Faster MapReduce Computation on Clouds through Better Performance Estimation

Seyed Morteza Nabavinejad, *Student Member, IEEE* and Maziar Goudarzi, *Senior Member, IEEE*

***Abstract*—Processing Big Data in cloud is on the increase. An important issue for efficient execution of Big Data processing jobs on a cloud platform is selecting the best fitting virtual machine (VM) configuration(s) among the miscellany of choices that cloud providers offer. Wise selection of VM configurations can lead to better performance, cost and energy consumption. Therefore, it is crucial to explore the available configurations and opt for the best ones that well suit each MapReduce application. Profiling the given application on all the configurations is costly, time and energy consuming. An alternative is to run the application on a subset of configurations (sample configurations) and estimate its performance on other configurations based on the obtained values by sample configurations. We show that the choice of these sample configurations highly affects accuracy of later estimations. Our Smart Configuration Selection (SCS) scheme chooses better representatives from among all configurations by once-off analysis of given performance figures of the benchmarks so as to increase the accuracy of estimations of missing values, and consequently, to more accurately choose the configuration providing the highest performance. The results show that the SCS choice of sample configurations is very close to the best choice, and can reduce estimation error to 11.58% from the original 19.72% of random configuration selection. More importantly, using SCS estimations in a makespan minimization algorithm improves the execution time by up to 36.03% compared with random sample selection.**

***Index Terms*—Cloud, Performance Estimation, Big Data Processing, MapReduce, Matrix Completion**

1 INTRODUCTION

FORECASTS such as [1] predict that the volume of digital data will increase by 300 times in 2020 compared with 2005. This significant growth further emphasizes the importance of Big Data as well as Big Data Processing. MapReduce [2], and its open source implementation Hadoop [3], are prevailing frameworks for implementing Big Data Analytics and applications. Because of inherently huge amount of data and computational requirements of Big Data applications, acquisition of large amount of computational resources is necessary. However, managing in-house clusters to respond the computational requirements is costly such that small- and middle-sized companies either cannot afford it, or find cloud-based solutions economically more attractive. Consequently increasingly more companies are moving towards the on-demand resources available in the cloud.

The cloud computing concept provides the opportunity to employ the required computational resources in the form of VMs and pay for them based on the pay-as-you-go pricing model. The cloud providers have even launched dedicated services such as Amazon Elastic MapReduce (EMR) service [4] to satisfy the growing demand of computational resources for MapReduce applications. While deploying Big Data applications on cloud platform brings fascinating opportunities, there are concerns such as cost and deadline that need to be addressed.

A large body of research [5]–[12] has tried to address the aforementioned concerns. A common challenge in these researches regarding resource allocation and scheduling for MapReduce jobs is that estimation is needed on the execution time or performance of the application on available VM configurations. The common framework used in most these works is shown in Fig. 1: the *Estimation Phase* provides an estimate of the performance, energy consumption or other metrics of applications on various available VM configurations in the cloud; then the *Resource Allocation/Scheduling Phase* chooses the best configuration(s) and the number of VM instances required to process the whole big data while adhering to the given constraints and objective. Obviously, overestimation of the VM configuration performance results in losing the deadline, and underestimation results in overbooking of resources and waste of energy and money. Thus, accurate and low-cost estimation methods are required to avoid both these cases.

The common method to address the estimation challenge is profiling. We can divide profiling approaches to complete profiling and partial profiling. In *complete profiling*, the application is executed on all the accessible configurations for a short period of time (e.g., a few minutes) to estimate performance of each configuration. If the state space of the problem is small (i.e., few number of available configurations) this approach will be suitable. Otherwise, when the number of selectable choices is high, which is usually the case with today and anticipated growth in cloud computing providers, running the application on all of them imposes large overhead. It means using a lot of resources that needs lots of money and leads to waste of energy and time. Thus, complete profiling is usually not acceptable.

In *partial profiling*, only a small subset of the state space (i.e., a few configurations) are actually profiled per new application, and then results are extended to the whole state space based on either the results of prior complete profiling of a few representative benchmarks or using mathematical methods [7], [9], [13]. In [14], the authors first run some applications on all the available configurations (Offline Training), and then, when a new application arrives, its performance is measured on two sample configurations for a short period of time, and then by a reconstruction technique called Matrix Completion [15], the values for other configurations is predicted. Consequently,

The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
E-mail: mnabavi@ce.sharif.edu, goudarzi@sharif.edu
Maziar Goudarzi is the corresponding author

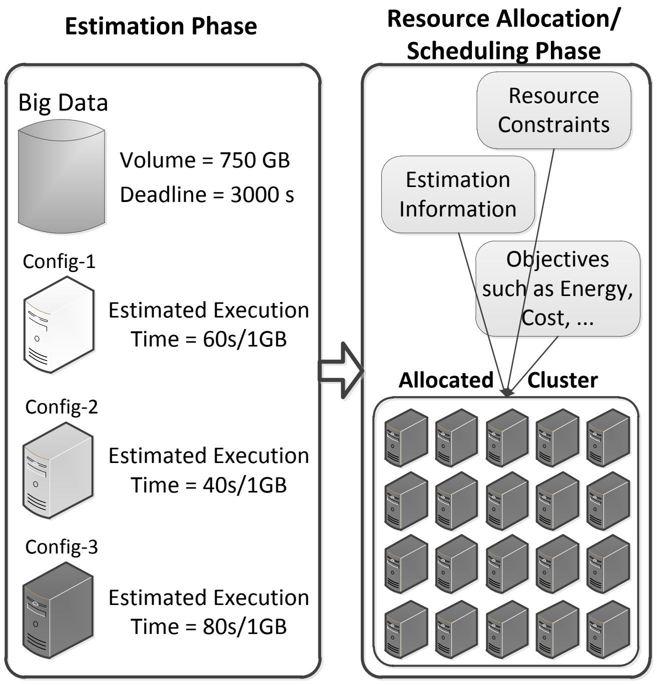

Estimation Phase
Resource Allocation/ Scheduling Phase
Big Data
Volume = 750 GB
Deadline = 3000 s
Config-1
Estimated Execution Time = 60s/1GB
Config-2
Estimated Execution Time = 40s/1GB
Config-3
Estimated Execution Time = 80s/1GB
Resource Constraints
Estimation Information
Objectives such as Energy, Cost, ...
Allocated Cluster

Fig. 1. Common framework in resource allocation/scheduling for MapReduce jobs in cloud.

the Offline Training Phase (see Fig. 2) serves several future applications, and less time and resources are used for the Usage Phase per new application. Reference [14] chooses the two sample configurations randomly. However, our results in section II show that the accuracy of predicting the missing values is widely dependent on the choice of these sample configurations.

In this paper, we propose *Smart Configuration Selection (SCS)* method to leverage the power of matrix completion by carefully choosing the sample configurations. SCS uses either Pearson correlation coefficient or Kendall rank correlation coefficient to opt for the most effective sample configurations. We apply our method on a set of VM configurations that are already offered by various cloud providers such as Amazon EC2 [16], and Microsoft Azure [17]. We show that the SCS choice of sample configurations is close to the best choice, and can reduce estimation error to 11.58% from the original 19.72% of random configuration selection. More importantly, we show that using SCS estimations in a makespan minimization algorithm, which uses matrix completion in its profiling phase to estimate the performance of applications on various configurations, improves the execution time by up to 36.03% compared with random sample selection.

This paper is an extension of our prior work published in [18]. In brief, we have added the following investigations in this paper:

- We have added more applications and VM configurations to more comprehensively verify our approach.
- We have used Kendall Rank Correlation Coefficient, in addition to Pearson Correlation Coefficient, and have compared the performance of our approach under these two variants of correlation coefficient.
- We have applied our approach to the well-known makespan minimization algorithm to investigate and demonstrate the impact of our SCS approach on the end result of scheduling and resource management techniques.
- We have analyzed the time complexity of our proposed approach to show its negligible overhead and its applicability to large-scale applications.

Our major contributions in this work are as follows:

- We investigate the impact of choice of sample configurations (known values) on the accuracy of matrix completion for estimating the performance of applications on new configurations (missing values). We show that accuracy changes dramatically by choosing different sample configurations.
- We propose Smart Configuration Selection (SCS) approach to select the sample configurations wisely. SCS uses correlation coefficient between configurations to opt for most appropriate pair of configurations as sample configurations. We demonstrate the accuracy improvement of matrix completion using SCS approach.
- Profiling phase, which uses matrix completion to estimate the performance of applications, is a prerequisite for resource management and scheduling phase. Hence, we further evaluate the influence of increased estimation accuracy by SCS on this phase using a makespan minimization algorithm. We show that the algorithm can yield less execution time for applications using SCS estimation compared with estimation obtained by random sample configuration selection.

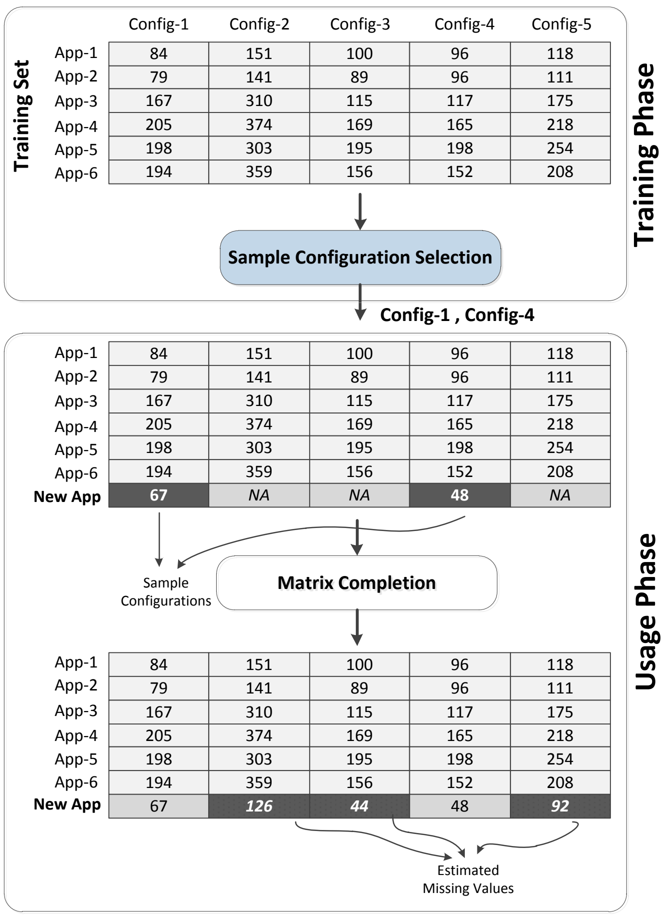
The rest of the paper is organized as follows. Section II gives an example to show the motivation beyond this work and discusses it. Section III introduces the SCS approach and its mathematical background. Section IV represents the experimental results. Related works are discussed in section V. Finally, the paper is concluded and future works are provided in section VI.

