since the correlation between assets would affect the final risk of the portfolio, MPT aims to reduce the risk of a portfolio by selecting the assets that do not have a perfect correlation with each other [7, 32].

MPT leverages portfolio effect to reduce the risk. The portfolio effect may be defined as follows: A reduction in variation (risk) of returns on a portfolio, which is a proportional selection of different assets, compared with the average of the variations (risk) of the individual assets. If we indicate each asset X_i with mean of its return μ_i and standard deviation of its return, σ_i, then for the portfolio Y, which is a combination of N assets, we would have [15]:

$$\mu_Y = \sum_{i=1}^{N} \mu_{X_i}, \sigma_Y = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N} \rho_{ij}\sigma_i\sigma_j} \leq \sum_{i=1}^{N} \sigma_i \tag{1}$$

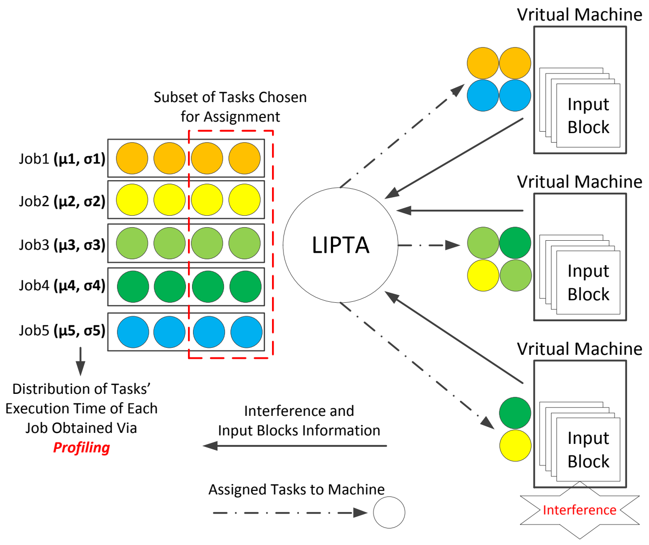
Vritual Machine
Vritual Machine
Vritual Machine
Input Block
Input Block
Input Block
Subset of Tasks Chosen for Assignment
Job1 (μ1, σ1)
Job2 (μ2, σ2)
Job3 (μ3, σ3)
Job4 (μ4, σ4)
Job5 (μ5, σ5)
LIPTA
Distribution of Tasks' Execution Time of Each Job Obtained Via Profiling
Interference and Input Blocks Information
Assigned Tasks to Machine
Interference

Figure 1: Overall flow of our proposed LIPTA approach

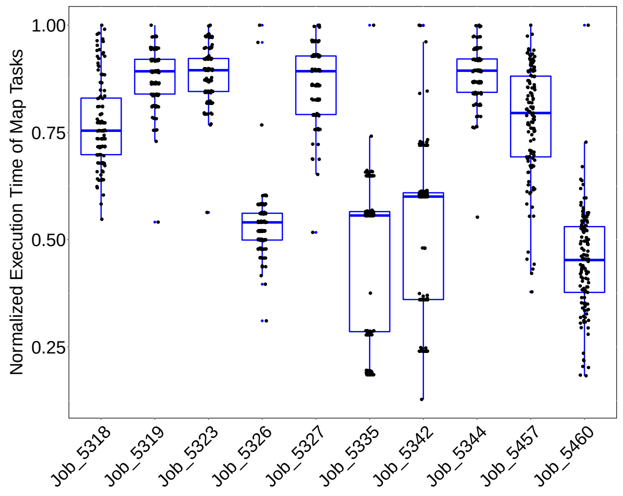

Figure 2: Percentiles (min, 25%, 50%, 75%, max) of execution time of tasks for 10 jobs from Hadoop workloads log [31]

In (1), ρ_{ij} is the correlation coefficient between assets X_i and X_j. The mean of portfolio (μ_Y) is the sum of the mean of N assets. Assuming that there is no correlation between assets ($\rho_{ij} = 0$), the expression under radical sign can be simplified as $\sum_{i=1}^{N} \sigma_i^2$. Finally, it is obvious that $\sqrt{\sum_{i=1}^{N} \sigma_i^2} \leq \sum_{i=1}^{N} \sigma_i$, and hence, the variation (risk) of portfolio would be less than or equal to all the N assets. In our study, we consider the tasks of each job as assets and each machine as a portfolio. we aim to leverage portfolio effect to assign the tasks of MapReduce jobs to machines, reducing the total runtime of machines and consequently, reducing the total cost. In the current study, we suppose that there is no correlation between jobs since there was no information about it in the logs. It is also a valid assumption to consider no correlation between jobs when they are processing independent data. However, in future works, we are going to use approaches such as [6] for determining the amount of interference between jobs that are processing the same data but for different applications.

3 LOCALITY AND INTERFERENCE AWARE PORTFOLIO-BASED TASK ASSIGNMENT

Before presenting our proposed approach, we describe all the parameters used in the rest of the paper in Table 1.

