Masha: Sampling-Based Performance Prediction of Big Data Applications in Resource-Constrained Clusters

Hani Al-Sayeh[1] , Bunjamin Memishi[2], Marcus Paradies[2], Kai-Uwe Sattler[1]
[1]TU Ilmenau, Germany
[2]German Aerospace Center
{hani-bassam.al-sayeh,kus}@tu-ilmenau.de
{bunjamin.memishi,marcus.paradies}@dlr.de

ABSTRACT

Nowadays deployment of data-intensive systems in multi-dimensional domains is achieved with insufficient knowledge regarding the data, application internals, and infrastructure requirements. In addition, the current performance prediction frameworks focus to predict the performance of data-intensive applications on mid to large-scale infrastructures, which does not seem to be always the case. We reproduced 16 applications on a small-scale cluster, and obtained concerning results from a baseline prediction framework. Consequently, we argue that neither the previous design of the experiments, nor the prediction models are sufficiently accurate at resource-constrained cluster scenarios. Therefore, we propose Masha, a new, black-box, sampling-based approach, that is initially lead by a new design of experiments, without relying on any historical executions. This is followed by a new performance prediction model, whose main idea is that apart from the computation, the data also needs a first citizen role. Our preliminary results are promising, by means of being able to characterize complex applications, having an average prediction accuracy of 83 %, and with a negligible overhead cost of only 2.42 %. Being framework-independent, Masha is applicable to any data-intensive distributed system.

1. INTRODUCTION

The abundant availability and nearly infinite scalability of compute & storage resources in public and private data center cluster infrastructures has spurred a rapid growth of advanced analytics tasks on massive-scale data sets from data-intensive sciences. This includes complex, compute-intensive algorithms, such as deep learning for classification & object detection tasks in medical [12] and satellite images [11], pulsar discovery in astronomy [7], and genome analysis [16]. In contrast to traditional, short-running, descriptive and explorative data analytics tasks, such advanced analytics tasks are not only data-intensive (i.e., I/O-intensive), but tend to be also computationally expensive and long-running (up to multiple days to weeks). With the recent advent of powerful, parallel computation frameworks, such as Apache Spark [21], Dryad [10], and Apache Flink [6], increasingly such advanced analytics tasks are implemented as dataflow programs and deployed on large public or private cloud infrastructures. Caused by the ever-growing diversification of available instance types in public cloud infrastructures with respect to their compute, memory, storage, and network characteristics, users are confronted with choosing the right instance configuration for a given application task.

Besides choosing the best-performing (and most economical) instance types in public cloud infrastructures, answering *what-if* questions for initial cluster hardware sizing and precise prediction of resource consumption of advanced analytics tasks in resource-limited cloud infrastructures demand for accurate performance prediction models of such tasks. Without dispute, execution time prediction for complex advanced analytics tasks in parallel computation frameworks is a core building block for addressing the aforementioned challenges and already received significant attention in the research community [4, 5, 13, 14, 18, 20].

A common approach is to run the given analytics job on sample input sets and to use the derived performance measures to create a job-specific performance model [20]. Besides devising a performance model skeleton, a major challenge lies in identifying the minimal number of sampling measurements, which sufficiently capture the applications' behavior for larger data sets. Our initial study of a popular existing performance prediction framework (cf. Section 3) reveals that the proposed techniques cannot easily be transferred from medium-sized clusters (40-60 machines within a cluster) to small-sized clusters (less than 20 machines) and to data-intensive applications from the HiBench big data benchmarking suite [3, 9]. To that end, we tackle the challenge of achieving accurate, low-overhead performance predictions on small-size clusters with less than 20 machines for a large variety of 16 different long-running data-intensive applications from HiBench. In order to isolate the development of a performance prediction model from the optimal design of experiments (DoE) methodology, we propose a DoE *oracle*, which produces an optimal set of sampling experiments independent of the actual prediction model. Our devised performance model provides 83 % prediction accuracy while achieving a comparable overhead of 2.42 % on average. In summary, we make the following contributions:

- We conduct an extensive performance analysis of 16 different data-intensive applications covering 120 different sampling configurations.
- We devise a DoE *oracle* methodology, which separates the optimal DoE design from the prediction model and allows independent performance comparison between different performance prediction models.
- We propose an enhanced performance prediction model and show experimentally using 16 different data-intensive applications that our approach provides on average 43.5 % higher accuracy for resource-constrained clusters compared to Ernest while being competitive with respect to the overhead induced by the additional training runs.

The remainder of the paper is organized as follows. Section 2 gives preliminary background. Section 3 describes the findings from an analysis of Ernest in resource-constrained clusters. Section 4 introduces the DoE *oracle* methodology. Section 5 introduces our approach. Section 6 shows the evaluation results of Masha compared with our baseline (i.e., Ernest). In Section 7 we discuss and analyse the limitations of Masha and Ernest, followed by a related work in Section 8. Finally, Section 9 summarizes the paper and provides outlook.

2. BACKGROUND

In this section, we briefly explain the main idea behind sampling and its main challenges (cf. Section 2.1). In addition, we discuss further details about big data analytics in Section 2.2.

2.1 Sampling

In complex computer systems such as distributed systems, it is hard to dive into details of each system component to construct performance prediction models and consequently make suitable decisions. In these occasions, sample system runs on a smaller problem sizes (e.g., data fractions) and various system configurations (e.g., number of processing units) can be performed to monitor and analyze system performance. Before carrying out sample runs, the following two main questions will have to be answered: (1) How many sample points do we need and which are the selected configurations and data fractions? (2) How to analyze the sample runs and predict the performance of the actual run? A design of experiments (DoE) component needs to answer the first question (cf. Section 4). A performance prediction model that takes various configurations into consideration addresses the second question (cf. Section 5).

