This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

d-Simplexed: Adaptive Delaunay Triangulation for Performance Modeling and Prediction on Big Data Analytics

Yuxing Chen, Peter Goetsch, Mohammad A. Hoque, Jiaheng Lu, Sasu Tarkoma
Department of Computer Science, University of Helsinki
{yuxing.chen, peter.goetsch, mohammad.a.hoque, jiaheng.lu, sasu.tarkoma}@helsinki.fi

Abstract—Big Data processing systems (e.g., Spark) have a number of resource configuration parameters, such as memory size, CPU allocation, and the number of running nodes. Regular users and even expert administrators struggle to understand the mutual relation between different parameter configurations and the overall performance of the system. In this paper, we address this challenge by proposing a performance prediction framework, called d-Simplexed, to build performance models with varied configurable parameters on Spark. We take inspiration from the field of Computational Geometry to construct a d-dimensional mesh using Delaunay Triangulation over a selected set of features. From this mesh, we predict execution time for various feature configurations. To minimize the time and resources in building a bootstrap model with a large number of configuration values, we propose an adaptive sampling technique to allow us to collect as few training points as required. Our evaluation on a cluster of computers using WordCount, PageRank, Kmeans, and Join workloads in HiBench benchmarking suites shows that we can achieve less than 5% error rate for estimation accuracy by sampling less than 1% of data.

Index Terms—Performance modeling, Big data analytics, Adaptive sampling, Delaunay Triangulation.

1 Introduction

Numerous Big Data frameworks have been introduced to address the problem of organizing large-scale fault-tolerant computation in a clustered environment (the cloud). The general-purpose frameworks which have emerged, such as Spark, [40], Hadoop [38], and MapReduce [9], are capable of handling diverse Big Data analytics workloads. In a cloud environment where resources are billed down to the second, the costs of job inefficiencies rapidly become visible, and it is becoming increasingly important for Big Data frameworks to run jobs efficiently.

These modern Big Data frameworks are complex systems. They have many resource tuning knobs, such as memory, CPU/GPU allocation, the number of running nodes, and other I/O considerations. With so many tuning knobs, the end users (often non-technical people) submitting the jobs are left clueless about the impact of each parameter on the performance of the job and the overall performance of the cluster. Furthermore, the choice of the configuration parameters is highly dependant on the type of job, the amount of resources available, and the input data size, etc.

Motivation and Challenges. Our work models the performance topography of whole feature space. Figure 1 depicts the execution time of a Spark job (Kmeans clustering) with three varied parameters, including input data size, the number of vcores (virtual CPU cores), and the amount of memory. In contrast, tuning only finds one parameter value towards the (local) optimal performance of the specific metric. Tuning to another goal often requires retraining. Best parameters for runtime is not necessarily cost-effective [35]. Unlike tuning, performance modeling can determine the best resource provisioning (e.g., [7], [32]) towards different goals, e.g., shortest runtime [30], lowest resource consumption [15], highest throughput [1], etc. It is meaningful to build a topological model with a fewer but important parameters to represent the performance surface for further decisions, such as the best monetary cost (determining optimal number of machines [35]) in `Amazon` "pay-as-you-go" services[1], fewer resource assignment but deadline-guarantee in a competed cluster [7].

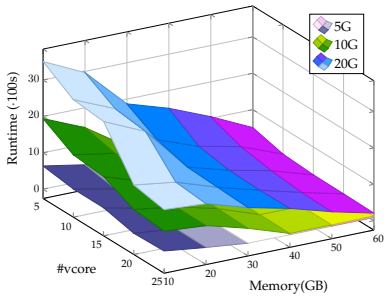


Fig. 1: 3D topography of the response surfaces for a Spark job (Kmeans workload) with varied virtual CPU cores (vcore) and memory with 5G, 10G, and 20G input data size.

However, we are faced with challenges in building an accurate prediction model and achieving enough samples for training the model. There exist many excellent works for performance modeling and tuning in different plat-

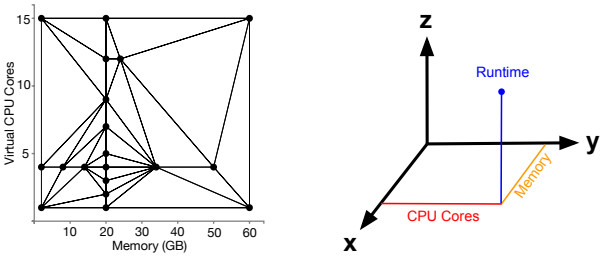
Manuscript received February 2019.

1. https://aws.amazon.com/ec2/pricing/on-demand/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

TABLE 1: An overview of the comparison of state-of-the-art approaches. In usage, modeling builds the performance topography of whole configuration space. However, tuning finds only one parameter value towards the (local) optimal performance of the specific metric.