The pseudo-code of LIPTA is presented in Algorithm 1. Earlier we mentioned that LIPTA uses statistics of the execution time of tasks provided by logs to obtain the mean and standard deviation. But, for tasks of new jobs (whose execution times are not known) or new VM configurations (where the execution time profiles can vary), LIPTA can use dynamic profiling (that can constantly update execution time statistics) to address the limitation of static profiling.

The LIPTA does not assign all the tasks at once, but divides them into several subsets and assign tasks of each subset in one iteration. Assigning subsets instead of whole tasks, LIPTA is able to adapt itself to VM interference fluctuation. The amount of interference and its impact on the performance of VMs fully depends on the co-located VMs and alters by any change in them such as change of VM workload [18].We consider the effect of VM interference in the LIPTA algorithm through *IF* parameter.

Waiting for one subset of tasks to finish completely before assigning the next one, imposes a delay. To avoid it, we can start assignment of the next subset after the first machine finished execution of all its tasks from the previous subset. Please note that this mechanism is not employed in current version of LIPTA and other approaches in experiments.

Effect of doing the task assignment in several iterations on the MPT might be of question since the original theory was proposed for single decision period. Several studies [8, 11] have analyzed that how single-period problem should be modified if the true problem is multi-period such as our task assignment problem. They have found that considering reasonable assumptions, the case of multi-period problem can be solved as a sequence of single-period ones. However, the final optimum portfolio would be different from the case of only one period of decision making [7]. Since our LIPTA approach is not searching for the optimum portfolio, we can use MPT in our task assignment iterations.

After dividing the tasks into subsets of tasks (line 1), LIPTA proceeds with each subset. First, LIPTA uses mean time of each task to estimate the total execution time of tasks in the subset. Dividing the result by number of machines, LIPTA will have an estimation of the execution time of each machine (lines 3 to 5). This estimation helps LIPTA to assign the tasks to machines to balance the execution time of machines against each other. It also avoids some machines act as stragglers and waste the time and resources of others. Next, LIPTA sorts the tasks based on their standard deviation of execution time in descending order (line 6). It helps the tasks with higher standard deviation being next to each other, and hence, LIPTA can assign them to the same machine to reduce their total standard deviation leveraging portfolio effect.

After estimating the expected execution time of machines and sorting the tasks, it is time to assign the tasks to machines. For each machine, LIPTA starts from the head of the subset, where tasks with higher standard deviation are there, and assigns tasks one by one to the machine. At this part, for the sake of locality, it only assigns those tasks that the machine has a replica of their input block. Later, if LIPTA decides to assign a task to a machine that has not a replica of input data block of that task, it increases the execution time of task by *LF* parameter to consider the effect of data locality. After assigning each task, LIPTA calculates the runtime of the machine using (1). If the runtime of the machine is more than $EVMT * (1 + \beta)$, it removes the last task and continues examining the rest of them (lines 8 to 13). We add β% to *EVMT* because earlier we saw that the *EVMT* is an estimated value, and hence, we assume that the runtime of machine can be β% less or more than the *EVMT*.

In the previous step, only the tasks that the machine has a replica of their input block are assigned to it. However, it is possible that the machine still has capacity for more tasks. If the *estimated* runtime of current tasks assigned to machine is less than $EVMT * (1 - \beta)$ (line 14), LIPTA tries to assign more tasks to machine similar to the previous step, but this time without considering locality (lines 15 to 19).

Table 1: List of the parameters used in the paper

Parameter	Description
IF	Interference Factor
LF	Locality Factor
VMS	Set of virtual machines
TS	Set of tasks
μ_T	Mean of task's execution time
σ_T	Standard deviation of task's execution time
RS_T	Set of machines that have replica of task's input data
ID_{VM}	ID of VM
TA_{VM}	Set of tasks assigned to VM
RT_{VM}	runtime of VM
ST	Subset of tasks
EVMT	Estimated VM Execution Time

Algorithm 1 Locality and Interference Aware Portfolio-based Task Assignment (LIPTA)

```
1:  Divide TS into M subsets
2:  for each subset of tasks $ST \in M$ do
3:      for each task $T \in ST$ do
4:          $mean = mean + \mu_T$
5:      $EVMT = mean/size(VMS)$
6:      Sort tasks of $ST_i$ in descending order based on $\sigma_T$
7:      for each virtual machine $VM \in VMS$ do
8:          for each unassigned task $T \in ST$ do
9:              if $ID_{VM} \in RS_T$ then
10:                 Temporarily assign $T$ to $VM$
11:                 $RT_{VM} = (\sum_{T \in TA_{VM}} \mu_T + \sqrt{\sum_{T \in TA_{VM}} \sigma_T^2}) * IF_{VM}$
12:                 if $RT_{VM} > EVMT * (1 + \beta)$ then
13:                     Remove $T$ from $VM$
14:         if $RT_{VM} < EVMT * (1 - \beta)$ then
15:             for each unassigned task $T \in ST$ do
16:                 Temporarily assign $T$ to $VM$
17:                 $RT_{VM} = (\sum_{T \in TA_{VM}} \mu_T * LF_{VM,T} + \sqrt{\sum_{T \in TA_{VM}} \sigma_T^2 * LF_{VM,T}^2}) * IF_{VM}$
18:                 if $RT_{VM} > EVMT * (1 + \beta)$ then
19:                     Remove $T$ from $VM$
20:     for each unassigned task $T \in ST$ do
21:         Assign them to machines with least estimated
            runtime in a round-robin fashion.
```

Lines 8 to 19 are repeated for each machine. Finally, if there are still unassigned tasks, LIPTA assigns them to the machines with the least estimated runtime in a round-robin fashion (lines 20 to 21). These tasks might have remained unassigned as a result of EVMT underestimation (total runtime estimated for machines being less than total required time for executing tasks).