Challenges. In the following, we list the main challenges of sampling-based performance prediction:

- Sampling overhead. The time and cost of sample runs should be relatively small compared with actual system runs. This is the main reason why sampling is not a convenient approach in many cases, where actual system runs are short-running. However, reducing sampling costs by limiting the number of experiments can affect the prediction accuracy of the performance model.
- Generality. This challenge needs to be addressed during the development of the performance prediction model. It should be comprehensive and cover various workloads and system behaviours.
- Prediction accuracy. In order to achieve acceptable performance accuracy, both, DoE and prediction model need to be well harmonized. Herein, the main challenge is that one component cannot be presented or evaluated individually, since they are tightly coupled. Therefore, many sampling based approaches present both components at the same time. To that end, in case of low prediction accuracy, we cannot easily determine which component causes the low prediction accuracy, and whether changing one component requires also changes in the other component.

2.2 Big data analytics

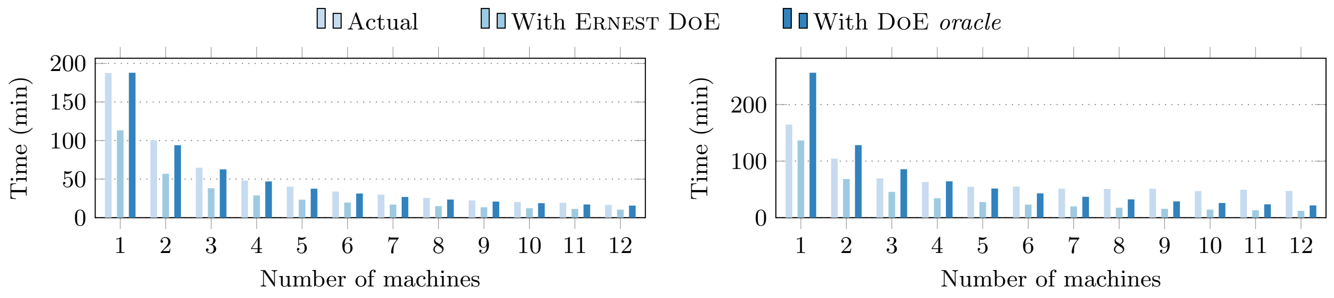
In order to run complex, long-running data flows on large data volumes, the data is partitioned into smaller chunks (called blocks), which are then distributed across processing machines and processed in parallel. Some operations (e.g., aggregation and join) require data exchange between machines (i.e., data shuffling). The collection of operators that follows each other and does not require shuffling is performed in a single stage. This means, data shuffling is only done between different stages. Each stage consists of multiple tasks that applies the same processing algorithm/s on different data blocks. The number of tasks in the first stage is determined by the number of input data blocks distributed over all machines. Thus, the number of tasks in stages remains the same, or changes proportionally.

Latency analysis. We discuss some aspects that influence system latency from multiple perspectives:

- Increasing the input data size increases the number of data input blocks, thus, increases the number of tasks per stage. In addition, more tasks are assigned to each processing machine.
- More tasks in a stage cause additional data shuffling overhead. For example, if we have 4 tasks per stage then 4×4 data block transfer operations are required to shuffle data blocks between two stages (each has 4 tasks). Increasing the number of tasks to 5 per stage, it means that 5×5 data block transfer operations are required instead.
- Increasing the number of processing machines, it means that lower amount of tasks are assigned to each processing machine.
- Increasing the number of processing machines, it means an increase of data transfer overhead, while more data blocks need to be exchanged over network. For example, if two machines are allocated, then around 50 % of data blocks need to be transferred over a network. 66 % of data blocks need to be transferred if three machines are allocated, and similar.
- Increasing the number of processing machines also increases the total amount of memory that can be used for execution and caching intermediate results. In principle, this reduces the overall processing time.

3. RESOURCE-CONSTRAINED CLUSTERS

Performance prediction of heterogeneous big data applications already received considerable attention from the research community and lead to the development of effective performance prediction frameworks, such as Ernest [20]



(a) *PageRank* (b) *WordCount*

Figure 1: Ernest prediction with different design of experiments strategies.

or CherryPick [5]. They employ sampling-based black-box prediction approaches, which capture application characteristics on sample runs running in small-sized clusters (between 1 to 16 machines), and actual runs running in medium-sized clusters (between 40 and 100 machines).

An analysis of private cluster infrastructures, however, reveals that most clusters often have far fewer machines, often less than 20 [20, 5]. We refer to such clusters as *small* or *resource-constrained* clusters, although there is no clear boundary between small and medium-sized clusters. We base this on the observation that oftentimes such clusters face resource allocation & scheduling challenges due to constrained resources (mostly computing power and main memory capacity) [13]. Our initial analysis of performance prediction accuracy and overhead measurements using the state-of-the-art prediction framework Ernest, whose main target cluster sizes are medium-sized clusters with 40–100 machines, reveals unique challenges for performance predictions on resource-constrained clusters that are not yet well captured by state-of-the-art approaches.

Impact of DoE strategy. To evaluate the impact of the DoE strategy, we selected *PageRank* as an example application and computed the prediction accuracy using Ernest based on 7 sample runs as suggested by the optimal DoE strategy[1] of Ernest on a 12-machine cluster (cf. Section 6 for details). In addition, we use a DoE *oracle* (cf. Section 4), which selects the optimal sample runs in a brute-force fashion by running the full factorial of possible sample runs first. Figure 1a depicts the results of the Ernest performance prediction model in combination with the Ernest DoE strategy (▯▯) and our DoE *oracle* (▮▮). The DoE *oracle* does not only select fewer sample runs (3 versus 7 for the Ernest DoE), but also enables the Ernest performance prediction model to achieve a higher average prediction accuracy of 96 % compared to 59 % with the Ernest DoE strategy and exhibits a 49 % lower sample run overhead.