II. Motivation

In this section, we employ an example to show motivation behind our research. We ran 14 MapReduce applications from PUMA suite [19] on 13 VM configurations with different amount of resources (e.g., RAM and CPU). VM configurations are the same as some of the instances from [16] and [17]. We also used datasets from [19] as the input of MapReduce applications. The numbers in Table I indicate the execution time of each application on the various configurations (in seconds) for 3GB of data.

Before presenting the results, we depict the overall flow of experiments and define necessary concepts for pursuing the rest of paper. Fig. 2 illustrates the flow from training phase to usage phase. First, some applications are executed on all the available configurations. These applications are training set and information obtained from their execution is used for next steps. This step is offline and need to be done just once (Training Phase). Next, a subset of configurations is employed and the new application is executed on them. From now on this subset is called sample configurations. Finally, from information of training phase and executing the new application on sample configurations, the matrix completion approach estimates the performance of new application on the rest of the configurations which we call missing values. This step is done online (Usage Phase).

For showing that selecting the sample configurations randomly might lead to imprecise estimation of missing values, we conducted a set of experiments for each application in Table I. In each set, we selected one application and assumed it as the new application we want to estimate its performance on configurations, and assumed there is no prior information about it. The other applications are considered as training set that we want to predict the missing values of new application based on their information. Then, we chose a pair of configurations randomly as known samples, which the execution time of application is known on them. The seven left ones are considered as missing values. Finally, we applied matrix completion program from [20] on the matrix with missing values and observed the results. This process repeated ten times for each application and in different iterations we chose a different pair of random sample configurations. Approximation error (the difference between estimated value and actual value) of matrix completion regarding missing values for each of ten iterations in all the eight applications is demonstrated in Fig. 3. As this figure reveals, the amount of error differs significantly from one sampling pair to the other. Therefore, it can be concluded that the selected sample configurations can affect the outcome of matrix completion remarkably; as Fig. 3 shows, the approximation error of matrix completion varies from 3.5% to over 71.2% depending on the selected sample configurations. So, it is essential to choose the sample configurations wisely in order to have an accurate prediction of missing values.



Training Set

	Config-1	Config-2	Config-3	Config-4	Config-5
App-1	84	151	100	96	118
App-2	79	141	89	96	111
App-3	167	310	115	117	175
App-4	205	374	169	165	218
App-5	198	303	195	198	254
App-6	194	359	156	152	208

App-1	84	151	100	96	118
App-2	79	141	89	96	111
App-3	167	310	115	117	175
App-4	205	374	169	165	218
App-5	198	303	195	198	254
App-6	194	359	156	152	208
New App	67	*NA*	*NA*	48	*NA*

App-1	84	151	100	96	118
App-2	79	141	89	96	111
App-3	167	310	115	117	175
App-4	205	374	169	165	218
App-5	198	303	195	198	254
App-6	194	359	156	152	208
New App	67	***126***	***44***	48	***92***

Fig. 2. Overall flow of performance estimation using matrix completion.

III. Smart Configuration Selection (SCS)

In this section, we present the SCS approach. The main goal of SCS is to select the sample configurations in a way that leads to more accurate prediction of missing values in matrix completion process. First, the mathematical background of SCS, including correlation coefficient and its variants, is explained and then the approach itself is expanded.

A. Mathematical Background

1) Matrix Completion: The general problem of matrix completion can be defined as follow: we want to recover

TABLE I
Execution Time Of MapReduce Applications on Different VM Configurations

Execution Time (s)	VM Configurations												
Application	Config 1	Config 2	Config 3	Config 4	Config 5	Config 6	Config 7	Config 8	Config 9	Config 10	Config 11	Config 12	Config 13
Adjacency-List	4126	2025	1803	3668	827	1870	1236	1405	1775	2505	1591	916	861
Classification	3262	1780	1714	3389	428	1685	903	897	1800	1813	901	457	485
Grep	557	287	247	470	148	255	156	164	243	389	210	168	191
Histogram-Movies	212	121	112	214	41	117	66	72	114	131	89	105	38
Histogram-Ratings	473	282	279	468	85	278	152	148	286	292	155	91	129
Inverted-Index	815	405	359	710	121	378	208	222	378	529	278	196	109
K-Means	3567	1865	1790	3568	472	1783	945	972	1876	2043	1006	550	518
Pi-Estimator	347	178	178	342	57	180	100	98	185	174	97	58	48
Ranked-Inverted-Index	1334	687	477	993	258	538	281	356	405	1128	629	371	292
Self-Join	506	281	216	410	143	229	150	171	202	415	250	170	160
Sequence-Count	989	521	461	867	228	465	272	297	455	667	385	293	259
Tera-Sort	669	355	268	513	170	294	187	205	256	512	302	223	243
Term-Vector	792	407	362	718	178	374	213	219	357	531	277	200	221
Word-Count	757	392	341	667	118	354	198	202	343	505	266	186	103

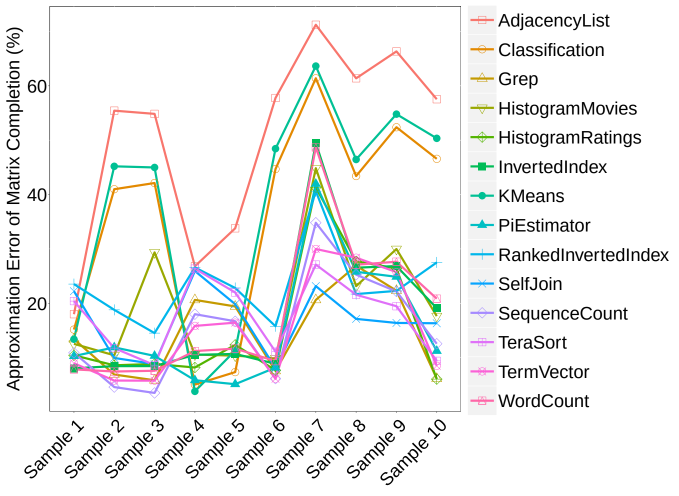


Fig. 3. Approximation Error of Matrix Completion for ten different pairs of sample configurations.

matrix M with n_1 rows and n_2 columns, but we only have m entries of M which is smaller than total entries $n_1 \times n_2$ [15].

Obviously, doing this job without having additional information is impossible. Knowing that Matrix M is low rank or approximately low rank can help recover the matrix. We say that matrix M has rank r provided that its rows or columns span an r-dimensional space. For example, In the Netflix problem, where one should recommend movies to users based on their preferences, it is believed that a few number of factors have impact on an individuals taste or preferences for movies, and hence, the matrix M of all user-ratings might be approximately low rank. Finding the similarity factors can be done using singular value decomposition.

Singular Value Decomposition or SVD is one of the methods used for dimensionality reduction and similarity identification in matrices. For example, recommendation systems use SVD to extract similarities between users and items [21]. If we feed SVD with matrix M, we will have a factorization of the form. $U \times \Sigma \times V^T$

U is the left and V is the right singular vectors matrices.

$$U_{n_1 \times r} = \begin{bmatrix} u_{11} & \dots & u_{1r} \\ u_{21} & \dots & u_{2r} \\ \vdots & \ddots & \vdots \\ u_{n_1 1} & \dots & u_{n_1 r} \end{bmatrix}, V_{n_2 \times r} = \begin{bmatrix} v_{11} & \dots & v_{1r} \\ v_{21} & \dots & v_{2r} \\ \vdots & \ddots & \vdots \\ v_{n_2 1} & \dots & v_{n_2 r} \end{bmatrix}$$

And Σ is the matrix of singular values which is a diagonal matrix. r is the rank of Matrix M [14].

$$\Sigma_{r \times r} = \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_r \end{bmatrix}$$

Each singular value σ_i and its magnitude represents a similarity concept (factors that affect users taste in Netflix problem) and the confidence in it respectively. The complexity of SVD on a $n_1 \times n_2$ matrix is $\min(n_1{}^2 \times n_2, n_2{}^2 \times n_1)$.

Now that we have matrices U, Σ and V, we use PQ-reconstruction to approximate Matrix X in which the missing values of matrix M are predicted.

$X = Q_{n_1 \times r} \times P^T{}_{r \times n_2}$, where $Q_{n_1 \times r} = U$ and $P^T{}_{r \times n_2} = \Sigma \times V^T$

Since the matrix M was incomplete, this incompleteness will propagate to X via Q and P. Different approaches are proposed to estimate the missing values in X such as convex optimization [15], gradient descent [22], and Alternating least squares minimization [23]. Using any of this approaches, we can have matrix X such that for m observed entries of M, 1) $X_{i,j} = M_{i,j}$ and 2) rank (X) is minimized. In this paper, we use the TFCOS [20] which uses convex optimization approach. Note that matrix completion cannot work properly in special cases when there is not enough information in matrix. However, discussing such cases is beyond the scope of this paper.