If we consider the number of tasks as n, number of subsets of tasks as m, number of tasks in each subset as (n/m), and number of machines as k, the time complexity of LIPTA can be analyzed as follow: *for* loop in line 2 repeats m times. Among *for* loops in lines 3, 7, and 20, obviously the time complexity of loop in line 7 is more than others since it traverses both machines and tasks while the two others only traverse the tasks. Loops of lines 8 and 15 are within *for* loop of line 7. They are executing sequentially and their time complexity is the same ($O(n/m)$). We can conclude that *for* loops of lines 2, 7 and either 8 or 15 determine the overall time complexity of LIPTA, which is $O(m \times k \times (n/m)) = O(k \times n)$.

4 EVALUATION

In this section, we first describe the experimental setup, workloads, and the comparison methods used in our experiments. After the experimental setup description, we report the results of evaluating LIPTA against other methods for different scenarios.

4.1 Experimental Setup

We have used seven workloads from Hadoop log dataset [31] in our experiments. Each workload represents log of jobs for one month. However, there is a slight difference between workloads and logs since we have refined logs and removed some tasks or jobs such as failed ones. Each workload consists of ten's of MapReduce jobs submitted to a cluster with 64 nodes. Since the original log consists of 64 nodes, data such as ID of machines that have a replica of input data of Map tasks is only available for those 64 machines. Hence, we have also considered 64 machines in our experiments. A brief description of workloads used in our experiments is presented in Table 2. For each task of a job, there is various information such as task type (Map or Reduce), job ID, task ID, task execution time, and ID of machines that have a copy of input block of data for Map tasks. Note that for most Map tasks in the logs there are three machines that have the copy of input block (i.e., the replication parameter had been set as three in HDFS settings). Since there is no information about the number of slots of each machine, we have supposed that each machine had one slot. We evaluated the performance of LIPTA against three other methods explained as follow:

- *Locality Aware (LA)*. This algorithm estimates the runtime of machines before assigning tasks. After that, it assigns the tasks to machines considering the data locality, but ignores the VM interference [45].
- *Interference Aware (IA)*. Same as LA estimates the runtime. But this algorithm ignores data locality and only pays attention to VM interference [41].
- *Interference and Locality Aware (ILA)*. Considers both locality and interference. The locality has higher priority than interference in this algorithm. This algorithm is closest to LIPTA [3].

Please note that all the methods assign the tasks periodically, and the tasks subsets are exactly the same as LIPTA regarding size and tasks of each subset. Furthermore, LA, IA, and ILA use exactly the same mechanism as LIPTA for estimating the initial runtime of machines (lines 3 to 5 of Algorithm 1). However, none of them employs the portfolio effect.

For including the impact of data locality, we set the value of 1.2 for the *LF* parameter in the experiments. In real-world clusters, the value of *LF* is determined by the replication pattern of data blocks among computing nodes. HDFS file system used in Hadoop distributes the data among machines in the form of data blocks. To increase the reliability of the cluster, HDFS creates several replicas (default value is 3) of a data block on different machines. If a task is placed on a machine that has not a replica of its input block,

Table 2: Brief description of workloads

Workload #	Corresponding log in [31]	No. of jobs	No. of tasks
1	2010-05	695	393075
2	2010-06	328	347455
3	2010-07	575	425271
4	2010-12	108	228150
5	2011-07	1261	276711
6	2012-03	1622	927361
7	2012-08	909	302431

then we increase the execution time of task on that machine by *LF* factor. Later in section 4.2.1, we study the impact of *LF* on the performance of our approach and other methods.

We also have considered the value of 1.2 for the *IF* parameter. We randomly assign this value to around 10% of machines in each task assignment iteration. *IF* of 1.2 for a machine means that the execution time of the tasks on this machine would take 1.2 longer than what is in the workload logs. The source of *IF* is the interference among VMs in the cloud and is determined by the significance of this interference. VMs co-located on the same physical machine compete for shared resources such as Disk I/O or cache, and hence, their performance might degrade and consequently, the execution time of tasks would be elongated. We have considered the VM interference in our study via *IF* parameter. We also study the sensitivity of different approaches to this factor parameter in section 4.2.1.

Finally, we have considered $\beta = 0.05$ in the experiments. The value of β should be determined empirically. Since the LIPTA algorithm is executed periodically, one can change it per iteration to find the best value for it.