Approach	Model	Parameter	Usage	Comments
Ernest [35]	Cost function	Vcore	Modeling	Works for only one parameter
Alvaro [16]	Regression models	Parallelism	Tuning	Hard to pick model, (local) optimal finding
OtterTune [1]	Guassian process	Sets of parameters	Tuning	Possible overfitting, (local) optimal finding
CDBTune [42]	Neural networks	Sets of parameters	Tuning	Possible overfitting, (local) optimal finding
d-Simplexed	Delaunay triangulation	Sets of parameters	Modeling	Apply for multiple parameters and adaptive sampling



(a) (b)

Fig. 2: (a) An example Delaunay Triangulation constructed from two-dimensional samples for the features memory and vcore, and (b) runtime prediction using the feature configurations for a Spark workload.

forms, such as Hadoop (e.g., [17], [30]), Spark(e.g., [16], [35]), Database (e.g., [1], [11]). Table 1 compares the most relevant state-of-the-art approaches. Alvaro [16], OtterTune [1], and CDBTune [42] use several regressors to tune a set of parameters. However, they train the models in the way of maximizing one objective, i.e., predicting local optimal performance. Therefore do not perform well on the whole topology of the feature space. In contrast, Ernest [35] is designed to predict any unknown parameters in the topology but capable of predicting only vcore parameter.

Yet, these regression models either (Gaussian Process [1] and multi-layer Neural Network [42]) face a deterioration accuracy due to overfitting and require many samples for model building or (Linear regression [16], Ernest [35]) are simple but achieves unsatisfactory accuracy. We contribute to apply a novel accurate model and develop an adaptive sampling to mitigate the training cost, as follows.

Contribution 1 in § 4 – Delaunay Triangulation model. We introduce a novel black-box method for modeling and predicting the performance of Big Data applications. We build a bootstrap performance surface model, which accurately predicts any unknown parameters. We take inspiration from the field of Computational Geometry to construct a $d+1$-dimensional mesh (e.g., [13]) over a selected set of d features with Delaunay Triangulation (DT) [10], which has wide applications in the fields of computational geometry (e.g. curved surface modeling [6]) and computer graphics (e.g. path planning in automated driving [2].

In particular, DT partitions the d feature space into a set of interconnected *d-simplexes*[2]. Such a piece-wise model helps to avoid overfitting. Figure 2a illustrates an example mesh of 2-simplexes constructed from memory and CPU core features. We first determine a simplex to which an unknown feature configuration belongs to. Then we predict the job execution time by calculating a hyperplane in $d+1$ dimensional space by bringing in the runtime dimension as shown in Figure 2b. In Figure 1, DT is required to construct a mesh of 3-simplexes, or tetrahedrons for three features (i.e., vcore, memory, and data size) and predicts the runtime for a Kmeans job.

Contribution 2 in § 5 – Adaptive sampling. The premise for the above performance prediction is building the mesh model. However, training the whole topography is extravagant with a large parameter space, and randomly picking samples does not guarantee the desired accuracy. Determining both the right fraction and the appropriate representatives of the samples for building a model is not trivial. To address this challenge, we integrate an adaptive sampling framework with d-Simplexed. The sampling method bootstraps with Latin Hypercube Sampling (LHS) (e.g., [25]), which spreads widely in each feature dimension, and uses fewer points to represent the whole range of feature values. Based on experiments, we develop an approach to estimate the *utility* of new candidate samples and select those samples which significantly improve the model.

Contribution 3 in § 6 – Comprehensive experiments. The Delaunay Triangulation (DT) model and the adaptive sampling algorithms (We name it d-Simplexed) are implemented and comprehensively evaluated on the Spark platform through benchmarking and synthetic workloads. The empirical experiments show that d-Simplexed outperforms the state-of-the-art methods, such as Decision Tree Regression (DTR) by Alvaro [28], Cost-based method (e.g., Ernest [35]), Gaussian Process (GP) in OtterTune [29], and Multilayer Neural Network (NN) in CBDTune [42]. Also, the proposed adaptive sampling method enables us to use fewer samples to train the model and outperforms the baseline sampling techniques, i.e., random and gridding sampling. d-Simplexed exploits a few samples to achieve a very low error prediction, and the error continues decreasing as the samples increase. For example, we achieve 1.58% prediction error by sampling 1% data points for the Kmeans workload.