2) Correlation Coefficient: Correlation Coefficient indicates the linear relationship between two variables and also shows the strength and direction of this relationship. Various versions of correlation coefficient are available such as Pearson and Kendal. We use both variants in our work to evaluate and compare their performance and see which one might be more effective for selecting sample configurations.

Kendal Rank Correlation Coefficient (KRCC) [24] indicates the portion of ranks that match between two data sets. Considering two random variables X and Y and a set of their observations (x_1,y_1), (x_2,y_2), (x_n,y_n), for each pair of observations (x_i,y_i) and (x_j,y_j), the pair is called:

concordant if

$$\begin{aligned} &\forall (x_i, y_i), (x_j, y_j), i \neq j \& i, j = 1...n : \\ &(x_i > x_j \wedge y_i > y_j) \vee (x_i < x_j \wedge y_i < y_j) \end{aligned} \tag{1}$$

discordant if

$$\begin{aligned} &\forall (x_i, y_i), (x_j, y_j), i \neq j \& i, j = 1...n : \\ &(x_i > x_j \wedge y_i < y_j) \vee (x_i < x_j \wedge y_i > y_j) \end{aligned} \tag{2}$$

In other words, in concordant pair the ranks for both observations agree while in discordant pair disagree. Using (1) and (2), the KRCC is defined as (3) where denominator indicates the total number of pairs.

$$KRCC(X,Y) = \frac{\#\ concordant\ pairs - \#\ discordant\ pairs}{\frac{n(n-1)}{2}} \tag{3}$$

If the random variables are completely independent, KRCC would be near zero. Otherwise, when the ranks for both observations agree in all the pairs, KRCC equals 1 and for disagree case equals -1. Note that for the case where observations are equal (tied values), for example $x_i = x_j$ or $y_i = y_j$, the formulation would be a bit different. However, in our experiments we never faced such situation, and hence, we skip those situations. The enthusiast readers can find more about these cases in [24].

Pearson Correlation Coefficient (PCC). For two variables X and Y, the PCC is a value between -1 and +1. +1 and -1 shows that the variables are strongly related to each other. Zero value means that there is no kind of relationship between two variables. The PCC of two variables can be calculated as (4):

$$PCC(X,Y) = \frac{cov(X,Y)}{\sigma_{\mathrm{X}}\ \sigma_{\mathrm{Y}}} \tag{4}$$

Where *cov* stands for covariance and σ stands for standard deviation.

Comparing KRCC and PCC, we can conclude that while PCC considers the value of observations, KRCC relies on ranking of observations rather than their values. Later in the experimental results, we see the impact of these two different approaches.

B. SCS Approach

In this section, we first show the impact of correlation between sample configurations on accuracy of matrix completion and then expand SCS approach.

Considering Table I, we perceive that configurations 5 and 7 have strong relationship with each other (PCC (5, 7) = 0.9997, KRCC (5, 7) = 0.84615). So, what if we choose these two as sample configurations? The results are shown in Fig. 4 and compared with the average of ten random samples of Fig. 3. It shows the approximation error is too high and we can say it is an inauspicious choice. But, what is the reason? The answer is that since these two configurations are highly correlated to each other, they cannot represent the diversity of all configurations. Hence the estimation leads to poor results. From this observation, we can conclude that the two selected samples should be the best representatives of the assortment of configurations in order to obtain a precise prediction. It is an interesting question to explore how many samples should be used to best represent a given training set based on the number of configurations present in that set, but in this paper we suffice to the case of using two sample configurations. From this conclusion, we present the SCS approach which concentrates on finding dichotomy between configurations and picking the two bests that can represent it. This approach is offline and need to be applied just once in *Sample Configuration Selection* part in Fig. 2.

SCS pseudo-code is presented in Algorithm 1. SCS first calculates the PCC or KRCC (which we refer to as CC from now on) (5) for each two configurations (columns in Table I) where N stands for the set of all configurations (lines 3 to 5). In fact, in SCS approach each configuration stands for random variables and execution time of applications on configurations stand for observations.

$$\forall i, j \in N, CC(i,j) \tag{5}$$

Then, it calculates the total amount of correlation between each configuration and other ones as (6) (lines 6 to 10). Note that since the CC might be negative, we use absolute value of it. TCC stands for Total Correlation Coefficient.

$$\forall i \in N, TCC_{\mathrm{i}} = \sum_{j=1}^{n(i \neq j)} |CC(i,j)| \tag{6}$$

After that, the first configuration that will be chosen is the one with least amount of TCC (7) because the one with least TCC shows the most diversity with other configurations (line 11).

$$Config1 = \{i | TCC_{\mathrm{i}} = Min(TCC_{\mathrm{j}}), j = 1...n\} \tag{7}$$

Finally, for second configuration, the one should be chosen that has the least CC with first selected configuration (8). In this way, the two selected configurations have: 1) minimum correlation so are different from each other and 2) at least one of them has the least possible correlation with other configurations and can represent diversity among them (line 12). For choosing more than two sample configurations, we can continue the procedure by choosing the configuration that has the least amount of correlation with previously chosen configurations. For example, the third sample configuration is the one that has minimum correlation with the two previously selected configurations.

$$Config2 = \{i | CC_{\mathrm{i}} = Min(CC(i, config1), i = 1...n\} \tag{8}$$

If consider the number of VM configurations as *n* and number of applications in training set as *m*, then the time complexity of SCS approach can be analyzed as follow: the time complexity of PCC is $O(m)$ and time complexity of KRCC is $O(m \log m)$ (Note that time complexity of conventional method for KRCC is $O(m^2)$, but a more sophisticated method based on merge sort can compute KRCC in $O(m \log m)$ [25]). Hence, the time complexity of (5) would be $O(n^2 m)$ if PCC is used or $O(n^2 m \log m)$ if KRCC is used. Note that PCC or KRCC should be calculated for each pair of configurations and number of observations for each pair of configurations is *m*. Time complexity of (6) is $O(n^2)$ and time complexity of (7) and (8) is $O(n)$. Since (5), (6), (7), and (8) are executed in serial, the time complexity of SCS would be $O(n^2 m)$ or $O(n^2 m \log m)$, if PCC or KRCC are used respectively.

Algorithm 1 Smart Configuration Selection

```
1:  N: Set of Configurations
2:  TCC: total correlation coefficient of each configuration
    with other ones (initialized as zeros)
3:  for each $i \& j \in N$ do
4:      calculate the Correlation Coefficient of i & j ($CC_{ij}$)
5:  end for
6:  for $i \in N$ do
7:      for $j \in N$ do
8:          $TCC_i$ += $CC_{ij}$
9:      end for
10: end for
11: sample configuration 1 = configuration with minimum
    TCC
12: sample configuration 2 = configuration that has minimum
    CC with sample configuration 1
```

SCS method is simple, yet very effective because it can quickly distinguish the two most proper configurations for

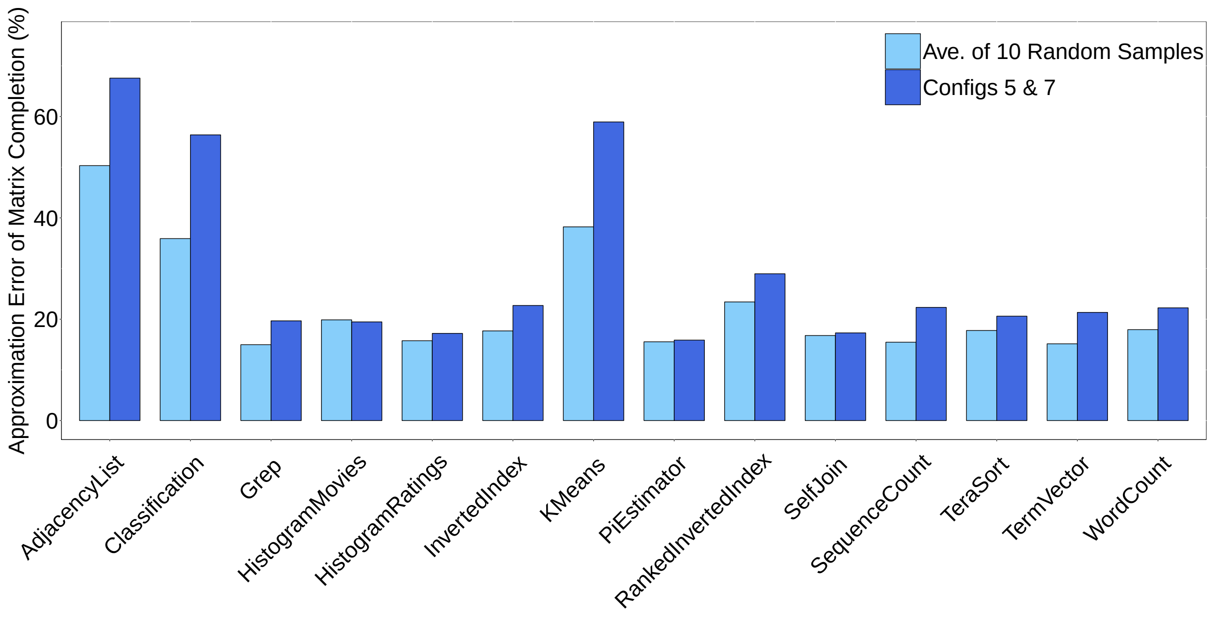


Fig. 4. Impact of choosing highly correlated configurations on approximation error.

sampling by doing a few calculations. The selected configurations are different from each other regarding CC and they also can represent the diversity among whole set of available configurations.