In summary, *LF* and *IF* depend on the system and are determined by replication of data blocks and VM interference, respectively. Since we did not have the exact values of them in the workload logs, we assume some values for them in our experiments and evaluate the sensitivity of LIPTA to them. But in reality, the user cannot determine their values. The value of β, however, should be determined by the user and can be tuned empirically.

4.2 Experimental Results

Before reporting the results, we should note that we have repeated each set of experiments 10 times and reported the average of them.

The overall runtime of the cluster for processing workloads, using different algorithms, is presented in Figure 3. The runtime of the cluster is considered as the maximum runtime of all the machines because while the last machine is working, we should pay for the whole cluster (the job and its results are distributed among all the machines in the cluster). As can be seen in Figure 3, LIPTA outperforms all other methods. The average improvement of runtime compared with LA, IA, and ILA is 26.7%, 13.52%, and 19.7% respectively. The maximum improvement compared with LA, IA, and ILA happens in workload 1, workload 5, and workload 1 where LIPTA reduces runtime by 46.7%, 19.3%, and 35%, respectively. While we expect ILA to perform better than IA, we see that in several workloads (1, 3, 6, and 7) it yields worse results than IA. The reason is that ILA considers a higher priority for locality than interference. In those workloads, the effect of VM interference is more significant than locality, and hence, IA gives better results. Note that we inject interference to VMs randomly as we described earlier in section 4.1

To further analyze the performance of different applications, in the following we present more details. In Figure 4 we report the number of tasks that are assigned to machines that have a replica of input data of task (i.e., tasks that are assigned considering locality). As expected, IA has the least amount of task locality and on average over seven workloads, only assigns 3.9% of tasks to machines that have a replica of input data of task. on the other hand, LA and ILA excessively consider task locality and assign 88.6% and 86.2% of tasks to local machines, respectively. Task locality of ILA is slightly lower than LA because ILA pays attention to interference too. LIPTA, however, stands between them and tries to find a trade-off between locality and interference. Its task locality is 67.6% on average.

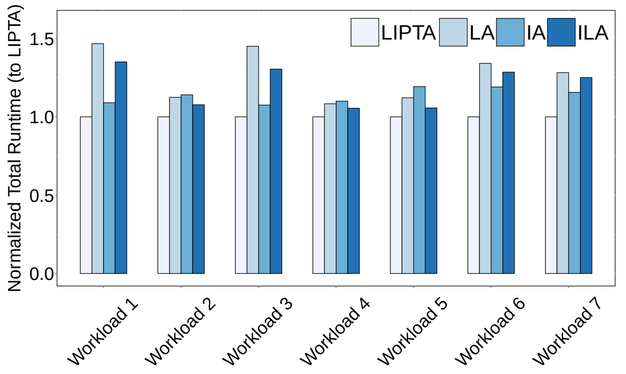

Figure 3: Normalized total runtime of workloads (to LIPTA)

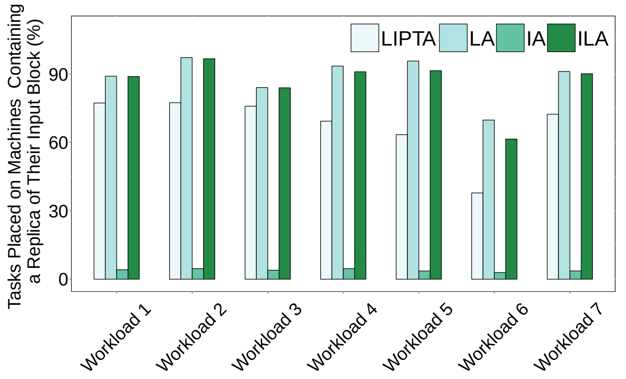

Figure 4: Percentage of locally assigned tasks

We illustrate the impact of different task assignment policies on the variation of machines' runtime in Figure 5 for one workload and then give more results in Table 3. In Figure 5 we can see the runtime of all the 64 machines normalized to the maximum value for workload 1 (for each method, the maximum value of that method is considered). as can be seen, LIPTA demonstrates the least amount of variation and all the machines have almost even runtime. However, other methods suffer significant fluctuation of runtime that elongates the completion of workload. In Table 3, the runtime variation of machines under different methods for all the workloads is presented. The reported variation is calculated using (2), where $\sigma_{machines\_runtime}$ is the standard deviation of runtime of machines and the $\mu_{machines\_runtime}$ is the mean of it.

$$Variation_{machines\_runtime} = \frac{\sigma_{machines\_runtime}}{\mu_{machines\_runtime}} * 100 \qquad (2)$$

4.2.1 Sensitivity Analysis. In the following, we analyze the sensitivity of LIPTA and other methods to the locality of data and the interference among VMs.

