Best VM Selection for Big Data Applications across Multiple Frameworks by Transfer Learning

Yuewen Wu
wuyuewen@otcaix.iscas.ac.cn
Institute of Software, Chinese Academy of Sciences
Beijing, China

Heng Wu*
wuheng@iscas.ac.cn
Institute of Software, Chinese Academy of Sciences
Beijing, China

Yuanjia Xu
Yi HU
{xuyuanjia2017,huyi19}@otcaix.iscas.ac.cn
Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China

Wenbo Zhang
Hua Zhong
Tao Huang
{zhangwenbo,zhonghua,tao}@iscas.ac.cn
Institute of Software, Chinese Academy of Sciences
State Key Lab of Computer Sciences, Institute of Software,
Chinese Academy of Sciences
Beijing, China

ABSTRACT

Cloud providers are presented with a bewildering choice of VM types for a range of contemporary data processing frameworks today. However, existing performance modeling and machine learning efforts cannot pick optimal VM types for multiple frameworks simultaneously, since they are difficult to balance model accuracy and model training cost.

We propose Vesta, a novel transfer learning approach, to address this challenge: (1) it abstracts knowledge of VM type selection through offline benchmarking on multiple frameworks; (2) it employs a two-layer bipartite graph to represent knowledge across frameworks; (3) it minimizes training overhead by reusing the knowledge to select the best VM type for given applications. Comparing with state-of-the-art efforts, our experiments on 30 applications of Hadoop, Hive and Spark show that Vesta can improve application performance up to 51% while reducing 85% training overhead.

CCS CONCEPTS

• **General and reference** → **Performance**; • **Computing methodologies** → **Transfer learning**; • **Software and its engineering** → **Cloud computing**; • **Social and professional topics** → *Pricing and resource allocation.*

KEYWORDS

virtual machine, big data application, transfer learning, multiple frameworks

*Corresponding author

ICPP '21, August 9–12, 2021, Lemont, IL, USA

ACM ISBN 978-1-4503-9068-2/21/08.
https://doi.org/10.1145/3472456.3472488

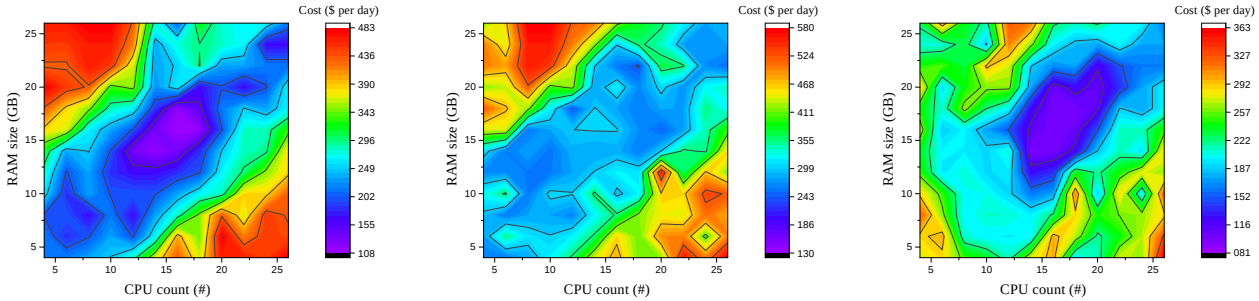
ACM Reference Format:
Yuewen Wu, Heng Wu, Yuanjia Xu, Yi HU, Wenbo Zhang, Hua Zhong, and Tao Huang. 2021. Best VM Selection for Big Data Applications across Multiple Frameworks by Transfer Learning. In *50th International Conference on Parallel Processing (ICPP '21), August 9–12, 2021, Lemont, IL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3472456.3472488

1 INTRODUCTION

Dozens of big data frameworks are available (e.g., Hadoop [22], Hive [24], Spark) in various clouds today, and most users usually choose two or more frameworks for their businesses, and have to face a bewildering choice of Virtual Machine (VM) types (e.g., a1.medium, t2.large). For example, at the time of writing, Amazon, Azure and Aliyun all provide over 100 VM types with varying resource configurations (e.g., CPU, memory, disk, network). Intuitively, jointly optimizing of multiple frameworks will inevitably lead to 10,000+ configurations. In this context, few users can tell which VM type is "best" for their selected applications across multiple frameworks. To make matters worse, they may choose a VM type with 12× extra budget but only get one third of performance (e.g, application execution time) [1].

To address this challenge, existing performance modeling efforts [21, 25, 29] and machine learning approaches [4, 18, 28] have to tolerate huge offline training overhead to build an accurate online model for each framework, since they just consider *low-level metrics* (such as resource utilizations) within a framework. Sadly, they have to spend a lot of time to train new models for similar applications for new frameworks, although recent works [3, 5, 10] have proved that these similar applications, both in Hadoop and Spark, involve a wide range of use cases (micro benchmark, machine learning, stream processing and etc.).

Figure 1 shows an example why we need to tolerate huge offline overhead for a new framework. Hadoop TeraSort, Hive Aggregation and Spark PageRank are applications from different frameworks. Here, we can see their heat maps of budget look completely different. This implies that existing approaches [4, 21, 25, 28, 29] may have to build one model per framework. Fortunately, we observed that



(a) The result of Hadoop TeraSort. (b) The result of Hive Aggregation. (c) The result of Spark PageRank.

Figure 1: Heat map of *budget* of three applications from different frameworks. The horizontal axis shows the number of CPU cores, and the vertical axis shows the memory size (GB). Color changes indicate different costs, the lower the better.

low-level metrics have *high-level similarities* across frameworks (blue area), the best (or near best) VM types all appear in the area which follows similar CPU-to-memory ratio (e.g., 8G8U, 16G16U). Here, we call them **correlation similarities**. As far as we know, the metrics of correlation similarities have not been discussed to choose the best VM types before.

Despite these observations, selecting best VM types for big data applications across frameworks in existing efforts remains rudimentary. In this paper, we use the term **"best VM type"** described in [1, 28] to represent the blue areas in Figure 1, where VM types can achieve optimal or near optimal application performance. Meanwhile, we consider **correlation similarities** as key factors in the selection of the best VM types, and present a transfer learning approach to address this challenge.

We believe our work makes the following advancements:

- we abstract knowledge (correlation similarities) of multiple frameworks by conducting a large-scale evaluation on a wide range of workloads [1] of Hadoop, Hive and Spark.
- we represent knowledge in a two-layer bipartite graph, and reuse knowledge for various workloads across frameworks to minimize prediction error and reduce training overhead.
- we evaluate Vesta by comparing against state-of-the-art approaches, the results show that Vesta can improve performance up to 51% while reducing 85% training overhead.

2 PROBLEM STATEMENT AND MOTIVATION

2.1 Problem statement

In this paper, we measure the selection of best VM type in the learning-based model by evaluating the *training overhead* and the *prediction error*. In particular, learning-based systems usually have two phases:

- **Offline phase**. For a given framework f, we first need to train a offline model for various workloads $\{x_1, x_2, ..., x_i\}$ on VM types $\{t_1, t_2, ..., t_k\}$. Here, we employ function $func(x_i, t_k, f)$ and $cost(x_i, t_k, f)$ to represent the model and the training overhead, respectively.