IV. Experimental Results

The state space of problem in our example (Table I) is small enough to be explored completely; so we have calculated the amount of approximation error of matrix completion for all the possible pairs of sample configurations for all applications. Fig. 5 illustrates the results. As can be seen, for all the applications there is a wide gap between minimum and maximum error; note that the horizontal lines of each box in Fig. 5 show the 25^{th}, 50^{th}, and 75^{th} percentile of dots. The minimum, maximum, and average error for all the applications is presented in Table II. These observations further prove the necessity for careful selection of sample configurations. In upcoming subsections, we first introduce the MapReduce applications we have used in our experiments and present the specifications of VMs in experiments setup subsection. After that, we illustrate the effectiveness of SCS for improving estimation accuracy of applications performance on different configurations. Then, we further explore the problem and depict the impact of estimation accuracy on meeting/missing applications deadline and resource utilization.

A. Experiments Setup

We have used 14 MapReduce applications and 13 types of VM configurations in our experiments to evaluate our SCS approach. We have used Hadoop version 1.2.1 on Ubuntu 12.04 as the framework of experiments. We have used two nodes per experiments to extract the performance of applications and fill Table I. Extending the number of VMs wont affect our approach but increases its resource overhead. In the following, we first introduce the applications and then present the VM configurations.

TABLE II
Details of Approximation Error (%) for Applications

Application	Min Error	Max Error	Average Error
Adjacency-List	16.1118	71.23322	47.14514
Classification	2.923779	61.39729	31.70884
Grep	4.819775	26.93327	14.04154
Histogram-Movies	8.762522	44.85916	16.85232
Histogram-Ratings	5.461222	43.36961	14.22351
Inverted-Index	5.384964	49.48646	13.90891
K-Means	3.364519	63.63151	34.30891
Pi-Estimator	3.999684	49.37178	12.1857
Ranked-Inverted-Index	7.596326	40.62434	21.11223
Self-Join	5.564344	31.89142	14.64871
Sequence-Count	3.506523	34.81392	13.63431
Tera-Sort	4.59812	32.4036	15.3514
Term-Vector	4.777848	29.99559	13.27253
Word-Count	5.519747	48.68563	13.73046

MapReduce Applications. We have used the applications in PUMA benchmark suit to evaluate our approach. This benchmark suit has been used in a large body of papers related to big data and MapReduce [26]–[29]. The applications are various in several aspects such as amount of generated intermediate data, the resource demand of map and reduce tasks, the number of tasks, the share of Map, Reduce, and shuffle phases in total execution time, etc. So, they can present variety of MapReduce applications..The following are name and a brief description of each application:

- *Adjacency-List*: produces the list of the neighbors for each vertex of a graph. Algorithms such as PageRank can use the results.

- *Classification*: uses the data of movie rating and classifies the movies into clusters.
- *Grep*: a common tool in many data analyses. It searches for a pattern in a file.
- *Histogram-Movies*: gets the movie rating data as input and generates the histogram of movies.
- *Histogram-Ratings*: the input is the same as Histogram-Movies, but it generates the histogram of ratings.
- *Inverted-Index*: generates the mapping of word to document for a set of documents and their constituent words.
- *K-Means*: a popular data mining algorithm that is used for clustering input data into k clusters. The input data of K-Means in our experiments is movie rating data.
- *Pi-Estimator*: estimates the value of Pi by a tunable precision.
- *Ranked-Inverted-Index*: for a list of words and number of their repetitions in documents, generates the decreasing list of documents that given words appears in them.
- *Self-Join*: gets the association of k fields and generates it for k+1 fields.
- *Sequence-Count*: finds the number of all individual sets of three successive words per document
- *Tera-Sort*: a well-known Hadoop benchmark. It sorts 100-byte tuples where the first 10-byte is key and rest is value.
- *Term-Vector*: suitable for finding the frequent words in a set of documents. The data can be used for understanding the relevance of a host to a search.
- *Word-Count*: calculates the number of appearance of each word in a set of documents

VM configurations. The specification of VMs that we have used as different configurations for generating Table I are presented in Table III. Since it is a common practice to consider the number of Map and Reduce slots same as the number of cores that a machine has [30], [31], we also considered the number of slots for each VM equal as the number of its cores. The specifications of VMs are obtained from Google Cloud [32], Amazon EC2 [16] and Microsoft azure [17].

B. Impact of SCS on Estimation Accuracy

We applied both variants of SCS (SCS_PCC and SCS_KRCC which use Pearson and Kendall correlation coefficients respectively) on Table I. For each application, we first removed it from the table and then applied SCS, so values of that application could not affect the results. Applying SCS_PCC, it selected configurations 1 and 13 for all the application except Classification, where it chose configurations 1 and 11. SCS_KRCC, on the other hand, opted configurations 9 and 13 as sample configurations for most of the applications. For applications InvertedIndex and WordCount, SCS_KRCC chose configurations 9 and 12.

The comparison between SCS_PCC, SCS_KRCC, Average of all possible pairs of sample configurations (Average in short) and best pair and worst pair of sample configurations per application are depicted in Fig. 6. The average estimation error of SCS_PCC over 14 applications is 11.58% and for SCS_KRCC it is 20.56%. The average of worst case is 44.90%, average of best case is 5.88% and average of all states is 19.72%. Considering the average results, we conclude that overall performance of SCS when using PCC, SCS_PCC, is better than when it uses KRCC (SCS_KRCC) since SCS_PCC can obtain more accurate estimations than SCS_KRCC.

However, exploring the results in more detail, we perceive that while overall performance of SCS_PCC is better than SCS_KRCC, in some applications the SCS_KRCC yields more accurate estimations than SCS_PCC. In two applications, Grep and HistogramRatings, SCS_KRCC can obtain near best

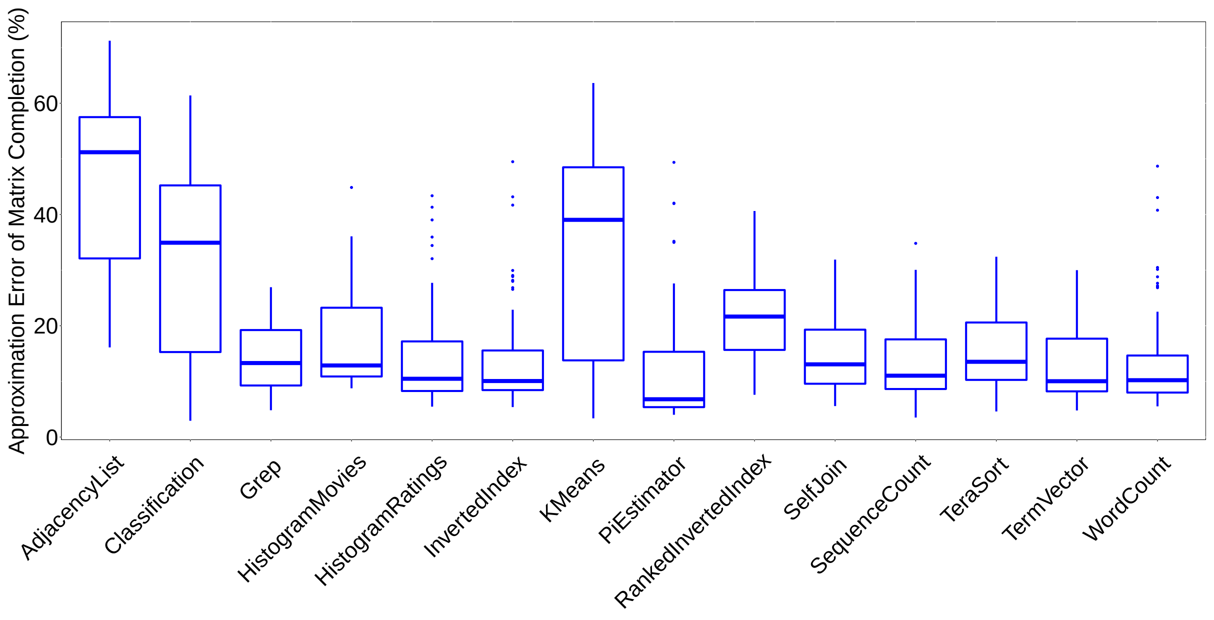


Fig. 5. Scattering of approximation error for all the possible sample configurations.

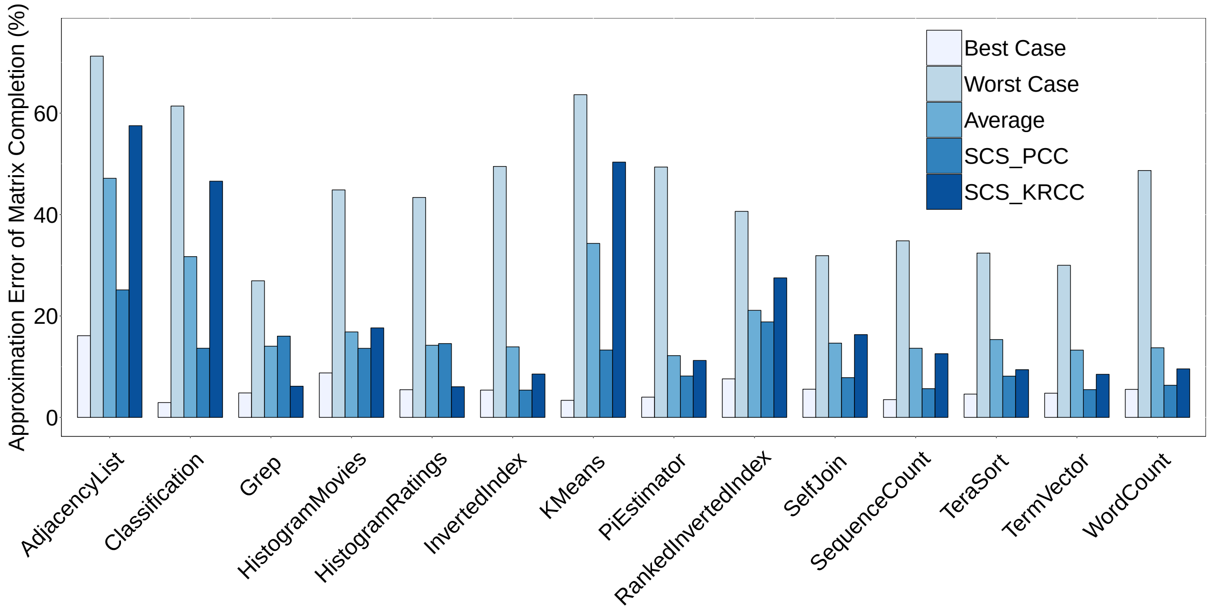


Fig. 6. Comparison of SCS with other sample configurations.

case answers, while SCS_PCC estimations are even worse than average. Thus, leveraging both PCC and KRCC at the same time might lead to near best case estimations.