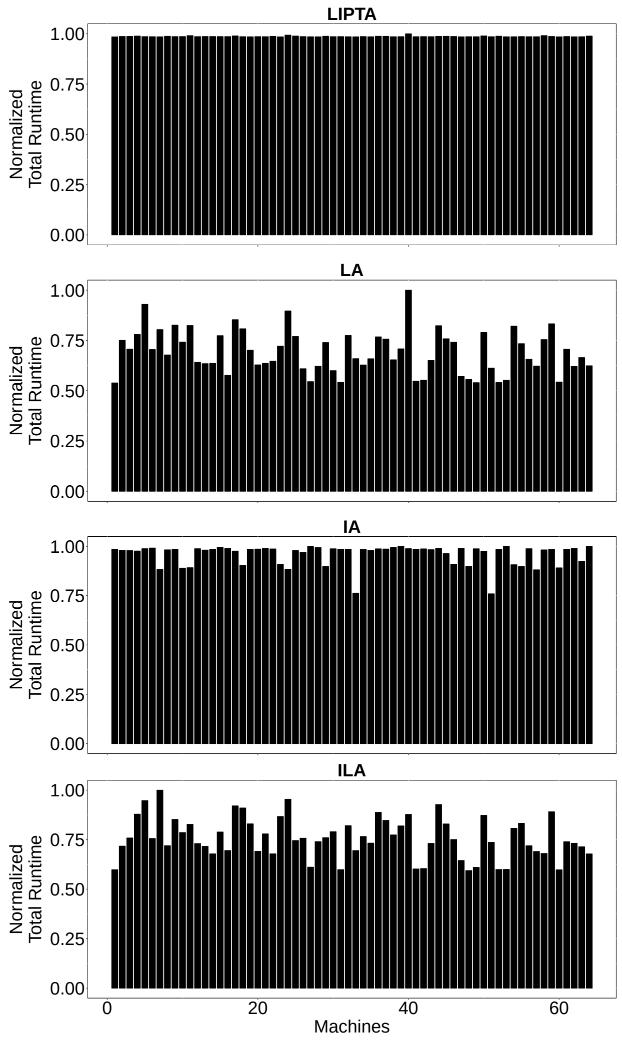

Figure 5: Normalized runtime of machines for workload 1 under different task assignment methods

Table 3: Runtime variation of machines for different workloads

	LIPTA	LA	IA	ILA
Workload 1	0.23%	15.29%	5.37%	13.43%
Workload 2	0.74%	7.76%	2.24%	3.14%
Workload 3	1.21%	17.52%	3.21%	11.28%
Workload 4	1.33%	15.69%	5.34%	10.91%
Workload 5	0.29%	14.65%	2.62%	4.26%
Workload 6	0.37%	44.62%	6.94%	37.16%
Workload 7	0.07%	20.81%	5.69%	18.50%

Locality. To evaluate the sensitivity of different algorithms to data locality, we change the value of *LF* from 1.2 to 2.0 by 0.2 steps. The results for workload 7 are presented in Figure 6. Increasing the value of *LF*, which means increasing the overhead of case when the task is not assigned to a machine that contains a replica of its input data block, worsens the performance of IA algorithm which is data locality oblivious. Since IA does not consider the data locality when assigning the tasks (see Figure 4), increasing the value of *LF* means that data locality dominates VM interference, and hence, IA shows poor performance. On the other hand, both LA and ILA show performance improvement since they either only pay attention to data locality (LA) or consider a high priority for it (ILA). LIPTA still presents acceptable performance; however, we see that the performance gap between LIPTA and ILA becomes slightly smaller. It shows that when the impact of data locality (*LF*) is too high, it can even dominate the portfolio effect used by LIPTA and give better results. However, existence of such high *LF* is doubtful.

Interference. In this section, we change the value of *IF* from 1.2 to 2.0 by 0.2 steps to see how algorithms react to this factor. The results (again for workload 7) is depicted in Figure 7. Unlike the previous section, we see that increasing the *IF* cause IA to show better performance than LA and ILA. Here we see that the performance gap between LIPTA and IA becomes smaller for high *IF* values. It implicates that in clouds with high VM interference rate, IA may outperform LIPTA. However, again the existence of such cloud with such high VM interference rate is something that needs more investigation.

Another factor that we wanted to analyze here but we couldn't due to limited information, is the impact of the standard deviation of tasks' execution time on the performance of different algorithms. The normal distribution of tasks' execution time originates from the Hadoop logs dataset, and hence, we couldn't change it. Study of this factor can reveal how much improvement LIPTA can gain over other algorithms when the execution time variation of tasks is higher than current values.

To conclude this section, we can say while ILA or IA can improve the performance when either *LF* or *IF* increases dramatically, in real-world scenarios where 1) both *LF* and *IF* increase concurrently and 2) the execution time variation of tasks is on the increase, LIPTA approach gives the best results.