[1]In this paper, the term workload refers the runtime state of a application, it can also refer to the amount of work (or load) that software imposes on the underlying computing resources.

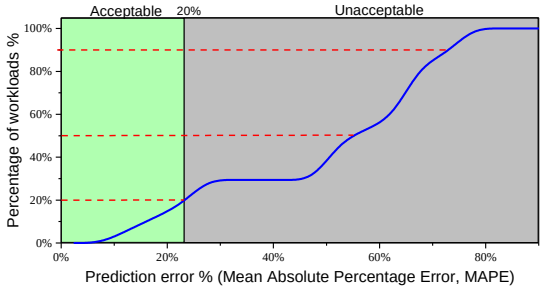


Figure 2: The *prediction error* by reusing a pre-trained model (Hadoop and Hive) based on *low-level metrics* to predict the best VM types for Spark. Nearly 80% of workloads (vertical axis) are suffering from high prediction error.

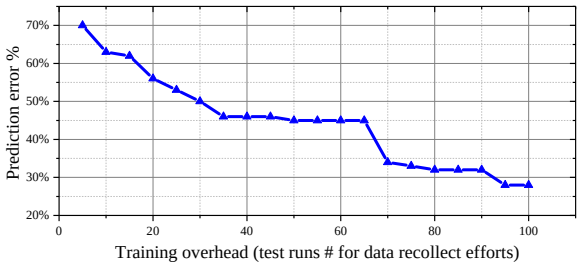


Figure 3: The *training overhead* from scratch for a new framework. This process may take hundreds of hours [7] to achieve acceptable prediction error.

- **Online phase**. For a new workload x' of the same framework f, we can easily get a best VM type from $func(x_i, t_k, f)$.

In this context, if we have another set of workloads $\{x_1^*, x_2^*, ..., x_n^*\}$ from a new framework f', our goal is to reuse data from pre-trained model $func(x_i, t_k, f)$ to accelerate retraining $func(x_n^*, t_k, f')$ with acceptable overhead.

Now the questions are:

- **How to minimize prediction error**. Reducing prediction error can improve application performance. In this paper, it

means we should find a way to minimize the performance difference between the predicted VM type $t_{predicted}$ (found by $func(x_n^*, t_k, f')$) and the ground truth "best" VM type t_{best} (e.g, found by brute-force searching), as shown in Equation 1.

$$\min \left| t_{predicted} - t_{best} \right| \tag{1}$$

- **How to minimize training overhead**. As discussed above, it is exhausting to train from the beginning for new frameworks, and the overhead cannot be accepted in most online scenarios [1, 28]. Reusing pre-trained model is a possible way to minimize overhead, but we need to improve model accuracy with acceptable retraining overhead, as shown in Equation 2.

$$\min cost(x_n^*, t_k, f') \tag{2}$$

2.2 Problem Analysis

High prediction error in low-level metrics. Recent literatures always focus on how low-level metrics affect the prediction error within frameworks [16, 25], such as *AWS Compute Optimizer* on Amazon [2]. However, Figure 2 shows that these efforts may suffer from high prediction error on new frameworks when just reusing pre-trained models. In this paper, we discover new high-level metrics (named **correlation similarities**) and make them key factors of **knowledge** across frameworks (Section III.A).

High training overhead from scratch for a new framework. Figure 3 shows the relationship between training overhead and prediction error for a new framework, and it is exhausting to build one model per framework. To balance these two factors, we first represent knowledge using a bipartite graph (Section III.B), and then reuse it for a new framework with incremental training instead of training from scratch (Section III.C).

3 VESTA DESIGN

Vesta solves the VM type selection problem with three steps: (1) it runs a large-scale evaluation on Amazon EC2 to abstract knowledge; (2) it builds a bipartite graph model to represent knowledge; (3) for workloads from a new framework, it leverages a transfer learning technique to reuse knowledge from previous frameworks.

In this paper, we treat classification (e.g., image classification) as a way of **"knowledge"** [30], and group VM types (considering various workloads for existing frameworks) into several categories. When introducing a new framework with just few training data, it would be helpful if we could transfer the classification knowledge.

3.1 Abstracting knowledge via a large-scale evaluation

In order to study high-level similarities across frameworks, we carry out a large-scale evaluation on Amazon Elastic Compute Cloud (EC2). Specifically, we use two big data benchmarks (BigDataBench [26] and HiBench [15]) to run our tests. These benchmark workloads are chosen to be diverse in terms of their use cases and resource requirements, which can cover different applications currently supported by Hadoop, Hive and Spark frameworks:

- **Micro benchmark.** It includes TeraSort, WordCount, Sort, Count, and etc.
- **Machine learning.** It contains Linear Regression (linear), Logistic Regression (LR), K-Means, Bayes, Principal Components Analysis (PCA), Alternating Least Squares (ALS), Collaborative Filtering (CF), Breadth First Search (BFS), Singular Value Decomposition (SVD), and etc.
- **SQL-like processing.** It includes Select, Join, Scan, Aggregation, and etc.
- **Search engine.** It incorporates PageRank, Index, and etc.
- **Streaming.** It consists of Twitter, PageReview, and etc.

After each test run, we collect 20 low-level metrics that can reflect application's resource requirements, execution features, and other system factors.

- **Resource metrics:** to evaluate the job performance. We collect CPU system, user, idle rate for CPU metrics. RAM, buffer, cache usage rate for memory metrics. Disk read, write rate for disk metrics. Network send, receive, drop rate for network.
- **Execution metrics:** to measure the relationship between input data sizes and job performance. We keep a record of the

Table 1: High-level similarities (correlations) across frameworks.

Correlations	Description
Resource metrics	
CPU-to-memory	A positive [★] correlation probably denotes a heavy computational workload, so it can infer to larger CPU and memory sizes in VM types. A negative correlation means the opposite side.
memory-to-disk	A negative [≀] correlation can represent relatively small data size, and can infer to lower VM memory size and disk bandwidth in VM types. A positive correlation represents the opposite side.
disk-to-network	A positive correlation reveals that the workload exchange data frequently to facilitate remote data storage capabilities, and can infer to higher disk and network bandwidths in VM types. A negative correlation means the opposite side.
buffer-to-cache	A positive correlation reveals that *buffer cache* and *page cache* are two critical memory caches in this workload, and can infer to larger buffer and cache capabilities. A negative correlation means the opposite side.
CPU-to-network	A negative correlation probably means that there are lots of data synchronizations in the workload, and can infer to higher network bandwidths. A negative correlation means the opposite side.
Execution metrics	
iteration-to-parallelism	A positive correlation means that the workload prefers running in a "thin" cluster (more iterations), and a negative correlation means that it prefers running in a "fat" cluster (more parallelism). It can infer to the choice of the number of VMs.
data-to-computation	A positive correlation reveals that the workload has lots of *computation* phases. A negative correlation means the opposite side. It can infer to the choice of CPU cores and CPU rate.
data-to-cycle	A positive correlation means that it may be a data-intensive workload or a compute-intensive workload. A negative correlation means the opposite side. It can infer to the choice of RAM size and RAM type.
disk-to-synchronization	A positive correlation reveals that the workload exchanges data frequently. A negative correlation means the opposite side. It can infer to the choice of disk bandwidth and disk size.
network-to-synchronization	A positive correlation means that the workload transfers data frequently. A negative correlation means the opposite side. It can infer to the choice of network bandwidth.