Up to this point, we showed how our SCS approach can decrease the approximation error of matrix completion by leveraging the correlation between configurations, and hence, opting more appropriate sample configurations than conventional random approach. The average execution time of SCS approach is around 30 ms when implemented in MATLAB (R2013a) and executed on a machine with Core i5 processor (2.96 GHz) and 4GB of memory, and hence, its time overhead is absolutely negligible.

To further evaluate the accuracy of SCS approach, we have conducted more experiments by changing Hadoop parameters and also input data of applications. In two experiments, we have changed the block size from 64MB of Fig. 6 to 32MB and 128MB. We have also changed slow start parameter from 1 to 0.1 and 0.5. Slow start determines how much the Map phase and Reduce phase will have overlap. We have chosen block size and Slow start parameter because a large body of research has studied them and it shows their importance among Hadoop parameters [33]–[37]. Block size affects the I/O performance of Hadoop clusters and Slow start has significant impact on the network behavior of clusters, especially during shuffle phase of MapReduce applications. Finally, we have changed the input data of applications in another two experiments to evaluate the impact of data variety, which is a key feature of Big Data, on the performance of our SCS approach. We have changed the input data of five applications: Grep, Inverted-Index, Sequence-Count, Term-Vector, Word-Count (Since we did not have data from different sources for other applications, we could not change their input data). The original source of input data for those five applications was Wikipedia data set from PUMA benchmark suit [19]. We used data from Gutenberg [38] and Common Crawl [39] sources for new experiments instead of Wikipedia.The results are presented in Fig. 7. Yet again, we can see that SCS can yield near best case or even best case predictions. SCS_PCC still has more accurate results compared with SCS_KRCC. From these results we conclude that SCS can improve the prediction accuracy under different scenarios, and hence, can be used in production clusters where jobs with different settings and input data are submitted constantly.

In the next section, we study the impact of estimation error on resource allocation and performance of applications.

TABLE III
VM SPECIFICATION USED IN OUR EXPERIMENTS

	VM Type	CPU (#)	Mem (GB)	Map Slot (#)	Reduce Slot (#)	Memory of Slot (GB)
Config 1	amazon-t2.small	1	2	1	1	2000
Config 2	amazon-t2.medium	2	4	2	2	2000
Config 3	amazon-t2.large	2	8	2	2	4000
Config 4	amazon-m3.medium	1	3.75	1	1	3700
Config 5	azure-A4	8	14	8	8	1750
Config 6	azure- D2	2	7	2	2	3500
Config 7	azure- D3	4	14	4	4	3500
Config 8	azure- F4	4	8	4	4	2000
Config 9	google-n1-highmem-2	2	13	2	2	6500
Config 10	google-n1-highcpu-2	2	1.8	2	2	920
Config 11	google-n1-highcpu-4	4	3.6	4	4	920
Config 12	google-n1-highcpu-8	8	7.2	8	8	920
Config 13	google-n1-highcpu-16	16	14.4	16	16	920

C. Estimation Accuracy Effect on Resource Management

Makespan minimization of MapReduce applications is pursued in a large body of MapReduce scheduling and resource

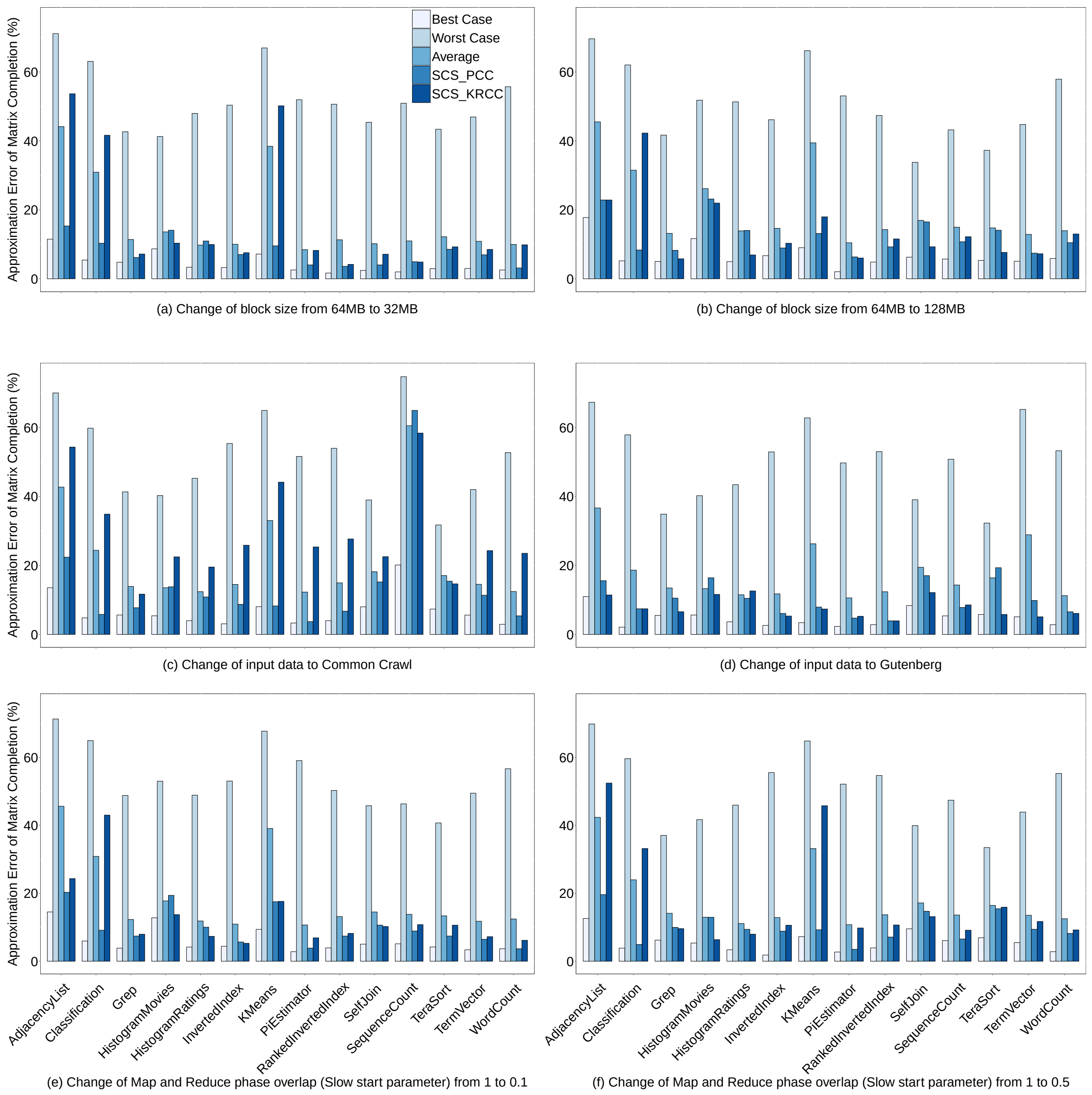


Fig. 7. Further experiments to evaluate the effectiveness of SCS approach under different Hadoop settings and input data.

allocation research [40]–[44]. Hence, we implement such an algorithm to investigate the effect of estimation accuracy on its performance. First, this algorithm profiles the application to obtain information such as Table I. The profiling approach of algorithm is the same as Fig. 2. It uses matrix completion to estimate the performance of applications on different configurations using known information of sample configurations. Then, it schedules the tasks of job on available instances of those configurations using profiling information in order to minimize the execution time of job.

For the sample configurations in profiling phase (see Fig. 2), we select them once randomly, and once more, using SCS approach. In either case, these sample configurations are given to the makespan algorithm to schedule the tasks. After scheduling the job based on profiling information, we calculate the execution time of job using real information of actual execution time we already have such as Table I. In this way, we obtain the impact of profiling on the actual final job execution time. Finally, we compare the results of random selection and SCS approach to illustrate how much SCS approach can be effective in terms of makespan minimization scheduling.