Impact of performance prediction model. Even if the DoE strategy produces the optimal sample runs, a performance prediction model might still produce a poor execution time prediction accuracy. To demonstrate this, we selected *WordCount* as an example application and used two different DoE strategies in conjunction with the performance prediction model of Ernest. Figure 1b shows the predicted execution times using the optimal DoE strategy from Ernest (▯▯) and our DoE *oracle* (▮▮). The main reason of the low prediction accuracy of the Ernest performance prediction model is because it includes the overhead caused by adding new machines (linear and logarithmic), but it does not include or address the benefits of adding new machines with respect to the capacity extension of memory for execution and storage. In Ernest sample experiments are performed in a small-sized cluster (1–16 machines), whereas the actual runs take place in larger clusters (45–64 machines). Consequently, during the actual runs, the total memory capacity is larger and situations with high memory pressure become less likely.

Based on these initial experiments on a resource-constrained cluster with 12 machines, we see evidence that the optimal DoE strategy and the execution time prediction model need to be adapted to optimally cover such small, resource-constrained clusters. Our focus in the context of this paper is the generalization of a prediction model towards covering resource-constrained clusters. Therefore, to ensure that our approach is generic, we evaluate our approach on all data-intensive applications of HiBench. To be able to separate the optimal DoE strategy from the prediction model, we initially propose a DoE *oracle* (cf. Section 4) and focus in the remainder of the paper on the description of the prediction model.

4. DESIGN OF EXPERIMENTS ORACLE

Performance prediction frameworks for big data applications typically come in a bundle consisting of a DoE methodology and a corresponding prediction model. As we have shown in Section 3, the DoE methodology and the prediction model can individually but also in conjunction have a dramatic impact on the overall prediction accuracy. To isolate the DoE methodology from the prediction model, we use a DoE *oracle* and focus in the following on the development of a generalized prediction model. This enables us to evaluate different prediction models independent of their respective optimal sampling strategies. By using the DoE *oracle*, we evaluate multiple prediction models, until a final model is proposed in the paper. Being space limited, we do not evaluate and compare between these models in Section 6, rather we emphasize on the prediction model shown in Section 5.1 (cf. Equation 1).

We carry out sampling experiments by taking into account two configuration parameters, namely the number of machines and the data scale (12 and 10 distinct options, respectively). Our DoE *oracle* aims at minimizing the overhead of running the sampling experiments while capturing sufficient characteristics of the application to ensure a high prediction accuracy.

Typically, DoE strategies suggest that a static set of experiments should be carried out for all applications. Due

[1]`https://github.com/amplab/ernest`, Accessed: 05-08-2020

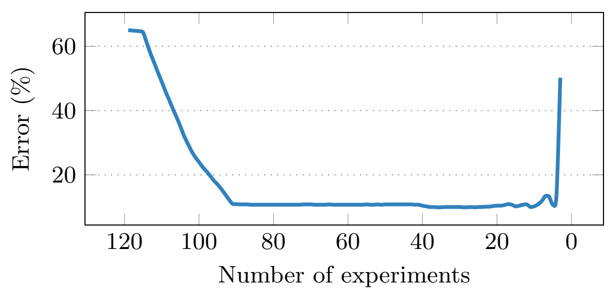


Figure 2: Model error of *Linear Regression* by removing sample experiments.

to the heterogeneity of advanced analytics applications and their resource requirements (e.g., I/O-, CPU-, or network-intensive), these aspects have to be taken into account when developing an optimal DoE methodology. For example, to capture the characteristics of data-intensive applications, running sampling experiments with large data scales on few machines is beneficial since it stresses and captures memory limitations (e.g., data spilling to disk) even during sample runs. Similarly, sample configurations with small data scales on large clusters allow capturing latency delays induced by the networking stack. Our DoE *oracle* relies on the following steps:

1. Run the application on the full data scale and all cluster configurations. In total, there are 12 actual runs of 100 data scale on 1–12 machines.
2. Apply the sampling phase by using a full factorial design of experiments. In total, there are 120 sample runs of 1–10 data scales on 1–12 machines.
3. In iterative manner, remove sample experiments one by one until 3 experiments are reached. At every iteration, one sample experiment should be removed. This is done by having an internal iteration over remaining experiments. In each internal iteration, the experiment is removed from the list and the model is trained using the remaining experiments, relying on the execution time model (cf. Section 5.1). The point whose removal causes the highest model score, relying on the score function (cf. Section 5.2), is removed. The outcome of each iteration is a list of sampling experiments and the prediction accuracy value. Three experiments are selected to be the lower bound of experiments because the features of the performance prediction model (cf. Section 5.1) comprise non-linear functions. Therefore, two experiments are expected to be very few.
4. Select the best sampling experiments list containing at most 10 experiments and with the highest efficiency score (model accuracy (cf. Section 5.1) to the sampling cost (cf. Section 5.3) ratio).

We take Linear Regression (LIR) as an illustration application example. Initially, we run LIR on the full data scale on 12 different configurations (1–12 machines). Next, we run 120 greedy (full factorial) sample experiments (data scales 1–10 on 1–12 machines). In a subsequent step, we remove the collected data points again one by one. Figure 2 illustrates the impact of removing data points (i.e., sample experiments) on the error (cf. Section 5.2) of the performance prediction model (cf. Section 5.1). We observe that the process of removing sample experiments can be divided into three phases:

Phase 1 **(Removal of outliers)** Initially, the model accuracy improves when we remove the first batch of sample experiments. Most of the removed experiments are short-running and can cause a large execution time variance, which could affect the overall model accuracy.

Phase 2 **(Removal of unnecessary experiments)** In this phase, the model accuracy remains stable while reducing the number of sample experiments limits the sampling overhead.

Phase 3 **(Removal of useful experiments)** In this phase, the model accuracy is directly affected by the removal of sample experiments. Thus, a trade-off has to be made between high model accuracy and low sampling overhead. In our experimental analysis, this behavior becomes noticeable when less than 7 sample experiments are selected.