Note: [★] The positive correlation reveals the relationship between two variables in which both variables move in tandem — that is, in the same direction. [≀] The negative correlation reveals one variable decreases as the other variable increases.

[2]The website of AWS Compute Optimizer: https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/ce-rightsizing.html.

number of tasks in the computation, communication, synchronization steps, and the ratio of data sizes to the number of cycles, iterations, parallelism for each workload run.

After that, we run a correlation analysis for each low-level metrics pair. For instance, we calculate the correlation value between CPU and memory and get the CPU-to-memory correlation with normalized value from -1 to 1.

Table 1 summarizes high-level similarities across frameworks. We first use *Principal Components Analysis* (PCA) to analyze the importance of correlation values, which are divided into different intervals (such as [0.1,0.15] and [0.8,0.85]), and determine which of them is more relevant to find the best VM types for various workloads. We evaluate PCA and correlations in Figure 9 and Figure 10, respectively.

Next, we use the correlation metrics as inputs to train a *K-Means* model to group VM types into several categories, and abstract classification knowledge of selecting the best (or near best) VM types. We choose K-Means since it outperforms many competing algorithms in terms of high accuracy and low overhead [20, 27] with a simple hyperparameter k. We tune the k value for better improvements in Figure 11.

3.2 Representing knowledge in a two-layer bipartite graph

As shown in Table 2 and Figure 4, we employ a two-layer bipartite graph to represent knowledge, which includes a workload-label layer and a label-VM layer. Blue edges in those two layers denote the knowledge we have obtained from existing workloads (source workloads). Our goal is to draw the red edges, which can represent the similarities of workloads from different frameworks (target workloads). As such, we split this process into two steps:

- **Build relationships between workloads and labels.** Each workload can be annotated by at least one label. In this design, some workloads share one or multiple labels which implies they tend to be similar with each other.
- **Build relationships between labels and VM types.** If a target workload has the similar labels with some sources (source and target workloads may from different frameworks), the best VM types of them would have similar features (e.g., 8G8U, 16G16U).

Further, the relationships can be divided into three subgraphs:

- $\mathcal{G}^{(XL)} = X \cup L = \varepsilon^{XL}$, where ε^{XL} represents the edges linking the nodes in X and L;
- $\mathcal{G}^{(X^*L)} = X^* \cup L = \varepsilon^{X^*L}$, where ε^{X^*L} represents the edges linking the nodes in X^* and L;
- $\mathcal{G}^{(LT)} = L \cup T = \varepsilon^{LT}$, where ε^{LT} represents the edges linking the nodes in L and T.

$$\mathcal{G}_{ij}^{(XL)} = \begin{cases} 1 & \text{if workload } x_i \text{ conforms to the label } l_j, \\ 0 & \text{otherwise;} \end{cases} \tag{3}$$

In this design, **knowledge** can be represented as $\mathcal{G}^{(XL)} + \mathcal{G}^{(LT)}$, and the way of **reusing knowledge** can be represented as $\mathcal{G}^{(X^*L)} + \mathcal{G}^{(LT)}$. We determine $\mathcal{G}^{(XL)}$ (blue lines in workload-label layer) in Equation 3, and evaluate $\mathcal{G}^{(LT)}$ (blue lines in label-VM layer) based on the results in K-Means (Section 3.1). In this context, we mainly deal with the following challenge:

- **How to reuse knowledge in $\mathcal{G}^{(X^*L)}$.** A target workload may suffer from data sparse problem when it enters the system [12], and it may be lack of data to create the workload-label relationship $\mathcal{G}^{(X^*L)}$ (red lines in the bipartite graph).

Table 2: Symbols and Notations

Symbol	Notation
$X = \{x_1, x_2, \ldots, x_i\}$	X denotes the set of source workloads and x_i denotes the i^{th} source workload
$X^* = \{x_1^*, x_2^*, \ldots, x_n^*\}$	X^* denotes the set of target workloads and x_n^* denotes the n^{th} target workload
$L = \{l_1, l_2, \ldots, l_j\}$	L denotes the set of labels and l_j denotes the j^{th} label
$T = \{t_1, t_2, \ldots, t_k\}$	T denotes the set of VM types and t_k denotes the k^{th} VM type

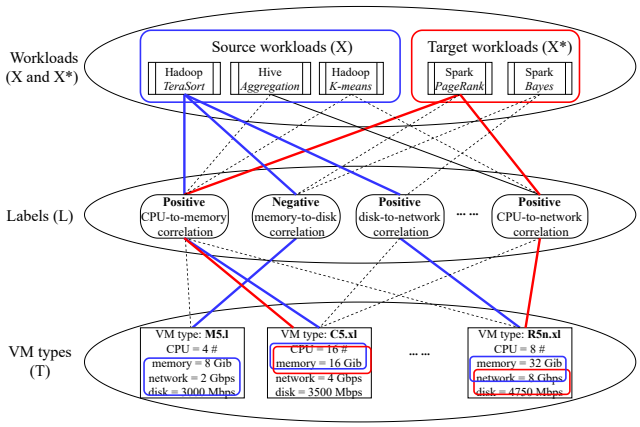


Figure 4: Knowledge across frameworks are represented by a two-layer bipartite graph, where edges represent the relationships among workloads, labels and VM types. The blue solid edges from source workloads reveal the relationships among them, which we treat them as knowledge. The red solid edges from target workloads represent the way of reusing knowledge.

3.3 Reusing knowledge by transfer learning

Based on the knowledge represented as blue edges in Figure 4, we describe how to reuse knowledge (draw the red edges) with relatively small training cost. Suppose that there are a bunch of target workloads X^* from a new framework, we first build the workload-label relationship X^*L in a matrix:

$$U^* = X^* L_1^T \tag{4}$$

where U^* is a matrix expression of workload-label relationship. $X^* \in \mathbb{R}^{n \times g}$, $L_1 \in \mathbb{R}^{j \times g}$, and n, j is the numbers in X^* and L_1 respectively, while g is the latent features. In this case, each workload x_n^* in X^* can be represented by a vector of latent features g, where as g could be constructed by the correlation metrics, input data size, etc.