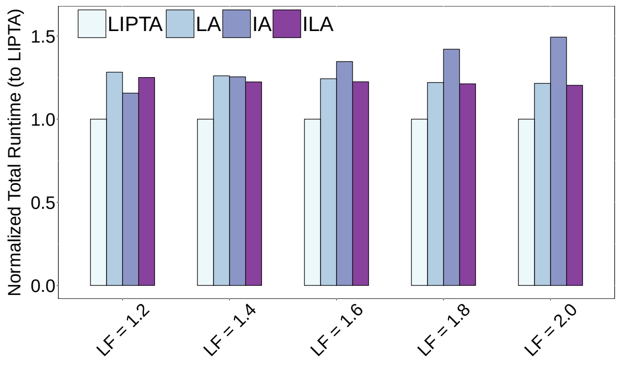

Figure 6: Analyzing the impact of Locality Factor (*LF*) on the performance of algorithms

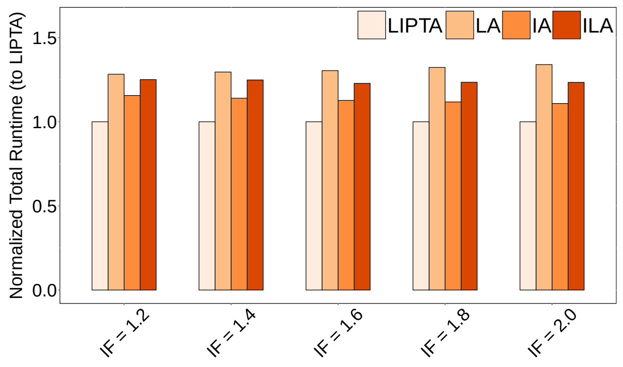

Figure 7: Analyzing the impact of Interference Factor (*IF*) on the performance of algorithms

5 RELATED WORK

A large body of research has studied the impact of data locality [9, 16, 24, 30, 34, 39, 42, 45, 46], VM interference [14, 37, 40, 41, 43, 44], or both of them [3, 10] on task assignment of big data jobs. In this section, we overview the most related ones to our work.

Data Locality. Delay scheduling approach [42] highlights the conflict between fairness in scheduling and data locality that happens for fair scheduler of Hadoop. To address this conflict, it delays the scheduling of tasks that the available machines have not a replica of their input data and schedules the other tasks behind them in the queue. The proposed approach in [45] tackles the data locality issue in heterogeneous environments. After detecting a node with spare capacity, this method first tries to schedule the tasks that their input data is stored on that node. If it cannot find such tasks, it will select a task whose input data is closest to the node and transfers the input data of that task to the node on the fly. vLocality [24] proposes a solution for data locality in virtualized environments. It consists of a storage architecture to mitigate the contention of shared disk, and a scheduling algorithm which prioritizes VMs co-located on the same physical machine. The main objective of all the aforementioned approaches is to place the tasks as close as possible to the data, but do not consider other parameters such as data skew and interference among the VMs.

VM interference. Studying the MapReduce framework, [41] finds that the integration of computational-intensive operations such as map/reduce and I/O-intensive ones such as shuffle in MapReduce seriously affects the performance of applications and degrades and the efficiency of the system especially in the cloud where the applications are deployed on VMs co-located on physical machines.Then, to address this contention, they propose a new interference-aware scheduling algorithm that assigns the MapReduce jobs to VMs. Considering the interference between big data batch jobs and interactive workloads in virtualized environments, proposed approach in [44] uses performance models to assign Hadoop tasks to the servers to improve their throughput. InSTechAH [40] also tries to address the challenge of co-location of big data batch jobs and service workloads in private cloud and improve the resource utilization by designing a multi-layer node model to reduce interference and a resource scheduling model that uses prediction based scheduling. The proposed approaches in this category ignore the importance of data locality and mostly try to improve the performance by mixing different workloads. They also skip data skew and its impact on the runtime and resource usage of applications.

Locality and Interference. FlexSlot [10] tries to mitigate the data skew in Map tasks by resizing the slots rather than task assignment. FlexSlot first identifies the stragglers, tasks that are slower than average, by two introduced parameters *progress rate and progress score* and then increases the resources of their corresponding slots to improve their performances. It also uses the speculative execution to further mitigate data skew. While it can improve the performance of applications, resource overhead and extra monetary cost of concurrent execution of several instances of one task render FlexSlot too expensive. Furthermore, when FlexSlot decides to resize the slot resources due to VM interference, it does not readjust it after the VM interference is resolved, which can lead to resource underutilization. ILA approach [3] presents a task scheduling approach to mitigate VM interference while maintaining task data locality for MapReduce applications.The ILA employs a task performance prediction model to tackle the interference and improves data locality via adaptive delay scheduling.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented Locality and Interference Aware Portfolio-based Task Assignment (LIPTA) algorithm to tackle the Map tasks' execution time variation in MapReduce applications, which is caused by data skew and exacerbated by VM interference. LIPTA leverages MPT to reduce the execution time of tasks assigned to each VM. We demonstrated how our approach can improve execution time, and consequently reduce cost of VMs, compared with several methods.

Since the original cluster that logs are created from jobs running on it is homogeneous, we also have considered a homogeneous VM cluster in our experiments. Hence, we supposed that the mean and standard deviation of tasks of a job is same for all the machines. For heterogeneous clusters, we need to estimate the normal distribution of tasks for each configuration separately. Moreover, it is worth trying to update the normal distribution of tasks for each machine after each task assignment iteration and assign the tasks in the next iteration using this updated information. Both these improvements can be obtained via dynamic profiling. Considering the correlation of tasks executing concurrently when there is more than one slot in each machine, or when the data blocks are shared among tasks of different jobs, is another direction for future work. Currently, we divide the tasks randomly among subsets, however, using clustering algorithms can help to categorize them wisely and form better subsets that can lead to more runtime improvement.