The rest of the paper is structured as follows. In § 2, we provide a background on Spark and some mathematical primitives required to understand our method. In § 3, we present the problem formally. § 4 and § 5 present our contributions in modeling, prediction, and sampling for d-Simplxed. In § 6, we analyze the empirical performance results. We survey the related work in § 7, and conclude

2. A Simplex is borrowed from computational geometry, denoting a triangle-like object that can be extended to arbitrary dimensions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

TABLE 2: Key Spark Executor Configuration Parameters

Executor Parameter	Configuration Parameter
Count	`executor.instances`
Executor Memory	`executor.memory`
Executor Threads	`executor.cores`
Our parameter	**Configuration Parameter**
Memory	`executor.instances` × `executor.memory`
Vcore	`executor.instances` × `executor.cores`

this paper in § 8.

2 Preliminaries

In this section, we lay the groundwork required to understand our method presented hereafter, including Spark preliminaries and computational geometry primitives.

2.1 Spark Preliminaries

Spark [40] is a general-purpose cluster computing framework. It was originally built to handle iterative Big Data algorithms (e.g., Machine Learning workloads). It uses a distributed memory abstraction called Resilient Distributed Datasets (RDDs) [41] that enables parallel computation on data and make such data exceptionally resilient to faults.

At runtime, Spark applications are split into tasks and assigned running executors. The executors remain alive for the duration of the job using multiple CPU threads running received tasks in parallel. The size and number of executors, as well as their available execution threads, are the key performance indicators for any given job. A summary of the relevant tuning parameters controlling them is presented in Table 2. Spark runs on top of a cluster resource manager such as Mesos [18] or YARN [34]. When submitting jobs to the resource manager, users may specify the configuration parameters from Table 2, and the resource manager will manage the allocation.

2.2 Delaunay Triangulation Primitives

In the following, we introduce some basic terms and concepts on Delaunay Triangulation (DT) from Computational Geometry.

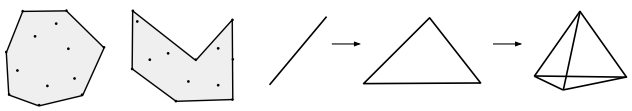
Convex Region. A convex region [26] is a region such that, for every pair of points in the region, every point on the straight line segment that joins the pair of points is also within the region. We show an example in Figure 3a, illustrating the convex and non-convex regions.

Convex Set. A convex set [26] represents the points inside a convex region. In convex geometry, a convex set is a subset of an affine space that is closed under convex combinations [5].

Convex Hull. It is the fundamental construction of Computational Geometry. The convex hull [3] of a set of points is the smallest convex set that contains the points.

Hyperplane. It is the generalization of a plane in three dimensions to higher dimensions, i.e., any d subspace in $\mathbb{R}^{d+1}$.

Simplex. It is the generalization of a triangle to different dimensions. In d dimensions, the concept of a triangle becomes a d-simplex. For example in Figure 3b, a 2d triangle is a 2-simplex, a 3d tetrahedron is a 3-simplex, and so on.



(a) (b)

Fig. 3: (a) Convex hull bounding a convex region (left) and a non-convex hull bounding a non-convex region (right). (b) A 1-simplex (line), 2-simplex (triangle) and 3-simplex (tetrahedron). Delaunay Triangulation in $\mathbb{R}^d$ construct d-simplexes.

Furthermore, each simplex is constructed of facets, which form the boundary (i.e., min and max values) of the surface. The number of facets in a simplex is a function of the number of edges.

In our context, we construct the model by plotting a d feature configuration of Spark into a d-dimensional space, using DT to partition the feature space into a set of interconnected d-simplexes, thereby forming a $d+1$-hyperplane over the feature set. Unlike previous approaches (e.g., OtterTune [1]), we build the DT performance model of the entire feature space instead of searching local optima (Section 4).

3 Problem Statement

At the high level, we want to predict the runtime of a Big Data analytics job with varied parameter configurations given a set of historical (or sampled) data points. Intuitively, the research problem of this paper is: *how to develop a performance model based on the sampled data to produce accurate runtime predictions with given parameter configurations?* More formally, given a collection of n samples, a configuration space F, a prediction model $PM(\cdot)$ returns the estimated running time $\hat{T}$ for F:

$$\hat{T} = PM(\mathcal{S}, F) \tag{1}$$

where $\mathcal{S} = \{\langle F_i, T_i\rangle | 1 \leq i \leq n\}$, and each configuration $F_i = \{f_1, f_2, ..., f_d\}$ includes d features[3] (parameters) and the corresponding runtime is T_i. To evaluate the model in our experiments, we use a metric called the Mean Absolute Percentage Error (MAPE), which is the average of the Percent Errors for a set of estimated runtimes ($\hat{T}$) and actual runtimes (T):

$$MAPE = \frac{100\%}{l} \sum_{i=1}^{l} |\frac{T_i - \hat{T}_i}{T_i}| \tag{2}$$

where l is the number of specific feature configurations to be tested.