Similarly, we define a new matrix V to represent label-VM relationship:

$$V = TL_2^T \tag{5}$$

where $T \in \mathbb{R}^{k \times g}$, $L_2 \in \mathbb{R}^{j \times g}$. k, j is the numbers in T and L_2 respectively, while g is the latent features. We suppose that both U^* and V share the same set of labels, which means we may get $L_1 = L_2$ and can use an unified representation of L.

We express knowledge in current frameworks by a matrix $U = XL^T$. Our goal is to reuse the decomposition results on U to help us building the relationships among target workloads, correlation labels and VM types.

In our scenario, the matrix U^* is sparse for a newly entered workload. We employ collective matrix factorization (CMF) proposed by Singh and Gordon [23] to solve this problem. Our objective function can be written as follows:

$$\min_{U,F,U^*} \lambda \parallel U^* - U \parallel_F^2 + (1-\lambda) \parallel U^* - V \parallel_F^2 + R(U, V, U^*) \tag{6}$$

where $0 \leq \lambda \leq 1$ is a tradeoff parameter to control the decomposition error between the two matrix factorization. $\| \cdot \|_F$ denotes the Frobenius norm of matrix [9]. $R(U, V, U^*)$ is the regularization function to control the complexity of the matrices. By calculating the distance between U^* and U, we can decide which $x_i \in X$ are suitable for transfer learning. Once the matrix factorization problem is solved, we can reuse data from X to **reduce training overhead**.

We employ *Stochastic Gradient Descent* (SGD) algorithm [2] to solve this optimization problem. Minimization of *SGD* is performed by fixing two of the matrices and optimizing the remaining one iteratively, until the results have converged.

The algorithm is summarized in Algorithm 1, where we use a set of source workloads, a set of target workloads, a set of labels and a set of VM types as inputs. First of all, we pick a sandbox [3] environment to run the target workload (see Lines 1-3). After that, we abstract and reuse knowledge from source workloads (see Lines 4-11). Finally, we select the best VM types by a combination of several techniques (see Lines 12-14). The converged cost can be estimated by *SGD* algorithm. In the worse case, it may need $O(n \log(n))$ costs to make the result converged [2], and n denotes the amount of training data for the target workload.

4 IMPLEMENTATION

The architecture of Vesta includes offline profiling and online predicting phases, as shown in Figure 5. The offline profiling phase uses the *Data Collector* and the *Correlation Analyser* to run source workloads and abstract classification knowledge, respectively. The online predicting phase uses the *Online Predictor* to predict the best VM types for target workloads by combining various techniques.

4.1 Offline profiling

The goal of offline profiling is to abstract knowledge among existing workloads (aka source workloads) and VM types. According to the previous analysis in Section 3.2, the offline profiling phase mainly accomplish two works:

[3] In here, sandbox VM type means it satisfies the resource requirements of the target workload.

Algorithm 1 Vesta Algorithm

Input:

A set of source workloads $X = \{x_1, x_2, ..., x_i\}$.
A set of target workloads $X^* = \{x_1^*, x_2^*, ..., x_n^*\}$.
A set of labels $L = \{l_1, l_2, ..., l_j\}$.
A set of VM types $T = \{t_1, t_2, ..., t_k\}$.

1: Run source workloads X on each VM type t_i and collect low-level metrics.
2: Pick a sandbox VM type to run a target workload x_i^* for initialization.
3: Analyze the correlations between low-level metrics variables.
4: Group the relationships between labels and VM types via K-Means.
5: Construct matrices $U^* = X^*L^T$, $U = XL^T$, and $V = TL^T$ that reveal the workload-label and label-VM relationships, respectively.
6: Denote the best VM type of target workload x_n^* is t_{best}.
7: **repeat**
8: Fix U and V, apply *SGD* algorithm to update U^*.
9: Fix U^* and V, apply *SGD* algorithm to update U.
10: Fix U and U^*, apply *SGD* algorithm to update V.
11: **until** U, V and U^* are convergent.
12: Accomplish a full representation of U^* (by filling data from U) in matrix space.
13: Retrain K-Means model with data in U^* with minimized overhead.
14: Find the best VM type t_{best} by row-normalized weight matrix $T = FL^{-1}$.

Output:

The best VM type of a target workload x_n^*.

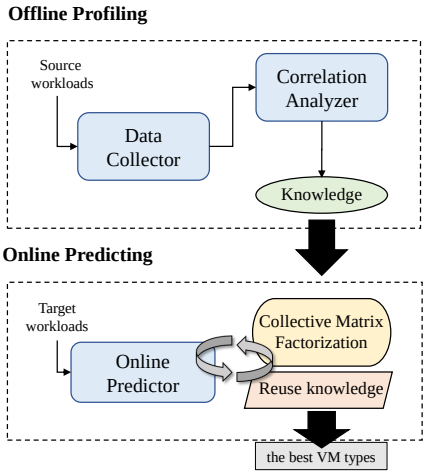


Figure 5: Architecture of Vesta.

Collect data from source workloads. We choose typical Hadoop and Hive workloads to run benchmark profiling on Amazon EC2 (details are in Section 5). Considering the performance variability in cloud environments, we run each workload 10 times to take a

conservative estimate of *P*90 values. The *Data Collector* collects low-level metrics in every 5 seconds using average resource utilizations. All data is stored in the MySQL database.

During each run, we also record correlation values for low-level metrics (e.g., 0.85 CPU-to-memory correlation) to support further analysis.

Train the offline K-Means model. The *Correlation Analyzer* is designed to initialize the relationship between labels and VM types. Specifically, we first measure the importance of correlations to reduce irrelevant information, since irrelevant information could deteriorate the accuracy of a system [6]. After that, we analyze the correlation similarities through an exhaustive search solution in [6] to further investigate the relationships. We use exhaustive search because it can bring out the optimal result with relatively high cost, which is acceptable for offline profiling. Finally, we train our offline model (K-Means) to group VM types into several categories.

4.2 Online predicting

In this phase, users should provide Vesta with target workloads and a set of VM types candidates. Then, Vesta predicts the best VM types for target workloads effectively by reusing classification knowledge from offline model.

In practical terms, Vesta first invokes the *Online Predictor*, which runs the target workload on 3 randomly picked VM types to initialize our *CMF* model. After that, Vesta can retrain the K-Means model with low overhead. In addition, the data of the target workloads may extremely sparse first (as target workloads are just entered the system). Vesta would continually update the model in the matrix space through *SGD* algorithm until the result converges. In the worst cases, Vesta may train workloads from scratch, just as the existing efforts.

5 EVALUATION

5.1 Experiment setup

Applications and settings. We employ two big data benchmarks — BigDataBench and HiBench to provide typical use cases in Hadoop, Hive and Spark frameworks. Table 3 shows the 30 big data applications, which provides a wide range of workloads. These workloads are not exhaustive but are intended to span the space of workload requirements to monitor a multiple-framework scenario.

We use default values to set application parameters except for *executor*[4] and memory parameters in Spark. In order to prevent *out of memory* (OOM) exceptions, we use Mesos [13] to watch the real usage of memory per executor. Then, we set the number of executors and the amount of executor memories based on the memory usage statistics.