Table 1: Model accuracy versus sampling overhead ratio in *Linear Regression*.

#Experiments	Accuracy	Overhead	Accuracy:Overhead
3	49.8	135.2	0.37
4	**86.1**	**171.6**	**0.5**
5	89.2	225.1	0.4
6	86.8	273	0.32
7	86.6	366.9	0.24
8	88.2	386.3	0.23
9	89.2	412.9	0.22
10	89.8	455.6	0.2

Table 1 shows how the optimal sampling configuration with 4 experiments was selected for LIR based on the *model accuracy / sampling overhead* ratio. The model accuracy and sampling overhead are calculated using Equation 2 and Equation 4, respectively. As such, the DoE *oracle* facilitates the separation of the DoE and the performance prediction model. In the remainder of the paper, our focus is on the performance prediction model and we leave the development of a more general DoE for future work.

5. APPROACH

To predict the execution time of an application on various cluster configurations (i.e., the number of machines), we run sample experiments by tuning two configuration parameters, namely the number of machines and the data scale. We refer to the data scale by a percentage of the full data input. For example, data scale 10 refers to 10 % of the full data scale. While running sample experiments, we tune the data scale parameter within a range of 1–10. We train the execution time model with the selected experiments and calculate the model coefficients. Finally, the model can predict the execution time of the application on the full data scale and for different cluster configurations (number of machines).

5.1 Execution time model

The execution time of an application consists of the following main parts, accordingly to the issues specified in Section 2.2:

- Serial part. A constant overhead (e.g., startup delays caused by cluster resource allocation and query planning).

- Parallel part. Adding more machines reduces the execution time of the application for two reasons: (1) The number of tasks per machine s/m is lower, where s is the scale, m is the number of machines, and (2) the total amount of memory allocated for the application will be higher, and thus, less cached intermediate result data partitions will be evicted. In addition, more amount of memory will be for data processing, which means less data management overhead (e.g., GC overhead). Experimentally, we could verify that the term $(s \times log(s))/m$ represents this behavior well.
- Overhead. Adding more machines to the cluster linearly increases the data transfer overhead during shuffling. In addition, more data results in more data blocks being written to a distributed file system, which subsequently results in more tasks per stage. That means, more tasks output shuffling data blocks between stages. Therefore, the complexity of shuffle cost is $O(n^2)$, where n is the data scale.

Following the discussion of the main parts of an application, our proposed execution time prediction model is as follows:

$$R = \theta_0 \times \frac{s}{m} + \theta_1 \times s^2 + \theta_2 \times m + \theta_3 \times \frac{s \times log(s)}{m} \qquad (1)$$

where R is the execution time, s is the scale, and m is the number of machines. In addition, the model coefficients θ_0, θ_1, θ_2, and θ_3 capture application-specific characteristics. The execution time model does not have a fixed delay that represents the serial computation. We omit this part since we assume the input data to already reside in a distributed file system, which allows applications to directly read data blocks after start up. In practice, however, there is a constant overhead during the startup phase, which can be attributed to the resource manager negotiating the resource allocation. In our experiments, this constant overhead was up to 40 seconds, which are negligible for long-running applications running for multiple hours.

To train the model, we use the *curve_fit* solver [1] with enforced positive bounds, which avoids getting negative coefficients. This approach achieves the same results as the *non-negative least squares* (NNLS) solver [2], but provides more flexibility for testing non-linear, advanced models.

5.2 Score function

To evaluate our execution time prediction model, we compare the actual and the predicted execution time for each configuration individually. We define a score function, which defines for every cluster configuration the error value E:

$$E = \frac{\sum_{c=1}^{NC} \frac{ABS(AR(c) - PR(c)) \times 100}{AR(c)}}{NC} \qquad (2)$$

where NC is the number of cluster configurations (number of machines), AR is the actual execution time, and PR is the predicted execution time. The final score is the average of all error values E for every cluster configuration.

In modifying the cluster, configurations influence the application execution time significantly. We selected the current scoring function instead of a mean square error function, because the latter one significantly hides the miss predictions. This is easily apparent in cluster configurations (e.g., with 12 machines) when the application execution time is relatively short. For example, actual runs in different configurations took 1 hour on one machine and 10 minutes on 12 machines. If the predicted values were 1 hour on a single machine and 20 minutes on 12 machines, our proposed score function returns an error value of 25 %.

5.3 Cost function

To define the cost of a sample experiment C_e, we use the total cluster occupation time, which we calculate as follows:

$$C_e = m_e \times r_e \qquad (3)$$

where m_e is the number of machines and r_e is the execution time of an experiment e. The cost of all sampling experiments CS is the sum of the total occupation time of all experiments (i.e., the machine minutes):

$$CS = \sum_{e=1}^{n} C_e \qquad (4)$$

where n is the number of experiments and C is the cost of a single experiment. In order to quantify the cost of the entire set of sampling experiments, we compare the cost of all experiments with the cost of all actual runs. That is the sum of all actual total occupancy time runs, with all different configurations on the full data scale, $\sum_{c=1}^{NC} AR(c) \times c$, where NC is the number of cluster configurations and $AR(c)$ is the actual execution time of the application, running on c number of machines.

6. EXPERIMENTAL EVALUATION

We evaluate MASHA by means of its prediction accuracy and sampling overhead. In addition to comparing with the actual execution times, we also compare MASHA with ERNEST. Furthermore, for the comparisons, ERNEST is deployed in two modes, ERNEST with its proper DOE strategy and with DOE *oracle* (cf. Section 4). Afterward, we identify these two ERNEST modes as ERNEST and ERNEST DOE *oracle*, respectively.

For evaluation, we use the full set of HIBENCH (i.e., 16 different applications) representing 3 different application categories (ML, Graph, Micro) with diverse job topology characteristics (cf. Table 2). Intentionally we exclude HIBENCH SQL applications, because they only consist from a set of mere query operations. All model accuracy and cost results presented in this section are derived from Equation 2 and Equation 4, respectively.