In general, this problem can be converted to a general prediction problem with historical data (i.e. training data), which can be solved by a statistics machine learning method such as Gaussian Process (GP) [29] and Neural Network [42]. However, as we will show in our evaluation in § 6, these methods require massive training data points for better prediction, and they are very costly to construct an accurate performance model in our problem. In contrast, we will develop a novel method for accurate runtime prediction by building an effective estimation model which requires much less training data.

3. Features can be not only system parameters (e.g., vcore and memory) but also job or input data information (e.g., data size).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

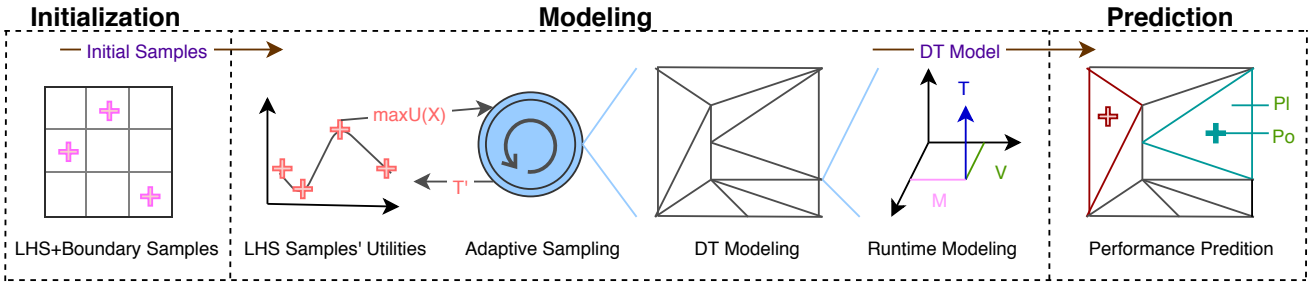


Fig. 4: d-Simplexed architecture. LHS, Boundary Samples and Adaptive Sampling in § 5. DT Modeling and Runtime Modeling in § 4.1. Performance Prediction in § 4.2. Abbreviation: Utility ($U(X)$) of samples X, Running time (T), Predicted time (T'), Memory (M), vcore (V), Point to be predicted ($Po = \langle f_1, f_2 \rangle$), Plane to predict ($Pl = \beta_1 x_1 + \beta_2 x_2 + \beta_3$).

Against our problem, we propose a framework, called d-Simplexed, by using the Delaunay Triangulation (DT) model to make the prediction for a given parameter configuration and heuristic adaptive sampling to reducing samples for training. Figure 4 shows our framework at a high level. Correspondingly, a brief algorithm is also described in algorithm 1. The main features are adaptive sampling, DT modeling, and performance prediction. We present the DT model and its prediction in the following section, then introduce adaptive sampling that accelerates the modeling in § 5.

4 Delaunay Triangulation

The central task of the paper is to model the performance (e.g., Kmeans performance mesh in Figure 1 and synthetic workload performance mesh in Figure 11) of a job with a set of parameters. There exist many methods for creating the polygon mesh. In particular, a Delaunay triangulation [10] is a triangulation DT(P) (where P is set of discrete points in a plane) such that no point in P is inside the circumcircle of any triangle in DT(P). Delaunay triangulation maximizes the minimum angle of all the angles of the triangles in the triangulation. The reasons for choosing the DT are threefold:

1) It prefers to form equilateral triangles as much as possible. Avoiding long and skinny triangles leads to a better interpolation (prediction runtime for our case) of values because the vertices of a long and skinny triangle tend to be spread out far from each other [8].
2) It can easily be extended into higher dimensions meaning that the model is not limited by the number of features in our context.
3) It is well researched – many algorithms and tools have been developed for creation and traversal of triangulation (e.g., Flip [8] and QuickHull [3]).

The division of a 2-dimensional or d-dimensional shape into a collection of discrete planes has wide applications in the fields of computational geometry and especially computer graphics (e.g. path planning in automated driving [2]). We introduce the following main steps to build and use the DT model to fit our problem of performance modeling and prediction:

1) **Triangulation:** Given a set of d features (parameters) $\{f_1, f_2, ..., f_d\}$, e.g., {16 GB, 4 vcores}, we build the Delaunay Triangulation model in $\mathbb{R}^d$ space;
2) **Projection:** From each d-simplex returned from a Delaunay Triangulation model, we use the running-times of each of the $(d+1)$ points to compute the hyperplanes;
3) **Prediction:** Given a new parameter configuration, we can make the running-time prediction based on the model constructed before.