As for the setting of input datasets, we follow the configurations in benchmarks. For instance, in HiBench, the dataset "gigantic" denotes 30 GB, "huge" denotes 3 GB and "large" denotes 300 MB, etc. In BigDataBench, we can set the input data size when required. We set the input data size based on the execution time so that all jobs run in a reasonable amount of time to reduce experiment costs.

[4]Executors are system processes in charge of running individual tasks in a given Spark application.

Table 3: Big data application workloads used in our experiments.

		No.	Name		No.	Name
Source Set	Training Set	1	*Hadoop-terasort*	Target Set	19	*Spark-spearman*
		2	*Hadoop-wordcount*		20	*Spark-svd++*
		3	*Hadoop-page-review*		21	*Spark-lr*
		4	*Hadoop-linear*		22	*Spark-page-rank*
		5	*Hadoop-lr*		23	Spark-kmeans
		6	Hadoop-twitter		24	Spark-bayes
		7	Hadoop-bayes		25	Spark-BFS
		8	Hadoop-index		26	Spark-CF
		9	Hadoop-identify		27	Spark-sort
		10	Hive-select		28	Spark-pca
		11	Hive-join		29	Spark-grep
		12	*Hive-scan*		30	Spark-count
		13	*Hive-full-join*			
	Testing Set	14	*Hadoop-nutch*			
		15	*Hadoop-pca*			
		16	*Hadoop-als*			
		17	Hadoop-kmeans			
		18	Hive-aggregation			

Note: Workloads with *italic* fonts are from HiBench, and workloads with normal fonts are from BigDataBench.

VM types. Considering there are more than one hundred VM types in today's public clouds, such as Amazon, Google and Microsoft, we choose 120 enterprise-level VM types of x86 architecture from Amazon EC2 [5]. Note that, in Amazon EC2, there are *VM Category* and *VM Family* on top of *VM type* to identify the resource characteristics. For each VM family, we choose reasonable VM types to reduce costs based on statistics in [7]. Table 4 shows the 120 VM types.

Table 4: VM types used in our experiments.

Category	VM Family	VM type
General Purpose	T3	small,medium,large,xlarge,2xlarge
	T3a	small,medium,large,xlarge,2xlarge
	M5	large,xlarge,2xlarge,4xlarge,8xlarge
	M5a	large,xlarge,2xlarge,4xlarge,8xlarge
	M5n	large,xlarge,2xlarge,4xlarge,8xlarge
Compute Optimized	C4	large,xlarge,2xlarge,4xlarge,8xlarge
	C5	large,xlarge,2xlarge,4xlarge,8xlarge
	C5n	large,xlarge,2xlarge,4xlarge,8xlarge
	C5d	large,xlarge,2xlarge,4xlarge,8xlarge
	C4n	small,medium,large,xlarge,2xlarge
Memory Optimized	R4	large,xlarge,2xlarge,4xlarge,8xlarge
	R5	large,xlarge,2xlarge,4xlarge,8xlarge
	R5a	large,xlarge,2xlarge,4xlarge,8xlarge
	R5n	large,xlarge,2xlarge,4xlarge,8xlarge
	X1	large,xlarge,2xlarge,4xlarge,8xlarge
	z1d	large,xlarge,2xlarge,4xlarge,8xlarge
Accelerated Computing	G3	large,xlarge,2xlarge,4xlarge,8xlarge
	G4	large,2xlarge,4xlarge,8xlarge,16xlarge
Storage Optimized	I3	large,xlarge,2xlarge,4xlarge,8xlarge
	I3en	large,xlarge,2xlarge,4xlarge,8xlarge

Alternative solutions. We compare our system against machine learning solution — PARIS [28] and performance modeling solution — Ernest [25]. PARIS uses a *Random Forest* model to predict the best VM types for data-intensive workloads. It assumes that a new-coming workload can be located to a category in *Random Forest* perfectly if it is from the same framework. Ernest leverages a performance-cost model to predict the performance of Spark based large-scale advanced analytics jobs through extensive performance measurement, but it only works well in Spark applications.

Table 5 shows the details of alternative solutions in our experiments.

[5]More details are in the website https://aws.amazon.com/ec2/instance-types/.

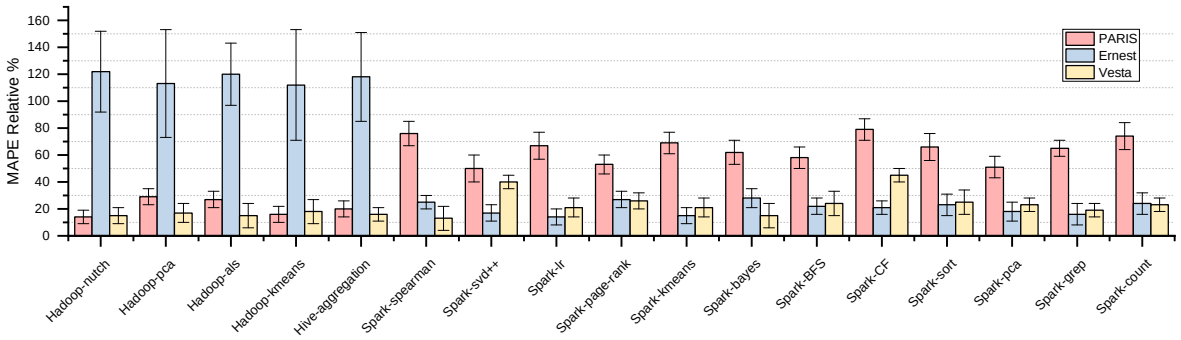


Figure 6: Comparing the prediction error against alternatives on multiple frameworks. The horizontal axis shows different workloads, and the vertical axis shows the MAPE. The bars show the standard deviations in the MAPE.

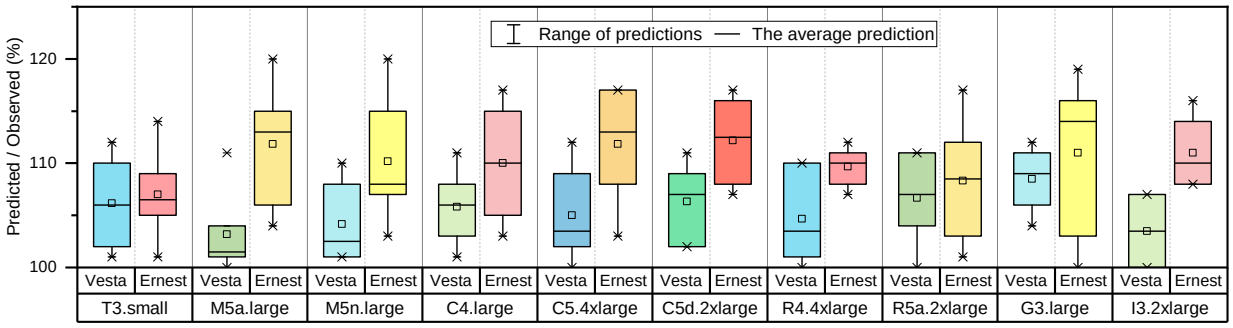


Figure 7: The results of predicting *Spark-lr* workload's execution time in 10 VM types. The horizontal axis shows 10 different VM types, and the vertical axis shows the prediction error. Boxes in different colors represent the prediction results of different VM types. The bar shows 10th and 90th percentile of deviations.