The results of makespan minimization algorithm for all applications with respect to 5 different random sample configuration selections, two variants of SCS approach (SCS_PCC and SCS_KRCC), and the best pair (i.e. the pair that minimizes the approximation error in Matrix Completion) are presented

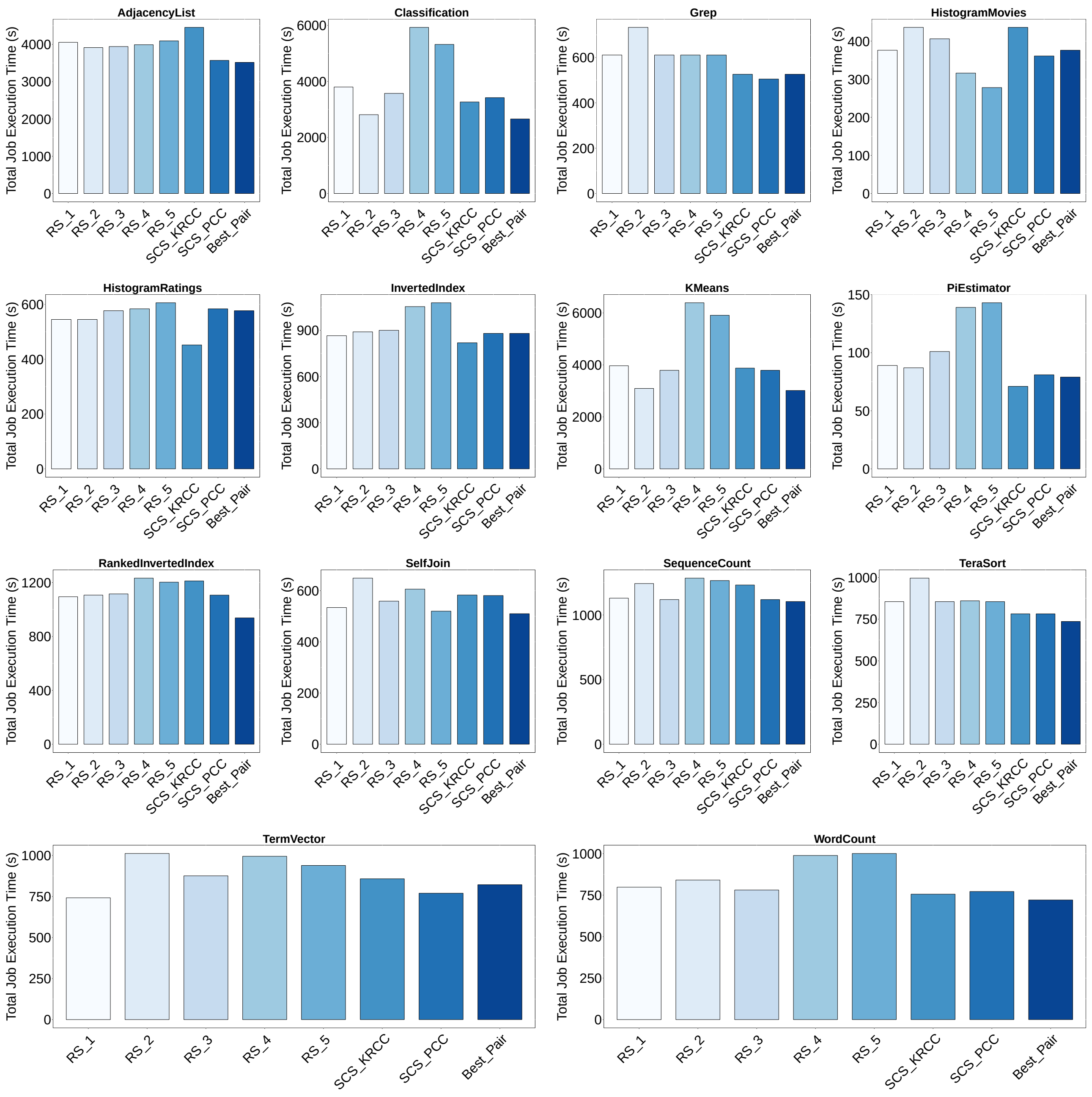


Fig. 8. Results of makespan minimization algorithm for different applications. In the Profiling Phase of makespan minimization, we have used matrix completion with different sample configuration selection approaches. RS_1 to RS_5 sample configurations are selected randomly. In SCS_KRCC and SCS_PCC, we have applied our smart configuration selection (SCS) approach. The best pair of sample configurations is found using brute force approach.

in Fig. 8. The cluster that msakespan minimization algorithm schedules the job on it consists of 130 VMs (10 instances from each VM type of Table III).

The results indicate that while a random sample can sometimes lead to better answers than SCS, its fluctuation in different applications is significant. Note that inaccurate performance estimation does not necessarily lead to poor scheduling. Obviously an imprecise estimation can accidentally yield a good schedule in some cases as we can see in Fig. 8, but the point is that on average over several applications, it has detrimental impact and degrades the performance dramatically. For example, we see that in RankedInvertedIndex and TermVector applications, the schedule based on results of matrix completion using RS_1 has obtained less execution time than two variants of SCS. However, overall performance of it over 14 applications illustrates that it is not a good choice. On the other hand, investigating the results obtained by SCS reveals that more accurate estimation can significantly reduce the execution time.

Comparing the SCS with average of five random samples demonstrate that using SCS_KRCC estimations in makespan minimization algorithm can improve the execution time by up to 36.03% and the average of improvement over all 14 applications is 8.51%. Using SCS_PCC also can yield improvement by up to 27.02% (10.84% on average). Only in two applications (SelfJoin and HistogramMovies) the SCS_PCC cannot improve the execution time compared to the average of all random samples. In these applications, SCS estimation degrades the execution time by 1.39% and 2.27% for SelfJoin and HistogramMovies, respectively, compared against average of random samples. Normalized average times to SCS_PCC are depicted in Fig. 9.

From the results in Fig. 8, we conclude that investing more time and resources in profiling phase in order to improve its accuracy can be compensated in resource management and scheduling phase by obtaining higher performance and less execution time. We showed, by measurements, that spending 30 ms (less than 0.01% of total execution time of job) more time on configuration selection by SCS for performance estimation yields up to 36%, and 10% on average, higher performance than blind random configuration selection.

Comparison between the best pair results and SCS in Fig. 8 sheds light on how much opportunity is left to improve SCS estimation accuracy. Scheduling based on the best pair estimation improves the makespan by up to 22.60% and 22.21% compared with SCS_KRCC and SCS_PCC respectively. The average improvement of the best pair for all the applications is 6.39% and 5.29% for SCS_KRCC and SCS_PCC respectively. These results indicate that it is still worth to investigate proposing even better approaches to find the best pair of sample configurations for each application. Note that the best pair of sample configurations for each application varies from one to another. Another interesting point to note is that in some cases, e.g. Grep and HistogramMovies, the execution time obtained by the best (in terms of approximation error) pair of sample configurations is marginally (4%) higher than SCS_PCC. Note that while the estimation accuracy has an important role in scheduling, it is not the only factor that affects it. Other factors such as cluster specification and its resources as well as scheduling algorithm also affect the final results. That is why we see SCS_PCC slightly outperforms the best pair in two applications.

V. Related Work

Profiling applications for resource allocation and scheduling purpose is commonplace especially in data centers and cloud environment [45]–[47]. To address the heterogeneous MapReduce placement in cloud, [10] employs a subset of MapReduce tasks and profiles them on all the available VM configurations to obtain their performance. Then, it uses the profiling information to allocate resources to all the MapReduce tasks. A similar approach is employed by [30], but instead of profiling a subset of tasks it profiles all the tasks on all the VM configurations. This approach is suitable for periodic MapReduce jobs where the number of tasks and their data size is constant, but the data contents changes from job to job. While these approaches use complete profiling, we use partial profiling in our work and do not profile jobs on all the VM configurations.

Leveraging matrix completion to estimate the performance of applications on all the configurations is objective of [14], [48]. In [14], each application is executed on two sample configurations. Then it uses matrix completion to estimate the performance of application on other configurations without executing the application. Then, it uses obtained performance values to choose the best configuration. Moreover, it estimates the interference between applications again using matrix completion. It co-locates the applications on physical machines based on obtained information to reduce performance degradation due to contention between applications. The proposed QoS ranking prediction framework in [48], employs matrix completion to predict the ranking of functionally equivalent cloud services without real world service invocations. It uses the experience of past users to predict the QoS of each service for current users. While both [14], [48] utilize matrix completion, none of them considers the impact of sample configurations or sample services on the accuracy of matrix completion. However, we take into account this fact and depict its influence on final results.

Several works [7]–[9], [11], [13], [49], [50] use partial profiling in the process of resource allocation or scheduling for MapReduce applications in cloud and data centers. The prediction module in [13] employs locally weighted regression to estimate the performance of MapReduce application. The prediction module is fed by a job analyzer module that gathers information from currently executed jobs. CRESP [9] presents mathematical model for execution time of tasks based on profiled information. HFSP [50] first estimates the size of jobs based on a sample subset of tasks and then schedules the jobs on cluster according to their size. Using Starfish [51] profiling tool, Cura [7] proposes a resource allocation mechanism to reduce cost of VMs for executing Hadoop jobs in cloud.

In the following, we will discuss Starfish [51] and MRTuner [52] in more details and explain their difference with our approach. First, we explain the mechanism that they use in their what-if engines. Prior to using the engine, a job or workload needs to be profiled executing a certain percent of its tasks on a cluster. Using this profiling, they construct a job profile that consists of information such as execution time of different phases and tasks, CPU and IO cost of tasks and subtasks, and amount of read and written bytes. After that, when the user sends a query to system to evaluate a new setting (e.g. change in number of nodes, MapReduce parameters, input data), the system tries to construct a virtual profile for job based on previous real one. After constructing the virtual profile, using simulation, they estimate different parameters such as total execution time of job under new settings. The process of constructing virtual profile and estimating performance on new settings is done through a what-if-engine. The what-if-engine uses a mixture of black-box and white-box models to construct the virtual profiles. During the process of constructing virtual profiles, it may need to generate training data by running actual jobs for black-box models. Finally, after the constructing virtual profile for a certain scenario, they simulate the virtual profile to obtain the final performance of job on the hypothetical cluster and under hypothetical MapReduce configurations.