Algorithm 1: d-SIMPLEXED: framework workflow

Input: Set of hitorical points $\mathcal{F}'$ with runtime data and new configurations $\mathcal{F}$

```
S ← boundary samples                          // initial in § 5
M ←initial prediction model from S
M ← Adaptive_Sampling(F' − S, M)   // refine the model by algorithm 3
foreach F ∈ F do
    T ← Prediction(M, F)                      // by algorithm 2
    T ← T ∪ T
```

Result: Prediction model M, Predicted runtimes $\mathcal{T}$

4.1 Modeling

Triangulation. With d features selected, we generate a DT mesh in $\mathbb{R}^d$ space. In two or three dimensions, DT forms familiar meshes over the feature configuration space, thereby connecting unknown points and enabling runtime prediction. Figure 2 shows an example of a Delaunay Triangulation over a 2-dimensional $\langle memory, vcore \rangle$ data set. In higher dimensions, the idea works in the same way except that it becomes more complex to visualize. Other approaches (e.g., cost-based models [31], [35]) attempt to fit a smooth function over a similar space. In contrast, we split the space into regions using DT, instead of trying to fit a single function to the whole data set. To efficiently to employ DT, we apply Quickhull [3] algorithm. There are many variations to the Quickhull algorithm, and the one we implemented is the most generic. Quickhull runs in $\Theta(n \log n)$ with a worst-case complexity of $O(n^2)$. The worst-case complexity occurs when points have unfavorable (highly symmetric) distributions. In our framework, we do not consider this to be a valid cause for concern, as our sampling method in § 5 ensures that we pick distributed asymmetric points, thereby avoiding the high-symmetry pitfalls.

Projection. Following that we use d features together with runtime to find a multivariate linear function representing a hyperplane in $\mathbb{R}^{d+1}$ which is responsible for the predictions of points contained in each region (d-simplex).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

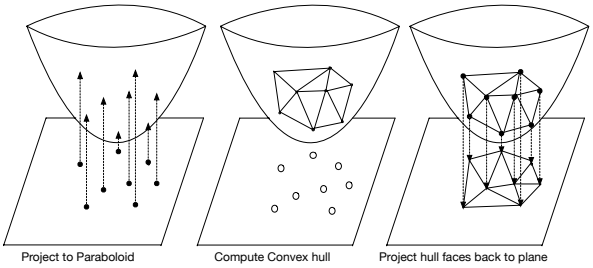


Fig. 5: Example of how do Delauney Triangulation and Convex Hull work by a parabola ($x^2 + y^2$). (i) Compute the convex hull of the point set in $\mathbb{R}^{2+1}$ space (by lifting the points to a parabola $(x, y, x^2 + y^2)$); (ii) project the convex hull back into $\mathbb{R}^2$ space, leaving the DT in $\mathbb{R}^2$.

There is no requirement for the fitting function to be continuous; defining it piece-wise makes it easier and more accurate. DT partitions the feature space into a series of simplexes and we construct a hyperplane by bringing in the runtime dimension for each simplex. This process is best explained by stepping through in two and three dimensions, then extrapolating into higher dimensions. In two dimensions, e.g. $\langle f_1, f_2 \rangle$, DT generates 2-simplexes (triangles), each naturally containing three points. For each point $\langle f_1, f_2 \rangle$, we add the runtime as the third dimension (e.g. $\langle f_1, f_2, runtime(f_1, f_2) \rangle$), then compute the hyperplane containing these three points. In three dimensions, e.g. $\langle f_1, f_2, f_3 \rangle$, DT creates 3-simplexes (tetrahedrons), each containing four points as shown in Figure 3b. Next, we add runtime in and then compute the hyperplane containing each of the tetrahedron's points. As we travel into higher dimensions, the process remains the same. In other words, in a model constructed with d features, we must lift it into $\mathbb{R}^{d+1}$ by including runtime in a separate step. We illustrate the modeling in 1.

Example 1. To generate a DT mesh for a set of points in $\mathbb{R}^d$ space, we utilize the relationship[4] between DT in $\mathbb{R}^d$ and a parabola in $\mathbb{R}^{d+1}$. Figure 5, by a parabola $(x, y, x^2 + y^2)$, depicts the simplex and hyperplane relation in 2-dimension (x, y). We first compute the hyperplane, i.e., the convex hull of the point set in $\mathbb{R}^{2+1}$ space (by lifting the points to a parabola), then project the convex hull back into $\mathbb{R}^2$ space (i.e. simplexes), leaving DT in $\mathbb{R}^2$.

4.2 Prediction

Once we have the new configuration $F = \langle f_1, f_2, ..., f_d \rangle$, we can predict runtime based upon the features in the model.