REFERENCES

[1] [n. d.]. Amazon EMR. https://aws.amazon.com/emr/. ([n. d.]). Acs: 2016-07-12.
[2] Hanieh Alipour, Yan Liu, Abdelwahab Hamou-Lhadj, and Ian Gorton. 2016. Model driven performance simulation of cloud provisioned Hadoop MapReduce applications. In *IEEE/ACM 8th International Workshop on Modeling in Software Engineering (MiSE)*. 48–54.
[3] Xiangping Bu, Jia Rao, and Cheng-zhong Xu. 2013. Interference and Locality-aware Task Scheduling for MapReduce Applications in Virtual Clusters. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing (HPDC)*. 227–238.
[4] Qi Chen, Jinyu Yao, and Zhen Xiao. 2015. Libra: Lightweight data skew mitigation in mapreduce. *IEEE Transactions on Parallel and Distributed Systems* 26, 9 (2015),

2520–2533.

[5] Emilio Coppa and Irene Finocchi. 2015. On Data Skewness, Stragglers, and MapReduce Progress Indicators. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC)*. 139–152.

[6] Christina Delimitrou and Christos Kozyrakis. 2013. The netflix challenge: Datacenter edition. *IEEE Computer Architecture Letters* 12, 1 (2013), 29–32.

[7] Edwin J Elton and Martin J Gruber. 1997. Modern portfolio theory, 1950 to date. *Journal of Banking & Finance* 21, 11 (1997), 1743–1759.

[8] Eugene F Fama. 1970. Multiperiod consumption-investment decisions. *The American Economic Review* (1970), 163–174.

[9] Yifeng Geng, Shimin Chen, YongWei Wu, Ryan Wu, Guangwen Yang, and Weimin Zheng. 2011. Location-aware mapreduce in virtual cloud. In *International Conference on Parallel Processing (ICPP)*. 275–284.

[10] Yanfei Guo, Jia Rao, Changjun Jiang, and Xiaobo Zhou. 2017. Moving Hadoop into the Cloud with Flexible Slot Management and Speculative Execution. *IEEE Transactions on Parallel and Distributed systems* 28, 3 (2017), 798–812.

[11] Nils H Hakansson. 1974. Convergence to isoelastic utility and policy in multi-period portfolio choice. *Journal of Financial Economics* 1, 3 (1974), 201–224.

[12] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. 2011. Starfish: A Self-tuning System for Big Data Analytics.. In *Cidr*, Vol. 11. 261–272.

[13] Tzu-Chi Huang, Kuo-Chih Chu, Guo-Hao Huang, Yan-Chen Shen, and Ce-Kuen Shieh. 2017. Smart Partitioning Mechanism for Dealing with Intermediate Data Skew in Reduce Task on Cloud Computing. In *IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*. 819–826.

[14] Zhe Huang, Bharath Balasubramanian, Michael Wang, Tian Lan, Mung Chiang, and Danny HK Tsang. 2016. RUSH: A RobUst ScHeduler to Manage Uncertain Completion-Times in Shared Clouds. In *IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. 242–251.

[15] Inkwon Hwang and Massoud Pedram. 2012. Portfolio theory-based resource assignment in a cloud computing system. In *IEEE International Conference on Cloud Computing (CLOUD)*. 582–589.

[16] Yu-Chon Kao and Ya-Shu Chen. 2016. Data-locality-aware mapreduce real-time scheduling framework. *Journal of Systems and Software* 112 (2016), 65–77.

[17] Mukhtaj Khan, Yong Jin, Maozhen Li, Yang Xiang, and Changjun Jiang. 2016. Hadoop performance modeling for job estimation and resource provisioning. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (2016), 441–454.

[18] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. 2007. An analysis of performance interference effects in virtual environments. In *IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. 200–209.

[19] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. 2012. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 25–36.

[20] Min Li, Dinesh Subhraveti, Ali R Butt, Aleksandr Khasymski, and Prasenjit Sarkar. 2012. CAM: a topology aware minimum cost flow based resource manager for MapReduce applications in the cloud. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing (HPDC)*. 211–222.

[21] Norman Lim, Shikharesh Majumdar, and Peter Ashwood-Smith. 2017. MRCP-RM: A Technique for Resource Allocation and Scheduling of MapReduce Jobs with Deadlines. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (2017), 1375–1389.

[22] Jia-Chun Lin, Ming-Chang Lee, and Ramin Yahyapour. 2014. Scheduling MapReduce tasks on virtual MapReduce clusters from a tenant's perspective. In *IEEE International Conference on Big Data (Big Data)*. 141–146.