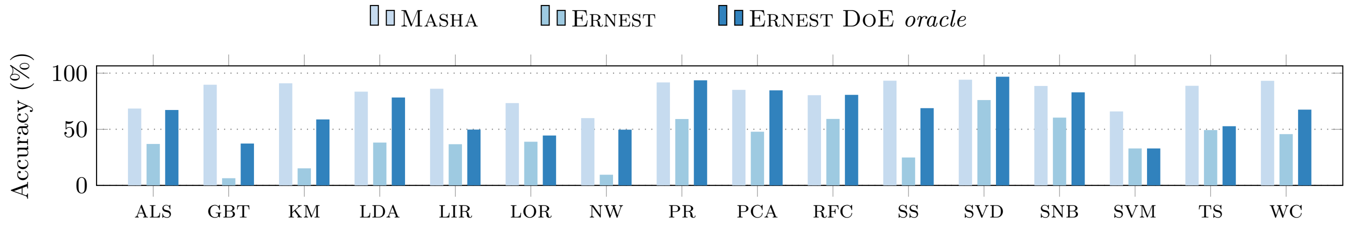
We ran all experiments on a private 12-node Apache Spark cluster, which consists of the following node specifications: Intel Core i5 CPU running at 4x 2.90 GHz, 16 GB DDR3 RAM, 1 TB disk, and 1 GBit/s LAN. The cluster runs Hadoop 2.7, Spark 2.4.0, Java 8u102, and Apache Yarn [19] on top of HDFS [17].

6.1 Zoom-out

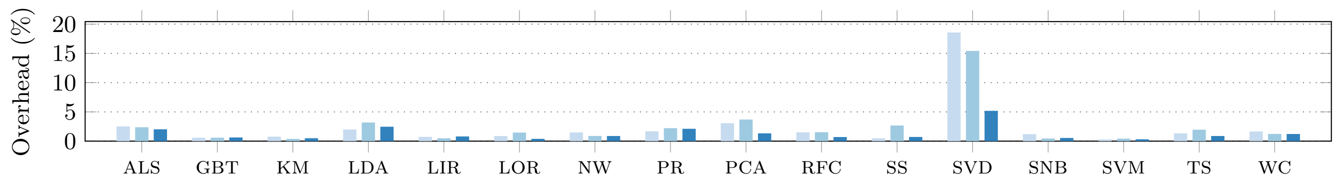
In this section we present an overall prediction accuracy evaluation of MASHA with respect to the actual values and its baseline, ERNEST and ERNEST DOE *oracle* over the entire set of the applications (cf. Figure 3a). In 75 % of the applications, MASHA shows a prediction accuracy of more than 80 %. Its overall average prediction accuracy is 83 %, and on average it outperforms ERNEST by 43.5 %, and ERNEST DOE *oracle* by 18.5 %. We observe the smallest difference of prediction accuracy between MASHA and ERNEST for *SVD*

Table 2: Overview of evaluated HiBench applications.

Application	**Type**	**Topology**			**Model Coefficients**				**Duration**	**Data**
		# Jobs	# Stages	# Tasks	θ_0	θ_1	θ_2	θ_3	Single machine (hrs)	(GB)
Alternating Least Squares (ALS)	ML	53	101	29103	13.68	0.00	30.20	11.60	5.89	13.39
Gradient Boosting Trees (GBT)	ML	712	1382	408620	27.43	1.16	32.10	37.80	12.45	0.61
Kmeans (KM)	ML	14	20	6000	0.00	0.00	0.00	68.96	7.66	112.22
Latent Dirichlet Allocation (LDA)	ML	27	38	2557	18.37	0.00	32.12	13.84	1.92	1.9
Linear Regression (LIR)	ML	5	7	28777	287.31	1.53	9.18	0.00	10.09	894.41
Logistics Regression (LOR)	ML	44	83	13439	0.00	0.00	44.63	31.57	7.97	37.21
NWeight (NW)	Graph	1	9	4500	0.00	0.05	5.00	6.52	2.82	1.37
PageRank (PR)	Graph	1	6	1788	27.67	0.00	5.49	21.65	3.13	18.56
Principal Components Analysis (PCA)	ML	209	411	64437	208.63	0.00	348.02	0.00	5.44	2.24
Random Forest Classifier (RFC)	ML	9	14	46489	851.82	0.00	103.92	0.00	22.67	223.52
ScalaSort (SS)	Micro	1	2	2400	0.00	0.14	7.00	17.10	2.34	153.98
Singular Value Decomposition (SVD)	ML	2	3	893	216.39	0.00	72.29	0.00	5.84	53.68
Sparse Naive Bayes (SNB)	ML	9	11	56320	508.40	0.00	0.00	62.64	21.09	350.77
Support Vector Machine (SVM)	ML	107	209	285804	0.00	0.00	3.89	289.71	43.15	167.63
TeraSort (TS)	Micro	2	3	5760	103.61	0.45	2.07	0.00	4.90	240
WordCount (WC)	Micro	1	2	9600	69.07	0.20	3.78	0.00	2.74	615.92



(a) Average prediction accuracy

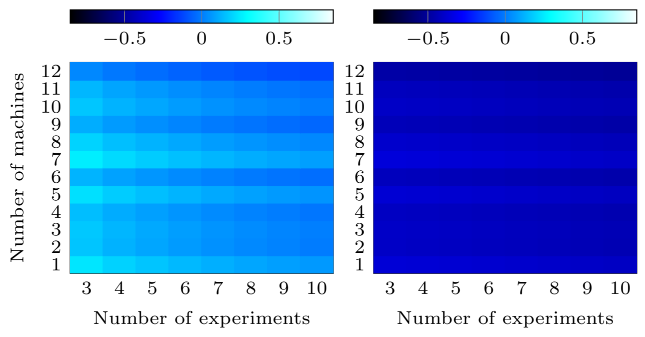


(b) Overhead measurements

Figure 3: Average prediction accuracy & overhead measurements on the entire set of applications.