4. Delaunay triangulation for parabola projection is detailed in http://www.cs.wustl.edu/~pless/546/lectures/L13.html

Algorithm 2: PREDICTION

Input : Model M in $\mathbb{R}^d$ includes a set of d-simplexes and attached $\mathbb{R}^{d+1}$ hyperplanes; New configuration $F = \langle f_1, f_2, ..., f_d \rangle$

```
foreach s_i ∈ M do                                  // simplex s_i
    if F ∈ s_i then                                 // find the simplex
        h ← hyperplane(s_i)                         // Calculate parameter values of hyperplane
        T ← h(F)   // compute runtime by Equation 3
        return T
```

Result: Predicted runtime T for configuration F

A formal procedure can be found in algorithm 2. Given the computed DT model and new configuration F, the first step is to determine which simplex a given point belongs to. We then can calculate the hyperplane for prediction. The generic scalar form of a hyperplane of d dimensions is $\beta_0 = \sum_{i=1}^{d+1} \beta_i x_i$. By the known d+1 configurations of the simplex and their runtimes, we can compute the hyperplane's parameter $\{\beta_0, ..., \beta_{d+1}\}$. We define $h(\cdot)$ to return runtime (e.g., x_{d+1}) when given new configuration values in Equation 3. When this simplex's hyperplane is obtained, we simply can plug the known configuration values into Equation 3 to get the estimated runtime at that specific configuration F.

$$T = x_{d+1} = h(x_1, ..., x_d) = \frac{\beta_0 - \sum_{i=1}^{d} \beta_i x_i}{\beta_{d+1}} \tag{3}$$

Note that this prediction algorithm relies heavily on the assumption that any potential prediction values will lie inside the convex hull of the model. This is precisely why in the approach that we picked 2^d *boundary samples* at the extremes of the feature configuration space. This step ensures that any future predictions will fall inside the convex hull, and thus have a simplex for prediction. Points lying outside the convex hull ("outer-hull" points) are difficult to predict because their values must be extrapolated from a hyperplane inside the hull. By selecting boundary samples at the beginning, we can avoid this situation as described in the right following section.

As sample adds to DT model, DT will split the surface into multiple regions, and model the surface locally and linearly, avoiding overfitting with an upper error bound. That is when every hyperplane covers a monotonic surface, the prediction performance is with a bound as follows:

Lemma 1. If each hyperplane predicts a monotonic surface, the MAPE of each hyperplane prediction bounds to:

$$MAPE \leq \frac{T_{max} - T_{min}}{2 \times T_{min}} \tag{4}$$

where T_{max} and T_{min} are the maximum and minimum performance in the split surface(or hyperplane).

The proof idea is to calculate the maximum error between an arbitrary monotonic function and linear function built by T_{max} and T_{min}.

5 ADAPTIVE SAMPLING

As mentioned in the earlier section, DT may perform poorly when the feature space has an unfavorable or highly symmetrical distribution. Also, even finding an optimal parameter for the system is NP-hard [1,42]. To remedy this, we rely on heuristic sampling. The baseline sampling techniques in this situation are random sampling and grid sampling. Random sampling selects points at random from the configuration space without replacement. Grid sampling (e.g., in [33] sampling for database parameters) meanwhile divides the configuration space into uniform grids and selects points which lie at an equal distance from each other. Samples gathered from these baseline sampling techniques may produce a sub-optimal model. The reason is that they may oversample in the regions where there is little change in the runtime and under-sample in the regions where the runtime

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

is highly dynamic. Consequently, in this paper, we introduce a novel feedback-driven sampling technique to improve the model's performance by selecting more samples from the place where the topography of the model is turbulent, i.e. to avoid model overfitting by improving the errors addressed in Lemma 1. To guarantee each hyperplane predicting the monotonic surface, we heuristically discover sparse area with less known parameters, while to lower the error bound, we heuristically choose the area with huge performance changes (define in Equation 6).

In particular, we split our sampling technique into two phases. First, we select seed samples to initialize the model. Second, we iteratively add samples by a utility-driven method to improve the model in each step. The sample with the highest utility, i.e., the highest potential to improve the model's prediction accuracy, is picked to update the model.

Initial Samples. We go through the following steps to generate the seed samples:

1) Determine d features to include in the model and their bounds, thereby forming the *feature space* for the model, e.g, d features have 2^d boundary samples. We normalize the feature space $\in [0, 1]$.
2) Use Latin Hypercube Sampling (LHS) [20] to select m feature configurations. In the LHS technique, samples are chosen in a way such that the complete range of parameter values is fully represented. However, LHS may generate bad spreads where all samples are spread along the diagonal. Therefore, we maximize the minimum distance between any pair of samples. Suppose we have n samples, we will select the sample set X^* such that:

$$X^* = arg \max_{\{1<i<n\}} \min_{\{f_1^{(x_1^i)},..f_d^{(x_d^i)} \in \mathcal{D}_{LHS}\}} \textbf{Dist}(f_1^{(x_1^i)}, ..., f_d^{(x_d^i)}) \quad (5)$$

   where $x_1 \neq ... \neq x_d$, and **Dist** is a typical distance metric, e.g., Euclidean distance.
3) Combine LHS samples with 2^d boundary samples (taken from the minimum and maximum points of each feature axis) to gather $2^d + m$ seed samples for the initial model. These seed samples will be excluded in adaptive sampling. The reason for including boundary points is to avoid outer-hull point prediction, which normally performs unsatisfactorily.