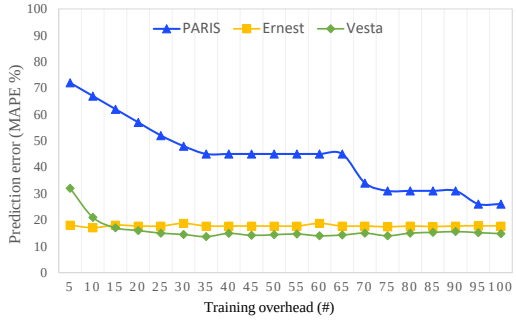


Figure 8: Training overhead comparing against PARIS and Ernest. According to our experimental setup (in Table 5), PARIS is training Spark workloads from scratch.

Datasets. To evaluate the effectiveness of optimizing workloads across multiple frameworks, we split 30 workloads of Hadoop, Hive and Spark into *source* and *target* sets. The source set is composed of Hadoop and Hive workloads, and the target set is formed of Spark workloads (see Table 3). Our goal is to compare the prediction of the best VM types for workloads in the target set.

Table 5: Alternative solutions in our experiments.

Solutions	Description
PARIS	In our empirical study in Figure 2, we argue that the way of reusing model is very fragile in practice — that is, the model is trained on Hadoop and Hive workloads, and we will test it on Spark.
Ernest	This performance model only works well on Spark workloads. We try to improve the model by assigning new weights for Hadoop and Hive, but the improvements for Hadoop and Hive are still limited.

In the source set, we use 13 workloads (mixed of Hadoop and Hive) as the training set to train our offline model, and use other 5 workloads in the testing set to test it. We use both Hadoop and Hive to build the source set because Hive workloads can cover SQL-like processing that Hadoop does not support.

In the target set, all workloads are from a different framework (Spark).

5.2 Experiment Metrics

We use the following metrics to evaluate Vesta:

- **Performance improvement.** In order to measure the improvement of application performance, we investigate how well the model can predict for a given application workload. In particular, we first observe ground truth "best" results by exhaustively running workloads on 120 VM types. Then, we

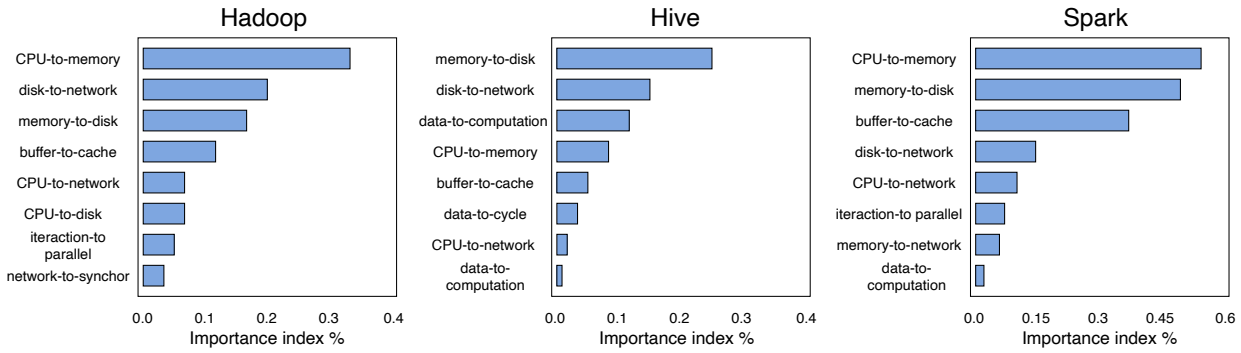


Figure 9: Importance of the correlations for workloads in Hadoop, Hive and Spark, respectively.

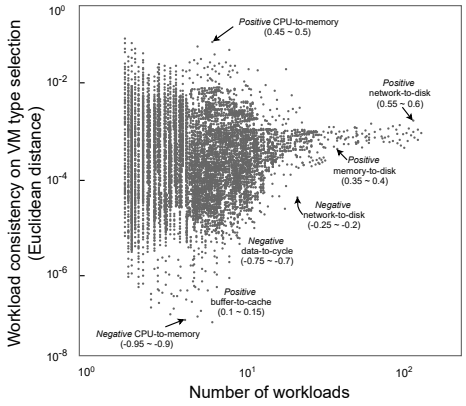


Figure 10: Evaluating correlations on different workloads and VM types. The horizontal axis shows the number of workloads with similar correlation values, and the vertical axis shows the VM type consistency on different workloads (evaluated by *Euclidean* distance). We can summarize that: there are popular correlations among considerable amount of workloads to facilitate good results in K-Means.

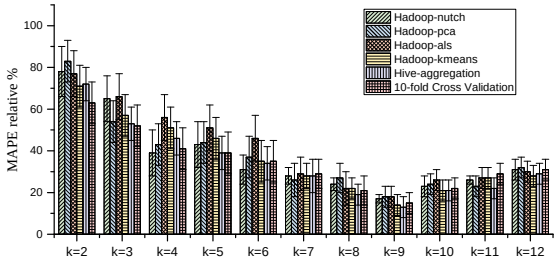


Figure 11: Evaluating the parameter k in K-Means, we tune it by running 10-fold cross validation. The horizontal axis shows different k values, and the vertical axis shows the MAPE. Boxes in different colors represent different workloads from the testing set in Table 3, and the bars show the 10th and 90th percentile of deviations in MAPE.

use MAPE (Mean Absolute Percentage Error) [8] to indicate the prediction error, and calculate the MAPE percentile between the predicted result to the ground truth "best" result.

$$MAPE = \frac{100\%}{m} \sum_{i=0}^{m} \left| \frac{predicted - ground\ truth}{ground\ truth} \right| \quad (7)$$

where m is the number of runs, and the MAPE range is $[0, +\infty)$. $MAPE = 0$ denotes a perfect model, while $MAPE > 100$ indicates a very bad model.

- **Training overhead.** In online phase, when workloads from a new framework arrive, Vesta needs to run these workloads on a few VMs to collect training data. We evaluate this training overhead comparing against alternatives.
- **Practical metrics.** To evaluate the best VM types on clouds for various workloads, we use two widely used performance targets — *execution time* and *budget* to measure the effectiveness of comparison systems. The execution time represents the time spent executing a workload on a specific VM type, and the budget denotes the cost of running the workload on the VM type.

5.3 Effectiveness of Vesta

We first evaluate the performance and overhead improvements for workloads from a new framework with several experiments. Following with evaluations of main components in Vesta such as *PCA* and K-Means. At last, we prove that Vesta can practical implicate to predict best VM types with shorter execution time and lower budget.