[23] Zhihong Liu, Qi Zhang, Raouf Boutaba, Yaping Liu, and Baosheng Wang. 2016. Optima: on-line partitioning skew mitigation for MapReduce with resource adjustment. *Journal of Network and Systems Management* 24, 4 (2016), 859–883.

[24] Xiaoqiang Ma, Xiaoyi Fan, Jiangchuan Liu, Hongbo Jiang, and Kai Peng. 2017. vLocality: Revisiting Data Locality for MapReduce in Virtualized Clouds. *IEEE Network* 31, 1 (2017), 28–35.

[25] Harry M Markowitz. 1968. *Portfolio selection: efficient diversification of investments*. Vol. 16. Yale university press.

[26] Lena Mashayekhy, Mahyar Movahed Nejad, Daniel Grosu, Quan Zhang, and Weisong Shi. 2015. Energy-aware scheduling of mapreduce jobs for big data applications. *IEEE Transactions on Parallel and Distributed Systems* 26, 10 (2015), 2720–2733.

[27] Seyed Morteza Nabavinejad and Maziar Goudarzi. 2016. Energy efficiency in cloud-Based MapReduce applications through better performance estimation. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*. 1339–1344.

[28] Seyed Morteza Nabavinejad and Maziar Goudarzi. 2017. Faster MapReduce Computation on Clouds through Better Performance Estimation. *IEEE Transactions on Cloud Computing* (2017).

[29] Seyed Morteza Nabavinejad, Maziar Goudarzi, and Shirin Mozaffari. 2016. The Memory Challenge in Reduce Phase of MapReduce Applications. *IEEE Transactions on Big Data* 2, 4 (2016), 380–386.

[30] Jongse Park, Daewoo Lee, Bokyeong Kim, Jaehyuk Huh, and Seungryoul Maeng. 2012. Locality-aware dynamic VM reconfiguration on MapReduce clouds. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing (HPDC)*. 27–36.

[31] Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. 2013. Hadoop's adolescence: an analysis of Hadoop usage in scientific workloads. *Proceedings of the VLDB Endowment* 6, 10 (2013), 853–864.

[32] Andrew Rudd and Henry K Clasing. 1982. *Modern portfolio theory: The principles of investment management.*

[33] M. Goudarzi S. Nasehi, S.M. Nabavinejad. 2017. A Novel Key Partitioning Schema for Efficient Execution of MapReduce Applications. *The 19th CSI International Symposium on Computer Architecture & Digital Systems (CADS)* (2017).

[34] TP Shabeera and SD Madhu Kumar. 2015. Optimising virtual machine allocation in MapReduce cloud for improved data locality. *International Journal of Big Data Intelligence* 2, 1 (2015), 2–8.

[35] Juwei Shi, Jia Zou, Jiaheng Lu, Zhao Cao, Shiqiang Li, and Chen Wang. 2014. MRTuner: a toolkit to enable holistic optimization for mapreduce jobs. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1319–1330.

[36] Jia Wang and Xiaoping Li. 2016. Task scheduling for MapReduce in heterogeneous networks. *Cluster Computing* 19, 1 (2016), 197–210.

[37] Kewen Wang, Mohammad Maifi Hasan Khan, Nhan Nguyen, and Swapna Gokhale. 2016. Modeling interference for apache spark jobs. In *IEEE 9th International Conference on Cloud Computing (CLOUD)*. 423–431.

[38] Suzhen Wang and Haowei Zhou. 2016. The research of mapreduce load balancing based on multiple partition algorithm. In *IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC),*. 339–342.

[39] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. 2013. A throughput optimal algorithm for map task scheduling in mapreduce with data locality. *ACM SIGMETRICS Performance Evaluation Review* 40, 4 (2013), 33–42.

[40] Xueying Wang, Zhihui Lu, Jie Wu, Tong Zhao, and Patrick Hung. 2015. In STechAH: An Autoscaling Scheme for Hadoop in the Private Cloud. In *IEEE International Conference on Services Computing (SCC)*. 395–402.

[41] Y. Yuan, H. Wang, D. Wang, and J. Liu. 2013. On interference-aware provisioning for cloud-based big data processing. In *IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*. 1–6.

[42] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems (EuroSys)*. 265–278.

[43] Wei Zhang, Sundaresan Rajasekaran, Shaohua Duan, Timothy Wood, and Mingfa Zhuy. 2015. Minimizing interference and maximizing progress for Hadoop virtual machines. *ACM SIGMETRICS Performance Evaluation Review* 42, 4 (2015), 62–71.

[44] Wei Zhang, Sundaresan Rajasekaran, Timothy Wood, and Mingfa Zhu. 2014. Mimp: Deadline and interference aware scheduling of hadoop virtual machines. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 394–403.

[45] X. Zhang, Y. Feng, S. Feng, J. Fan, and Z. Ming. 2011. An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In *International Conference on Cloud and Service Computing (CSC)*. 235–242.

[46] Xiaohong Zhang, Zhiyong Zhong, Shengzhong Feng, Bibo Tu, and Jianping Fan. 2011. Improving data locality of mapreduce by scheduling in homogeneous computing environments. In *IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. 120–126.