For analyzing different cluster configurations (e.g., changing the number of nodes in cluster or changing the type of nodes),

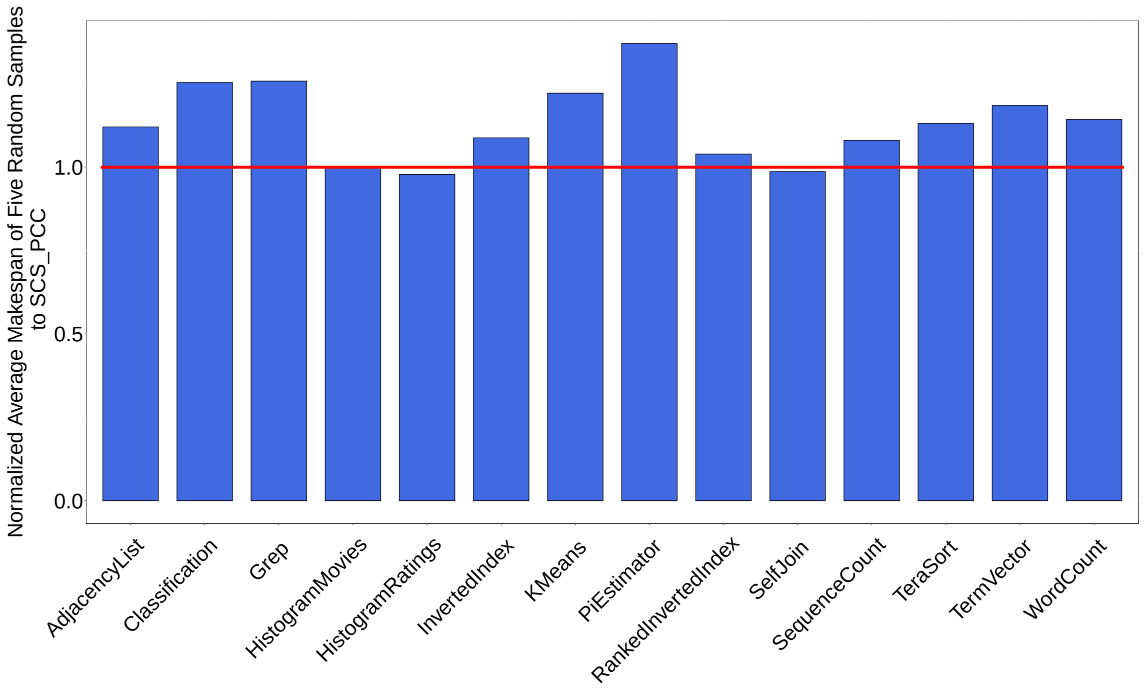


Fig. 9. Normalized average makespan of scheduling using all the five random samples to the SCS_PCC for different applications. Except for HistogramRatings and Selfjoin applications, in other applications the average execution time of random samples is higher than SCS_PCC.

the what-if engine needs to know the performance of job on all the available machine configurations. For example, if the cloud consists of 10 different VM configurations, it needs to profile the job on all of them. In other words, *it cannot estimate the performance of job on a VM configuration by knowing performance on other VM configurations and needs to use complete profiling.* Here, our proposed SCS approach can tackle the problem of complete profiling. The relationship between our approach and what-if engine is depicted in Fig. 10. This figure indicates that Starfish is not an alternative for our approach, but complementary to it. Our SCS approach can estimate the performance of new jobs on VMs and Starfish can use them to evaluate and analyze different scenarios.

VI. Conclusion

Obtaining accurate estimation for performance of applications on VM configurations in cloud can lead to better resource allocation and scheduling, and consequently, to faster execution. Matrix completion is among well-known estimation methods that use partial profiling to estimate the performance. It first profiles the applications on a subset of configurations (sample configurations) and then estimates the performance of application on all the remaining configurations. Our studies show that the choice of these sample configurations has direct impact on the estimation accuracy, and hence, choosing them wisely can improve the precision of matrix completion. In this work, we proposed the SCS approach that selects the sample configurations based on correlation coefficient. Our experiments illustrate that SCS improves the accuracy significantly. Further evaluations show that this more accurate estimation leads to better scheduling by a makespan minimization algorithm. The algorithm, powered by our SCS estimation engine, improved the execution time of applications up to 36% compared against random selection of sample configurations.

The best pair of sample configurations differs per application. We saw in Fig. 6 that the best pair can still further decrease the approximation error compared with SCS; we showed this can further reduce makespan by 6% on average in Fig. 8. Finding this pair for each application is an interesting direction for future research. Another direction for future work is proposing new metrics or approaches to select the sample configurations. For example, we can use clustering methods instead of correlation coefficient to identify the dichotomy between configurations. Furthermore, exploring the proper number of sample configurations, to best represent a given training set, based on the number of configurations present in that set is another open avenue for further research.

Acknowledgment

This research is supported by grant number G930826 from Sharif University of Technology. We are grateful for their support.

References

[1] J. Gantz and D. Reinsel, “The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east,” *IDC iView: IDC Analyze the future*, vol. 2007, pp. 1–16, 2012.
[2] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[3] “Apache hadoop,” https://hadoop.apache.org, accessed: 2016-08-16.
[4] “Amazon emr,” http://aws.amazon.com/elasticmapreduce, accessed: 2016-08-16.
[5] M. Malekimajd, D. Ardagna, M. Ciavotta, A. M. Rizzi, and M. Passacantando, “Optimal map reduce job capacity allocation in cloud systems,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 4, pp. 51–61, 2015.

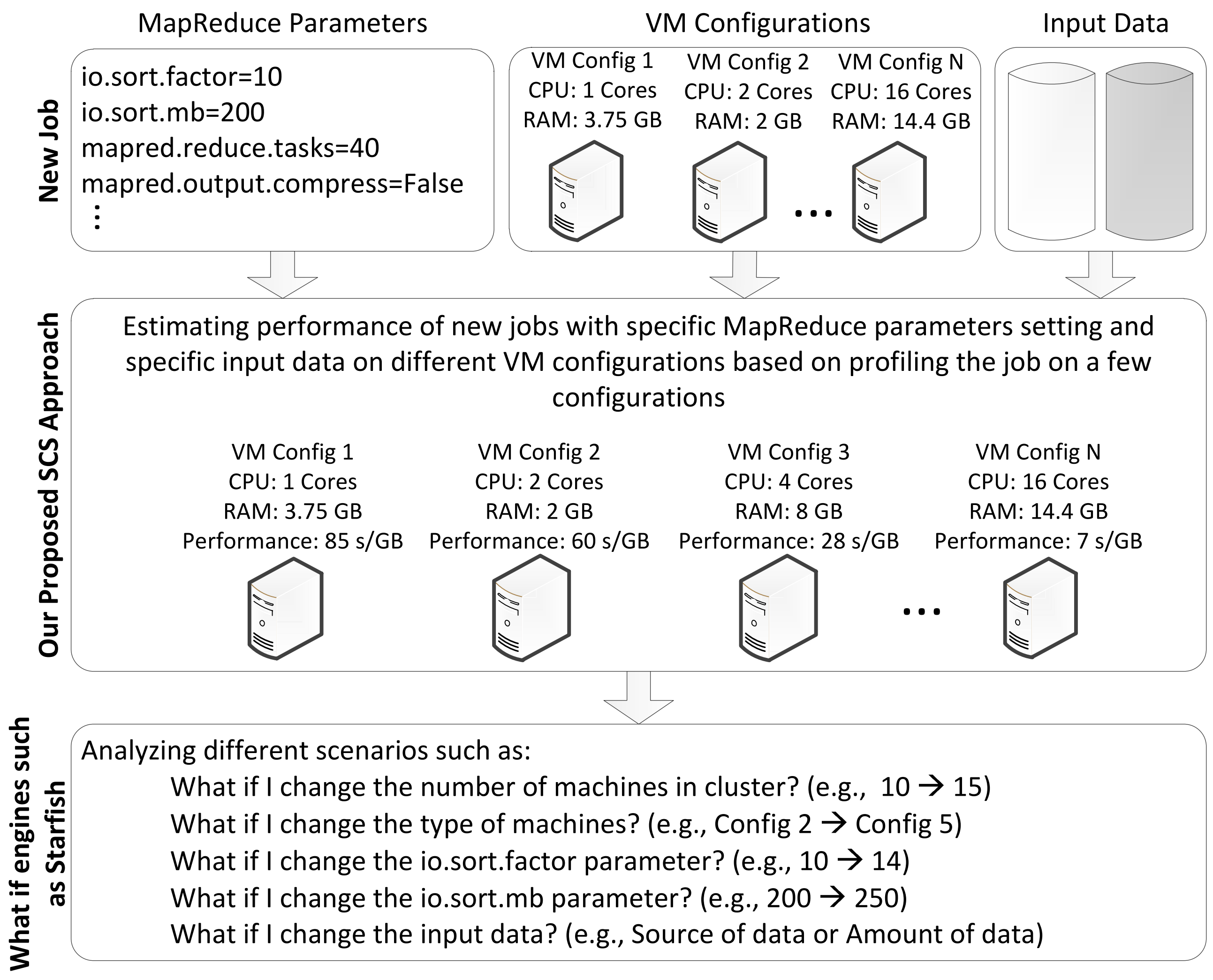

MapReduce Parameters
VM Configurations
Input Data
New Job
io.sort.factor=10
io.sort.mb=200
mapred.reduce.tasks=40
mapred.output.compress=False
VM Config 1
CPU: 1 Cores
RAM: 3.75 GB
VM Config 2
CPU: 2 Cores
RAM: 2 GB
VM Config N
CPU: 16 Cores
RAM: 14.4 GB
Our Proposed SCS Approach
Estimating performance of new jobs with specific MapReduce parameters setting and specific input data on different VM configurations based on profiling the job on a few configurations
VM Config 1
CPU: 1 Cores
RAM: 3.75 GB
Performance: 85 s/GB
VM Config 2
CPU: 2 Cores
RAM: 2 GB
Performance: 60 s/GB
VM Config 3
CPU: 4 Cores
RAM: 8 GB
Performance: 28 s/GB
VM Config N
CPU: 16 Cores
RAM: 14.4 GB
Performance: 7 s/GB
What if engines such as Starfish
Analyzing different scenarios such as:
What if I change the number of machines in cluster? (e.g., 10 → 15)
What if I change the type of machines? (e.g., Config 2 → Config 5)
What if I change the io.sort.factor parameter? (e.g., 10 → 14)
What if I change the io.sort.mb parameter? (e.g., 200 → 250)
What if I change the input data? (e.g., Source of data or Amount of data)

Fig. 10. The relationship between SCS and what-if engine approaches such as Starfish

[6] E. Hwang and K. H. Kim, "Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud," in *2012 ACM/IEEE 13th Intl. Conf. on Grid Computing*, 2012, pp. 130–138.