The performance and overhead improvements for workloads from a new framework. We use workloads in source set to train the model, and use workloads in target set to test the model.

Figure 6 shows the prediction error comparison against alternative solutions. The result shows that the overall prediction error of Vesta has been reduced by up to 51% comparing against machine learning approach PARIS. In other words, applications can achieve 51% performance improvement in Vesta for a new framework. When comparing against Ernest, for Hadoop and Hive workloads, Vesta can reduce 4× prediction error since Ernest can only works well on Spark workloads (we have explained the reason in Table 5). But there are two exceptions (*Spark-svd++* and *Spark-CF*) whose errors are much higher than others. In the case of *Spark-svd++*, we find

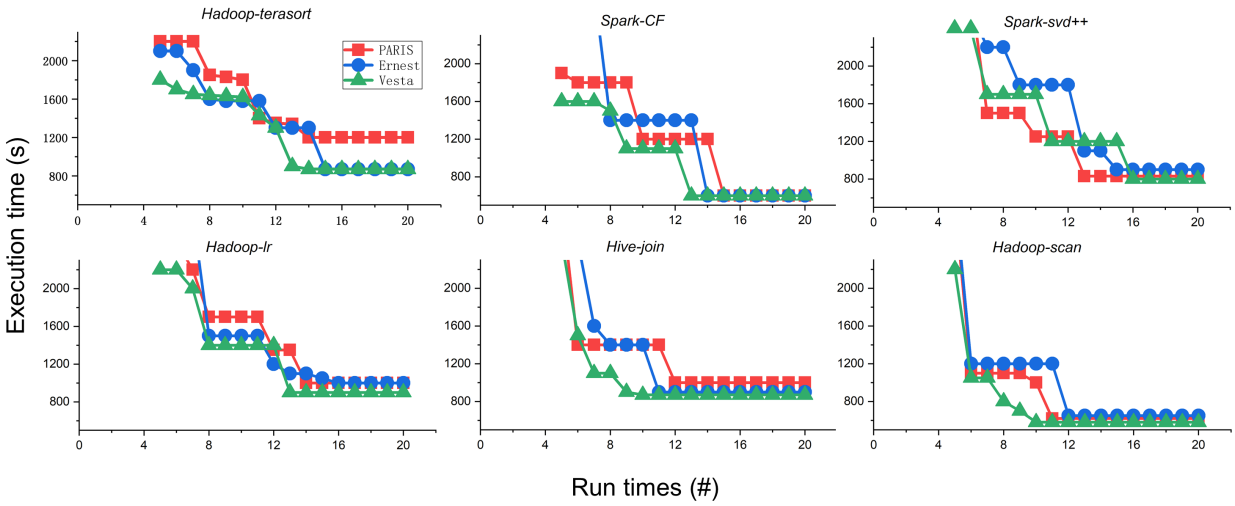


Figure 12: Comparing the effects of execution time optimization against alternatives. The horizontal axis shows the number of runs, and the vertical axis shows the predicted execution time in seconds.

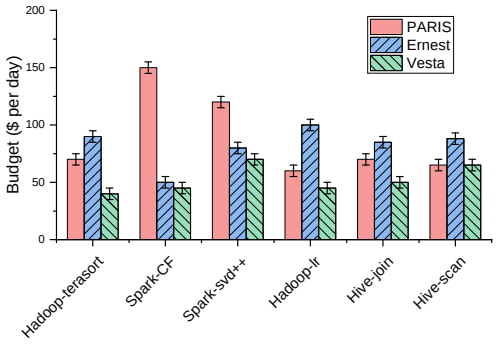


Figure 13: Comparing the effects of budget optimization against alternatives, the lower the better. The horizontal axis shows multi-framework applications, and the vertical axis shows the predicted budget. The bars show the 10th and 90th percentile of deviations in budget.

that it runs with high variance (close to 40%) every time and the prediction error in Vesta is within the variance. In the case of *Spark-CF*, the result does not converge in the *SGD* algorithm, which means it can hardly match with current knowledge in the offline model. In such case, we set a converge limitation in the online predicting phase to control when to stop the online process.

To further investigate the ability of improving applications' performance compared with performance modeling approach Ernest, we choose 10 typical VM types in Table 4, and predict the execution time of a compute-intensive workload *Spark-lr*. Figure 7 shows the predicting result, and we use $(Predicted/Observed) \times 100\%$ to evaluate the deviation between the predicted result and the observed result. In all of the cases, Vesta performs a better or at least a comparable result against Ernest since Vesta trains with large data sets offline.

Next, we evaluate the training overhead comparing against alternatives. In general, Vesta benefits from *CMF* model that can reuse knowledge from source workloads to reduce data collection efforts. In Equation 6, there is a parameter λ which can control the decomposition error in matrices. We set $\lambda = 0.75$ according to our best practice. Other techniques we used such as *PCA* and K-Means are discussed in following experiments. After that, we measure the effectiveness of reducing training overhead. Figure 8 compares the number of reference VMs for predicting Spark workloads against PARIS (train its model from scratch) and Ernest (low training overhead due to accurate modeling). The result shows that Vesta can reduce up to 85% training overhead comparing against machine learning approach PARIS (15 to 100), and the overhead is close to performance modeling approach Ernest. In total, Vesta incurs in substantially lower training overhead, due to various techniques, namely reusing the best VM selection knowledge via a combination of techniques.

The availability of main components in Vesta. In this experiment, we will evaluate several techniques used in Vesta.

First, we evaluate the importance of features via PCA and use *importance* index to evaluate them. As shown in Figure 9, the result shows that the *importance* indexes of three workloads of Hadoop, Hive and Spark, respectively. We use these results to reduce irrelevant information, and can reduce 49% useless data effectively.

Second, we leverage an exhaustive evaluation to study the correlations to further investigate correlation distribution among workloads. As we described in Section 3.1, we divide correlation values into 0.05 intervals to evaluate relevant correlations on different workloads and VM types. Figure 10 shows the result of correlation analysis. The horizontal axis denotes the *popularity* of labels (evaluated by the number of workloads in a same correlation interval), and the vertical axis represents the *consistency* of different workloads that prefer the same VM type (calculated by *Euclidean* distance). From the result, we can observe that most of the data

(near 90%) are stick together in the center, and we believe that the correlation distribution can reveal some facts: popular labels are not always more valuable than others (e.g., positive network-to-disk correlation of 0.55 to 0.6 on the right area); many workloads are close to each other in terms of VM type consistency (e.g., data in the center). In summarize, the result can prove that there are popular correlations among considerable amount of workloads to facilitate good results in K-Means.

Finally, we leverage a K-Means algorithm to train Vesta's offline model, we tune the hyperparameter k, and use *10-fold* cross validation to evaluate the result. Figure 11 shows that Vesta achieves lower prediction error when we set k to 9.