Algorithm 3: ADAPTIVE_SAMPLING

```
Input: Sample set: S; initial prediction model: M
1  repeat
2      D_LHS ← LHS(S)                                  // Equation 5
3      F_(n+1) ← ∅                                     // next sample
4      U_X ← 0
5      foreach X* ∈ D_LHS do    // U(X*) from Equation 7
6          if U_X < U(X*) then
7              F_(n+1) ← X*
8              U_X ← U(X*)
9      M ← update(M, F_(n+1), T(F_(n+1)))   // update model
           with Quickhull [3]
10     S ← S − F_(n+1)
11 until stopping condition is met
12 return M
Result: Prediction model: M
```

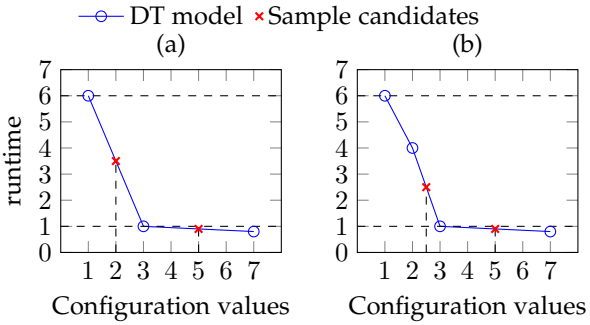


Fig. 6: Illustration of Example 2. The next sample point to pick is based on the current DT model. (a) Pick configuration value 2. (b) Pick configuration value 5.

Adaptive Sampling. The adaptive sampling technique is to select new points for improving the model accuracy. We heuristically search the area with the greatest runtime changes and more unknown configurations. We need more samples from these areas where small changes in feature values may result in significant changes in runtime. Such heuristic helps accelerating the convergence of the model in a turbulent surface, i.e. performance rapidly and non-monotonically changing. We introduce a *utility* metric to compute the distance between the predicted point and its hyperplane. Intuitively, a higher *utility* value indicates a larger distance to the points of its prediction hyperplane and thus a higher potential improvement to the model. Given samples X from LHS domain $\mathcal{D}_{LHS}$ and n samples $\mathcal{S}$ with d features, we achieve the predicted runtime $\hat{T^{(i)}}$ by sample $X^{(i)}$ and its hyperplane $\mathcal{S}' \subseteq \mathcal{S}$ with h sample $\langle F_k^{(i)}, T(F_k^{(i)})\rangle, 1 < k < h$. The utility $U(X^{(i)})$ of sample $X^{(i)}$ is defined as follows:

$$U(X^{(i)}) = \frac{1}{h}\sum_{k=1}^{h}\left(\frac{1}{d+1}\left(\sum_{j=1}^{d}(f_{k,j}^{(i)} - X_j^{(i)})^2 + (T(F_k^{(i)}) - \hat{T^{(i)}})^2\right)\right) \quad (6)$$

where h is the number of points constructing its hyperplane. The iterative sampling technique proceeds as follows until a stopping condition has been met.

Our adaptive sampling is described in algorithm 3. The algorithm adds sample in each iteration. Line 2 is to get new m sample points $\mathcal{D}_{LHS}$ using Latin Hypercube Sampling across the entire feature space in every iteration. Line 5-8 are to get the sample with the highest utility using the current model to compute the utility $U(\cdot)$ of X^* by Equation 6, where for each X^* in $\mathcal{D}_{LHS}$. Next, we rank all the samples by utility and pick the largest $U(X^{(i)})$ as the next sample to add to the model (by Equation 7). The features of this new sample $F_{(n+1)}$ is achieved as follows:

$$F_{(n+1)} = arg \max_{X \in \mathcal{D}_{LHS}} U(X) \quad (7)$$

Line 9 updates the model and Line 10 removes the added sample from sample set. We define an explicit threshold of prediction error as the stopping condition. In our empirical experiments, we continue selecting more sampling points into the model until an average error has been reached (e.g., MAPE$\leq$ 5%). We provide an example to illustrate the idea of picking the next point by the utility as follows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

Example 2. Figure 6 depicts an example scenario with synthetic data. Suppose three experiments have been done, and the collected data points are shown in subfigure (a). Suppose the runtime is 3.5 for configuration value 2 from points $(1, 6)$ and $(3, 1)$ by linear regression. Based on the points $(1, 6)$, $(3, 1)$, we compute the utility $U = 3.625$ for this candidate point 2 by Equation 6. Similarly $U = 2.005$ for point 5. We choose point 2 to be the next sample point for experiments, as the runtime between 1 and 3 decreases steeply (by exploitation). Further, suppose the actual runtime for configuration value 2 is 4. Then we update the model as shown in subfigure (b). At that moment, we compute 1.25 and 2.005 utility for configuration value 2.5 and 5, respectively. We choose the configuration value 5 as our next sample, as the largest uncertainty of points between 3 to 7 (by exploration).