(18 %), whereas the highest difference of prediction accuracy is for *GBT* with 83.3 %. In comparison to Masha and Ernest DoE *oracle*, we observe the smallest difference of prediction accuracy for *RFC* (0.3 %), whereas the highest difference of prediction accuracy is for *GBT* with 52.4 %.

Statistically, for 37.5 % of the evaluated applications, Masha needs 5 sampling points to obtain the best prediction accuracy. For 87.5 % of the applications, it needs between 3–5 sampling points, and for the entire set of applications (i.e., 100 %), it needs between 3–6 sampling points. In other words, this shows that Masha reaches its prediction accuracy with a reasonable amount of sampling points, considering long-running applications.



(a) Masha (b) Ernest

Figure 4: Zoom-in analysis of the *KMeans* application.

6.2 Zoom-in

In this section, we select *KMeans* as an exemplary application to show a more detailed analysis. As can be seen in Figure 4, *KMeans* is by no means the application in which we observed the best prediction accuracy of Masha.

In *KMeans*, on a single machine, the prediction accuracy of Masha is around 85 % and the prediction accuracy of Ernest and Ernest DoE *oracle* is 16 % and 62.5 %, respectively. In this case, the improvement of Masha over Ernest and Ernest DoE *oracle* is 68.6 % and 22 %, respectively. As the number of machines increases, Masha reaches a prediction accuracy of 96.7 % (on 9 machines), whereas the prediction accuracy of Ernest reaches 16.4 % (on 7 machines), and ERNEST DoE *oracle* reaches 63.4 % (again, on 7 machines) at their best case. The worst prediction accuracy by MASHA is 83 % on 7 machines. The worst prediction accuracy shown by ERNEST is 13.5 % on 12 machines. The worst prediction accuracy shown by ERNEST DoE *oracle* is 52 %, again on 12 machines. The smallest difference prediction accuracy between MASHA and ERNEST and ERNEST DoE *oracle* is observed on 7 machines, where MASHA shows 66.7 % and 19.7 % improvement over ERNEST and ERNEST DoE *oracle*, respectively. The biggest difference prediction accuracy between MASHA and ERNEST and ERNEST DoE *oracle* is observed on 12 machines, where MASHA shows 82 % and 43.8 % improvement over ERNEST and ERNEST DoE *oracle*, respectively. Overall, the average prediction accuracy of MASHA is 91 %, compared to 15 % and 58.8 % average prediction accuracy for ERNEST and ERNEST DoE *oracle*, respectively. This means, on average, MASHA shows 75.8 % and 32 % improvement over ERNEST and ERNEST DoE *oracle*. It is not difficult to notice that, DoE *oracle* also improves ERNEST prediction accuracy with 43.6 %. Further details to this later improvement (not just for *KMeans*, but also for additional applications) will follow in Section 7.2.

As the prediction accuracy of MASHA mostly overestimates the actual running values, ERNEST's prediction accuracy is negative, by underestimating the actual running values in each number of machines. Furthermore, whereas MASHA shows mixed and inclined towards overestimating the actual running values, ERNEST, in all the experiments, independently to the number of samplings and machines, shows a negative prediction accuracy, by means of underestimating the actual running values (cf. Figure 4). The main ERNEST execution time model, did not address the memory limitation behavior, alike we did in Equation 1. Therefore, ERNEST always underestimates the actual running values. In fact, this is also the reason why in Figure 4b we represent only ERNEST, because both, ERNEST and ERNEST DoE *oracle* are oriented towards the very same inclination, that of underestimating the actual running values.

6.3 Overhead measurements

In Figure 3b we show the overhead measurements of MASHA in comparison to ERNEST and ERNEST DoE *oracle*. The average overhead of MASHA is 2.42 %, whereas for ERNEST is 2.44 % and for ERNEST DoE *oracle* is 1.3 %. This indicates that MASHA improves the prediction accuracy, by means of having a similar overhead compared to ERNEST. Nevertheless, ERNEST DoE *oracle* shows better overhead, even if does not improve the prediction accuracy as MASHA.

The very single application that has shown a high overhead in MASHA is *SVD* with 18.6 %. For the very same application, the overhead in ERNEST is also high (15.4 %). For the very same application, again ERNEST DoE *oracle* shows the smallest overhead, with only 5.2 %. We explain the reason to this phenomenon in Section 7.1. When excluding *SVD* numbers, the average overhead for MASHA, ERNEST and ERNEST DoE *oracle*, descends further to 1.34 %, 1.6 % and 1 %, respectively. Comparing MASHA and ERNEST, in *GBT* we observe the smallest relative overhead (0.006 %), whereas in *SVD* we observe the highest relative overhead (3.16 %). Comparing MASHA and ERNEST DoE *oracle*, in *SVM* we observe the smallest relative overhead (0.023 %), whereas again in *SVD* we observe the highest relative overhead (13.4 %).

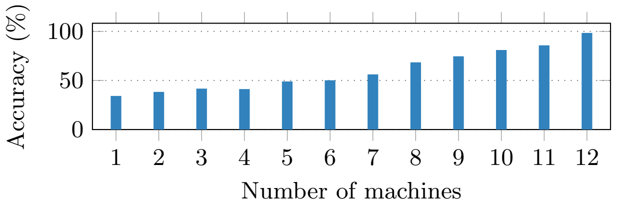


Figure 5: Performance prediction of *NWeight* in details.