The practical implication of selecting the best VM type. Due to a combination of several technique, Vesta can find best VM types with shorter execution time effectively. Figure 12 shows the progression of finding shorter execution time for application workloads separately. Vesta is fastest for 5 of the 6 workloads except for *Spark-svd++*, whereby PARIS by chance finds better configurations during initial runs.

Vesta can also find VM types with lower budget. Figure 13 shows the comparison of the progress of finding lower budget for each application workload. Vesta performs better or comparable results comparing against alternatives. PARIS performs poorly on Spark since it is trained on Hadoop and Hive. Ernest does not work well on Hadoop and Hive because it is specifically designed for Spark.

6 RELATED WORK

Performance modeling. Ernest [25] makes a statistical profile of advanced analytics frameworks (Spark-like), and decomposes Spark jobs into different communication patterns. Similarly, MRTuner [21] builds a comprehensive model to represent the inter-task pipelines of a MapReduce job. These works cannot flexibly adapt to other frameworks without redesigning their models.

Black-box searching. In recent years, black-box search solutions have been designed to address this issue. CherryPick [1] introduces a *Bayesian Optimization* method to search the optimal cloud configurations for recurring jobs. However, it is designed to predict performance in a small set of VM types, and may suffer a low prediction accuracy if the search space is too large. Arrow [14] tries to solve the limitations on *CherryPick* using low-level performance metrics to augment the process of BO to reduce search cost and improve search accuracy. However, as shown in Figure 2, low-level metrics may suffer from high prediction error across frameworks.

Machine learning. PARIS [28] uses a *Random Forest* model to train a VM predictor for workloads across multiple cloud providers. However, we have illustrated that the way of reusing pre-trained model is fragile when workloads are from different frameworks. Vanir [4] combines a series of techniques such as Mondrian forest model and transfer learning to search the right cloud configurations. But their have not yet studied the correlation similarities or other types of similarities across frameworks.

Tuning application configurations. Recent projects aim at tuning application configurations in clouds. Some of them [11, 17, 19] monitor a specific framework and adjust configurations to improve the application performance. Others search for the best configurations using random [19] or local [11] strategy. Compared to application configurations, VM types have a smaller search space but a higher cost of trying out a VM type (both the expense and the time). Thus we find that reusing knowledge is critical for reducing costs.

7 CONCLUSION

In this paper we present Vesta, a transfer learning based system that predicts the best (or near best) VM types for big data applications across frameworks. Vesta abstracts knowledge offline and reuses knowledge online to predict the best VM types effectively.

Our method can cover a wide range of existing big data frameworks since they follow a basic architecture design of *Bulk Synchronous Parallelism*. What we need to do is to choose appropriate metrics according to workload characteristics and train new predictive function on them. For example, latency and throughput are important variables for measuring the performance of latency-sensitive workloads.

ACKNOWLEDGMENTS

We thank zhangbing ZHOU for careful and constructive comments. This work was supported by National Key Research and Development Program of China (2018YFB1402803).

REFERENCES

[1] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics.. In *NSDI*, Vol. 2. 4–2.
[2] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. 2018. Byzantine stochastic gradient descent. *Advances in Neural Information Processing Systems* 31 (2018), 4613–4623.
[3] Mohamed A. Attia and Ravi Tandon. 2019. Near Optimal Coded Data Shuffling for Distributed Learning. *IEEE Transactions on Information Theory* 65, 11 (2019).
[4] Muhammad Bilal, Marco Canini, and Rodrigo Rodrigues. 2020. Finding the right cloud configuration for analytics clusters. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 208–222.
[5] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Alexandre V Evfimievski, and Prithviraj Sen. 2018. On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML. *Proceedings of the Vldb Endowment* (2018).
[6] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. 2018. Feature selection in machine learning: A new perspective. *Neurocomputing* 300 (2018), 70–79.
[7] Eli Cortez and etc. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *SOSP*. ACM.
[8] Arnaud De Myttenaere, Boris Golden, and Le Grand. 2016. Mean absolute percentage error for regression models. *Neurocomputing* 192 (2016), 38–48.
[9] John Kevin Doyle, Thomas W Tucker, and Mark E Watkins. 2018. Graphical frobenius representations. *Journal of Algebraic Combinatorics* 48, 3 (2018).
[10] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. 2018. Compressed linear algebra for large-scale machine learning. *Vldb Journal* 27, 5 (2018), 719–744.
[11] Anastasios Gounaris and Jordi Torres. 2018. A methodology for spark parameter tuning. *Big data research* 11 (2018), 22–32.
[12] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.
[13] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A platform for fine-grained resource sharing in the data center.. In *NSDI*, Vol. 11. 22–22.
[14] Chin-Jung Hsu, Vivek Nair, and Freeh. 2018. Arrow: Low-level augmented bayesian optimization for finding the best cloud vm. In *ICDCS*. IEEE, 660–670.
[15] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *ICDEW 2010*. IEEE, 41–51.
[16] Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri, and Engelbert Mephu Nguifo. 2018. An experimental survey on big data frameworks. *Future Generation Computer Systems* 86 (2018), 546–564.
[17] Yan Li, Bo An, Junming Ma, Donggang Cao, Yasha Wang, and Hong Mei. 2020. SpotTune: Leveraging Transient Resources for Cost-efficient Hyper-parameter Tuning in the Public Cloud. *arXiv preprint arXiv:2012.03576* (2020).

[18] Yan Li, Junming Ma, and Donggang Cao. 2020. Cross-Domain Workloads Performance Prediction via Runtime Metrics Transferring. In *2020 IEEE International Conference on Joint Cloud Computing*. IEEE, 38–42.

[19] Harshitha Menon and etc. 2020. Auto-tuning Parameter Choices in HPC Applications using Bayesian Optimization. In *IPDPS*. IEEE, 831–840.

[20] Xiaobo Shen, Weiwei Liu, Ivor Tsang, Fumin Shen, and Quan-Sen Sun. 2017. Compressed k-means for large-scale clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

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

[22] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. 2010. The hadoop distributed file system.. In *MSST*, Vol. 10. 1–10.

[23] Ajit P Singh and Geoffrey J Gordon. 2008. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 650–658.

[24] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, and etc. 2009. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment* 2, 2 (2009), 1626–1629.

[25] Shivaram Venkataraman, Zongheng Yang, Michael J Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics.. In *NSDI*. 363–378.

[26] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, et al. 2014. Bigdatabench: A big data benchmark suite from internet services. In *HPCA*. IEEE, 488–499.

[27] Jason Xu and Kenneth Lange. 2019. Power k-means clustering. In *International Conference on Machine Learning*. 6921–6931.

[28] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. 2017. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *SoCC*. ACM, 452–465.

[29] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 338–350.

[30] Hua Zuo, Jie Lu, Guangquan Zhang, and Feng Liu. 2019. Fuzzy Transfer Learning Using an Infinite Gaussian Mixture Model and Active Learning. *IEEE Transactions on Fuzzy Systems* PP, 2 (2019), 1–1.