[7] B. Palanisamy, A. Singh, and L. Liu, "Cost-effective resource provisioning for mapreduce in a cloud," *IEEE Trans. on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1265–1279, 2015.

[8] Z. Zhang, L. Cherkasova, and B. T. Loo, "Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 4, pp. 38–50, 2015.

[9] K. Chen, J. Powers, S. Guo, and F. Tian, "Cresp: Towards optimal resource provisioning for mapreduce computing in public clouds," *IEEE Trans. on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1403–1412, 2014.

[10] X. Xu and M. Tang, "A new approach to the cloud-based heterogeneous mapreduce placement problem," *IEEE Trans. on Services Computing*, in press 2015.

[11] Y. Wang and W. Shi, "Budget-driven scheduling algorithms for batches of mapreduce jobs in heterogeneous clouds," *IEEE Trans. on Cloud Computing*, vol. 2, no. 3, pp. 306–319, 2014.

[12] Q. Zhang, M. F. Zhani, Y. Yang, R. Boutaba, and B. Wong, "Prism: fine-grained resource-aware scheduling for mapreduce," *IEEE Trans. on Cloud Computing*, vol. 3, no. 2, pp. 182–194, 2015.

[13] G. Song, Z. Meng, F. Huet, F. Magoules, L. Yu, and X. Lin, "A hadoop mapreduce performance prediction method," in *IEEE HPCC_EUC'13*, 2013, pp. 820–825.

[14] C. Delimitrou and C. Kozyrakis, "The netflix challenge: Datacenter edition," *IEEE Computer Architecture Letters*, vol. 12, no. 1, pp. 29–32, 2013.

[15] E. J. Candès and B. Recht, "Exact matrix completion via convex optimization," *Foundations of Computational mathematics*, vol. 9, no. 6, pp. 717–772, 2009.

[16] "Amazon ec2," http://aws.amazon.com/ec2, accessed: 2016-08-16.

[17] "Microsoft azure," https://azure.microsoft.com/en-us, accessed: 2016-08-16.

[18] S. M. Nabavinejad and M. Goudarzi, "Energy efficiency in cloud-based mapreduce applications through better performance estimation," in *DATE'16*. IEEE, 2016, pp. 1339–1344.

[19] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Puma: Purdue mapreduce benchmarks suite," 2012.

[20] S. R. Becker, E. J. Candès, and M. C. Grant, "Templates for convex cone problems with applications to sparse signal recovery," *Mathematical programming computation*, vol. 3, no. 3, pp. 165–218, 2011.

[21] R. M. Bell, Y. Koren, and C. Volinsky, "The bellkor 2008 solution to the netflix prize," *Statistics Research Department at AT&T Research*, 2008.

[22] R. H. Keshavan, S. Oh, and A. Montanari, "Matrix completion from a few entries," in *IEEE Intl. Symposium on Information Theory*, 2009, pp. 324–328.

[23] P. Jain, P. Netrapalli, and S. Sanghavi, "Low-rank matrix completion using alternating minimization," in *ACM symposium on Theory of computing*, 2013, pp. 665–674.

[24] J. I. Marden, *Analyzing and modeling rank data*. CRC Press, 1996.

[25] W. R. Knight, "A computer method for calculating kendall's tau with ungrouped data," *Journal of the American Statistical Association*, vol. 61, no. 314, pp. 436–439, 1966.

[26] S. M. Nabavinejad, M. Goudarzi, and S. Mozaffari, "The memory

challenge in reduce phase of mapreduce applications," *IEEE Trans. on Big Data*, vol. 2, no. 4, pp. 380–386, 2016.

[27] B. Wang, J. Jiang, Y. Wu, G. Yang, and K. Li, "Accelerating mapreduce on commodity clusters: An ssd-empowered approach," *IEEE Trans. on Big Data*, in press 2016.

[28] M. C. Lee, J. C. Lin, and R. Yahyapour, "Hybrid job-driven scheduling for virtual mapreduce clusters," *IEEE Trans. on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1687–1699, 2016.

[29] Z. Bei, Z. Yu, H. Zhang, W. Xiong, C. Xu, L. Eeckhout, and S. Feng, "Rfhoc: A random-forest approach to auto-tuning hadoop configuration," *IEEE Trans. on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1470–1483, May 2016.

[30] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of mapreduce jobs for big data applications," *IEEE Trans. on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2720–2733, 2015.

[31] F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni, "Optimizing power and performance trade-offs of mapreduce job processing with heterogeneous multi-core processors," in *IEEE CLOUD'14*, 2014, pp. 240–247.

[32] "Google cloud," https://cloud.google.com, accessed: 2016-08-16.

[33] H. Ahmed and M. A. Ismail, "Impact of hdfs block size in mapreduce based segmentation and feature extraction algorithm," in *ICOSST'15*, Dec 2015, pp. 58–63.

[34] T. L. S. R. Krishna, T. Ragunathan, and S. K. Battula, "Performance evaluation of read and write operations in hadoop distributed file system," in *2014 Sixth Intl. Symposium on Parallel Architectures, Algorithms and Programming*, 2014, pp. 110–113.

[35] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Mapreduce with communication overlap (marco)," *Journal of Parallel and Distributed Computing*, vol. 73, no. 5, pp. 608–620, 2013.

[36] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Shufflewatcher: Shuffle-aware scheduling in multi-tenant mapreduce clusters," in *USENIX ATC'14*, 2014, pp. 1–13.

[37] J. Wang, M. Qiu, B. Guo, and Z. Zong, "Phase–reconfigurable shuffle optimization for hadoop mapreduce," *IEEE Trans. on Cloud Computing*, in press 2015.

[38] "Project gutenberg," https://www.gutenberg.org, accessed: 2016-09-17.

[39] "Common crawl," http://commoncrawl.org, accessed: 2016-09-17.

[40] Y. Yao, J. Wang, B. Sheng, C. Tan, and N. Mi, "Self-adjusting slot configurations for homogeneous and heterogeneous hadoop clusters," *IEEE Trans. on Cloud Computing*, in press 2015.

[41] Y. Yao, J. Tai, B. Sheng, and N. Mi, "Lsps: A job size-based scheduler for efficient task assignments in hadoop," *IEEE Trans. on Cloud Computing*, vol. 3, no. 4, pp. 411–424, 2015.

[42] R. Tudoran, A. Costan, and G. Antoniu, "Overflow: Multi-site aware big data management for scientific workflows on clouds," *IEEE Trans. on Cloud Computing*, vol. 4, no. 1, pp. 76–89, 2016.

[43] S. Tang, B.-S. Lee, and B. He, "Dynamic job ordering and slot configurations for mapreduce workloads," *IEEE Trans. on Services Computing*, vol. 9, no. 1, pp. 4–17, 2016.

[44] A. Verma, L. Cherkasova, and R. H. Campbell, "Orchestrating an ensemble of mapreduce jobs for minimizing their makespan," *IEEE Trans. on Dependable and Secure Computing*, vol. 10, no. 5, pp. 314–327, 2013.

[45] A. O. Ayodele, J. Rao, and T. E. Boult, "Performance measurement and interference profiling in multi-tenant clouds," in *IEEE Intl. Conf. on Cloud Computing*, 2015, pp. 941–949.

[46] F. Juarez, J. Ejarque, and R. M. Badia, "Dynamic energy-aware scheduling for parallel task-based application in cloud computing," *Future Generation Computer Systems*, in press 2016.

[47] S. M. Nabavinejad and M. Goudarzi, "Chapter five-communication-awareness for energy-efficiency in datacenters," *Advances in Computers*, vol. 100, pp. 201–254, 2016.

[48] Z. Zheng, X. Wu, Y. Zhang, M. R. Lyu, and J. Wang, "Qos ranking prediction for cloud services," *IEEE Trans. on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1213–1222, 2013.

[49] F. Zhang, J. Cao, K. Hwang, K. Li, and S. U. Khan, "Adaptive workflow scheduling on cloud computing platforms with iterativeordinal optimization," *IEEE Trans. on Cloud Computing*, vol. 3, no. 2, pp. 156–168, 2015.

[50] M. Pastorelli, D. Carra, M. Dell'Amico, and P. Michiardi, "Hfsp: Bringing size-based scheduling to hadoop," *IEEE Trans. on Cloud Computing*, in press 2015.

[51] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics." in *CIDR*, vol. 11, 2011, pp. 261–272.

[52] J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang, "Mrtuner: a toolkit to enable holistic optimization for mapreduce jobs," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1319–1330, 2014.

Seyed Morteza Nabavinejad is a Ph.D. candidate at the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. He received the B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad and the M.Sc. in Computer Architecture from Sharif University of Technology in 2011 and 2013, respectively. He is currently working at Energy Aware Systems (EASY) Laboratory under supervision of Dr. Maziar Goudarzi. His research interests include big data processing, cloud computing, green computing, and energy aware datacenters.

Maziar Goudarzi (S00, M11, SM'16) is an Associate Professor at the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. He received the B.Sc., M.Sc., and Ph.D. degrees in Computer Engineering from Sharif University of Technology in 1996, 1998, and 2005, respectively. Before joining Sharif University of Technology as a faculty member in September 2009, he was a Research Associate Professor at Kyushu University, Japan from 2006 to 2008, and then a member of research staff at University College Cork, Ireland in 2009. His current research interests include architectures for large-scale computing systems, green computing, hardware-software codesign, and reconfigurable computing. Dr. Goudarzi has won two best paper awards, published several papers in reputable conferences and journals, and served as member of technical program committees of a number of IEEE, ACM, and IFIP conferences including ICCD, ASP-DAC, ISQED, ASQED, EUC, and IEDEC among others.