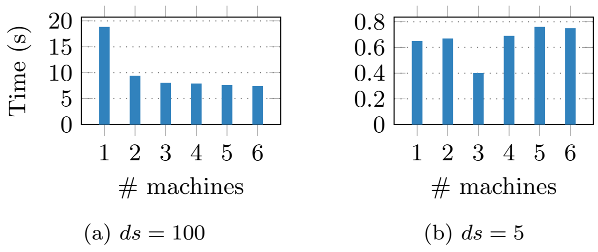


Figure 6: Garbage collection overhead per task in *NWeight*.

7. DISCUSSION

In the following we discuss main findings that were observed during our experimental results regarding MASHA (cf. Section 7.1) and ERNEST (cf. Section 7.2).

7.1 Masha accuracy and overhead

Herein, we discuss our observations from running performance predictions on a large variety of different applications from HIBENCH, in particular cases where MASHA performs poorly with respect to *prediction accuracy* (e.g., low prediction accuracy for applications *NWeight* and *SVM*; cf. Figure 3a) and *sampling overhead* (e.g., high overhead for application *SVD*; cf. Figure 3b).

Low prediction accuracy. For *NWeight*, the performance prediction of MASHA (and any other black-box, sampling-based approach) exhibits a low accuracy for running on few (<5) machines—with lowest accuracy of 34 % when running on a single machine—and a high accuracy (up to 98 %) for up to 12 machines (cf. Figure 5). The reason for this behavior is caused by an increased memory pressure when running the application on the full data scale on only few machines. With a sampling-based approach (i.e., 1–10 % of the full data scale), however, potential memory limitations that would occur during a run on full data scale, cannot be easily captured or even be predicted on small data scales.

To validate this hypothesis, we collected JVM garbage collection (GC) overheads for the *NWeight* application and show the results in Figure 6. A high JVM GC overhead is an indicator for a large number of object creations & evictions in the JVM heap, which in turn is a result of high (execution) memory pressure (e.g., used for computing and shuffling). We selected the stage within *NWeight*, which consumes 50 % of the total application execution time. For a small data scale of 5 % (cf. Figure 6b), the overhead for GC remains almost constant, independent of the number of machines. In contrast, on the full data scale (cf. Figure 6a), the GC significantly increases when we decrease the number of machines.

Another potential memory limitation concerns storing intermediate results (i.e., data partitions). In sample runs on small data scales, such cached data partitions often entirely fit into main memory, effectively eliminating the overhead for reading the data partition from a slower storage medium. To validate this hypothesis, we analyze a representative, iterative stage of the *SVM* application on four different sample configurations and present the results in Figure 7. On the 10 % data scale, 2 machines have sufficient memory capacity to store all data partitions entirely in memory. In contrast, on a single machine, 38 % of data partitions were evicted from memory and had to be recomputed, which in turn causes a large performance penalty. On the 10 % data scale, for *SVM* this results in a dramatic execution time difference between 112 minutes on 1 machine vs. 4 minutes on 2 machines. In order to include the cache eviction behaviour during the sample runs, again we evaluated *SVM* after double sizing the input data, and consequently we realized an improvement of the prediction accuracy (82.3 %).

High sampling overhead. Compared to all other applications, *SVD* has with about 20 % the highest overhead. For *SVD*, our DoE *oracle* selects 6 sample experiments (out of 120), where each individual experiment comes with high total cluster occupation time for running them (2 experiments on 12 machines, 2 experiments on 11 machines, 1 experiment on 10 machines, and 1 experiment on 9 machines). Here, our DoE *oracle* incurs a higher overhead than the DoE in Ernest, even though the number of selected sample experiments is lower, but the selected sample experiments by themselves take more time to run.

A second observation stems from an inspection of the derived model coefficients in *SVD* (cf. Table 2). From the model coefficients we can conclude that the overhead part caused by increasing the number of machines ($\theta_2 = 72.3$) dominates the overhead parts caused by increasing the data scale ($\theta_1 = 0$, and $\theta_3 = 0$). In such cases, the sampling overhead is high compared to the actual runs on the full data scale due to the missing overhead caused by larger data scales.

7.2 Ernest versus Ernest DoE oracle

Herein, we discuss our observations in cases where Ernest exhibits low prediction accuracy due to its DoE strategy and its prediction model.

In Table 2, we can observe that the main reason of poor prediction accuracy in Ernest comes mainly from the DoE component. The average prediction accuracy of Ernest is 40 %, whereas the average prediction accuracy of Ernest DoE *oracle* is 65 %. By excluding *SVM* and *NW* (cf. Section 7.1), it can also be observed that the prediction model of Ernest performs well in the following 7 applications (i.e., 50 % of the total number of applications) compared to Masha: *ALS*, *LDA*, *PR*, *PCA*, *RFC*, *SVD* and *SNB*. The average accuracy of Ernest DoE *oracle* in these applications is 83.5 %. In the remaining applications, the prediction accuracy of Ernest DoE *oracle* is much lower than in Masha. The average accuracy of Ernest DoE *oracle* for the remaining applications is 54 %. From this, we can conclude that the Ernest prediction model is not sufficiently general to cover a wider set of applications compared with the presented Masha performance model (cf. Section 5.1).

Low accuracy caused by DoE. To get more insights into the low performance of the DoE strategy presented by Ernest, we observed that in those applications on which Ernest showed improvements when coupled with our DoE *oracle*, the respective applications contain many outliers. The reason is, most of these applications have short-running

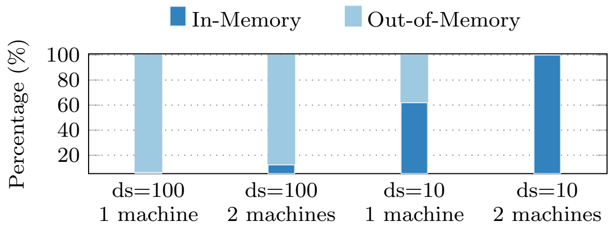


Figure 7: Details of the iterative stage in *SVM*.

sample experiments. Even if Ernest can detect these outliers by means of using its presented cross-validation evaluation of sample experiments, however, it cannot handle them in a way that improves the overall prediction accuracy.

Low accuracy caused by prediction model. We observed the 7 applications in which Ernest prediction model resulted high accuracy. From Table 2, we can notice that in all these applications $\theta_1 = 0$. However, we can observe that also *KM* and *LOR* have $\theta_1 = 0$, but have low accuracy. The reason behind this is that *KM* and *LOR* have high θ_3 coefficients (68.96 and 31.57, respectively). Whereas θ_1 represents the shuffle cost overhead, θ_3 represents the overhead caused by lack of memory. In other words, both overheads are not addressed in the Ernest prediction model.

8. RELATED WORK

There has been extensive number of contributions that have observed, analyzed, and predicted the execution time of data-intensive applications running on distributed systems.

Starfish [8] introduced a self-tuning framework on top of Apache Hadoop that applies an analytical approach to analyze MapReduce jobs execution time metrics, by running them on a data fraction and optimize system performance by tuning its configuration options. PREDIcT [15] is an experimental methodology to predict the execution time of a class of iterative algorithms. Its main idea is to predict the number of iterations and the execution time of each iteration depending on sample runs. Sidhanta et al. [18] have proposed OptEx, a model for estimating the job execution time on Apache Spark, under a given service level objective deadline with near to optimal cluster configuration cost. Alipourfard et al. have proposed CherryPick [5], a performance prediction model that is able to select near to optimal configurations, by means of leveraging Bayesian Optimization and applying an adaptive sampling approach to reduce the sampling overhead. Marco et al. [13] model the memory behavior of Apache Spark jobs based on a mixture of experiments. Based on the extracted models, they present execution time prediction models and propose a task co-location strategy to improve the system throughput. Doppio [22] is an execution time prediction model for Apache Spark jobs, by means of studying the I/O impact on in-memory cluster computing frameworks. Their observation identified I/O overhead as a dominant bottleneck in such frameworks. Al-Sayeh et al. [4] presented a graybox model for predicting the application execution time on Apache Spark, by considering different computing resources and selected application parameters based on prior knowledge of application internals.

Among others, the superiority of Masha lies in the generic application of its methodology, an intersection of a DoE *oracle* with a sampling-based prediction model, high prediction accuracy in resource-constrained clusters, and a broad coverage of different applications.

9. CONCLUSION & OUTLOOK

MASHA is fostered to help application users that need to execute long-running applications in resource-constrained cluster environments. It does this, by giving another perspective of experiment design and prediction model, which outperforms other state-of-the-art approaches. We do not claim to have a Swiss knife prediction model, as it was observed in the discussion section. However, the preliminary results are promising, and encourage a future work. Initially, we plan to study the carried out sampling measurements of each application and define a general DOE approach. In addition, while we have the model coefficients of each application, we can analyze these coefficients to classify data-intensive applications, analyze their bottlenecks, and further characterize them. Finally, we want to explore MASHA on data infrastructures that go beyond small-scale clusters.

Acknowledgements. We would like to thank the anonymous reviewers for their helpful comments and suggestions. Hani Al-Sayeh was supported by funding from the German Aerospace Center.

10. REFERENCES

[1] CURVE_FIT SOLVER. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html`. Accessed: 05-08-2020.

[2] NNLS SOLVER. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.nnls.html`. Accessed: 05-08-2020.

[3] THE HIBENCH SUITE. `https://github.com/Intel-bigdata/HiBench`. Accessed: 05-08-2020.

[4] H. Al-Sayeh, S. Hagedorn, and K.-U. Sattler. A gray-box modeling methodology for runtime prediction of Apache Spark jobs. *Distributed and Parallel Databases*, pages 1–21, 2020.

[5] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482, Mar. 2017.

[6] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink^{TM}: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[7] T. R. Devine, K. Goseva-Popstojanova, and M. McLaughlin. Detection of dispersed radio pulses: a machine learning approach to candidate identification and classification. *Monthly Notices of the Royal Astronomical Society*, 459(2):1519–1532, 2016.

[8] 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 2011, Fifth Biennial Conference on Innovative Data Systems Research*, pages 261–272, 2011.

[9] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pages 41–51.

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, Mar. 2007.

[11] N. Kussul, M. Lavreniuk, S. Skakun, and A. Shelestov. Deep learning classification of land cover and crop types using remote sensing data. *IEEE Geoscience and Remote Sensing Letters*, 14(5):778–782, 2017.

[12] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[13] V. S. Marco, B. Taylor, B. Porter, and Z. Wang. Improving Spark Application Throughput via Memory Aware Task Co-location: A Mixture of Experts Approach. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, Middleware'17, pages 95–108, 2017.

[14] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun. Making sense of performance in data analytics frameworks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 293–307, 2015.

[15] A. D. Popescu, A. Balmin, V. Ercegovac, and A. Ailamaki. PREDIcT: Towards Predicting the Runtime of Large Scale Iterative Analytics. *Proc. VLDB Endow.*, 6(14):1678–1689, Sept. 2013.

[16] D. Quang, Y. Chen, and X. Xie. Dann: a deep learning approach for annotating the pathogenicity of genetic variants. *Bioinformatics*, 31(5):761–763, 2015.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[18] S. Sidhanta, W. Golab, and S. Mukhopadhyay. OptEx: A Deadline-Aware Cost Optimization Model for Spark. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 193–202, May 2016.

[19] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing (SoCC 2013)*, pages 1–16, 2013.

[20] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 363–378, Mar. 2016.

[21] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10. USENIX Association, 2010.

[22] P. Zhou, Z. Ruan, Z. Fang, M. Shand, D. Roazen, and J. Cong. Doppio: I/O-Aware Performance Analysis, Modeling and Optimization for In-Memory Computing Framework. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 22–32, 2018.