It is noteworthy that the above sampling technique balances the conflicting tasks of exploration (understanding the global surface of the model) and exploitation (going through the regions, where the performances of the adjacent points change fast) that arise in model building, which is non-trivial to achieve.

6 Empirical Evaluation

In this section, we present the empirical results and evaluate our approach systematically. Experimental results show the superiority of the d-Simplexed model and adaptive sampling over the state-of-the-art models and baseline sampling techniques. The detailed evaluations are the following.

6.1 Experiment Setup

6.1.1 Experiment Design

Compared state-of-the-art models. We compared the d-Simplexed framework with baseline method Multivariate Linear Regression (LR) [16], and state-of-the-art methods, i.e., Decision Tree Regression (DTR) [16], Gaussian Process (GP) [1], Ernest [35] and Multilayer Neural Network (NN) [42] .

Compared samplers. We implemented adaptive sampling, comparing with two baseline sampling techniques: random, grid. The adaptive sampling helps to achieve the next best samples for the d-Simplexed as well as the compared models.

Evaluated workloads. Our evaluation consists of two suites of workloads. The first suite of workloads, i.e., WordCount, PageRank, Kmeans clustering, Bayesian classification, and SQL Join (described in Table 3), is from the HiBench benchmarking [19]. The second suite of workloads, i.e., SingleWave, MultiWave, Kmeans4d (described in § 6.2.4), is the synthetic turbulent surface to simulate more complex workloads for Spark or any other systems.

6.1.2 Experiment Setting

On the software side, we used Apache Spark v2.1.0 on top of Hadoop v2.8.1. With Hadoop, we used HDFS as our distributed file system, and YARN as our resource manager. This is a typical open-source Spark software stack. From the hardware side, our experiments were conducted on a high performance computing cluster. The cluster consists of over 10 Dell PowerEdge M610 servers, each having 32GB of RAM, 2 Intel Xeon E5540 2.53GHz CPUs and 4 cores per CPU, making 16 vcores available with hyperthreading. We construct various experiment specific feature spaces from the available resources in this cluster for our experiments. We present runtimes as the average of three experimental runs. For each experiment, we randomly took 10% of corresponding feature space samples as the test data for computing accuracy.

6.1.3 Model Implementation

d-Simplexed is implemented as a Python project organized as a set of modules. We implement Delaunay Triangulation by using a Python wrapperfor the `qhull`[5] library which internally uses the Quickhull Algorithm ([3]) to determine the Convex Hull and resulting Delaunay Triangulation for a set of points. The `qhull` library also contains useful abstractions for the components in the triangulation, such as determining if a simplex contains a point. Latin Hypercube Sampling is developed in Python from scratch.

We implemented LR, DTR, and GP by `scikit-learn`[6] with their default model settings. Ernest is trained by the curve-fitting function from sklearn. We implemented Multilayer Neural Network (NN) (with three hidden layers) [42] by `Keras`[7] using TensorFlow [12] backend.

6.2 Experiment Results

6.2.1 Overview of Workload Evaluation

In this part, we show a high level comparison of different prediction models for different workloads. Figure 7 shows the prediction accuracy of all workloads with 2% training samples out of all configuration combinations. In general, Delaunay Triangulation (DT) model with adaptive sampling, i.e., *d-Simplexed* outperforms the multi-variable Linear regression (LR), Gaussian Process (GP), Decision Tree regression (DTR), Multi-layer Neural Network (NN) for all evaluated workloads. *d-Simplexed* achieves less than 5% MAPE for all HiBench workloads except Bayesian 11.281%. In challenging SingleWave and MultiWave workloads, *d-Simplexed* achieves 9.576% and 20.723% MAPE, respectively.

Among all the compared methods, DTR performs satisfactorily, as DTR owns piece-wise regression function similar to *d-Simplexed*. Also, GP and NN outperform LR in the case of Kmeans, Bayesian and synthetic workloads (i.e., SingleWave & MultiWave), since the performance of these workloads change abruptly (usually non-linear changing) with feature values. No surprise, GP and NN are more suitable than LR in these non-linear curves fitting. In contrast, LR outperforms GP in the case of WordCount and Join workloads, as the performance of these two workloads is flatter, i.e., slight performance change as the parameters change.

We next show the detailed evaluation concerning various prediction models, features, sampling techniques, and complex workloads.

5. http://qhull.org/
6. https://scikit-learn.org/
7. https://keras.io

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

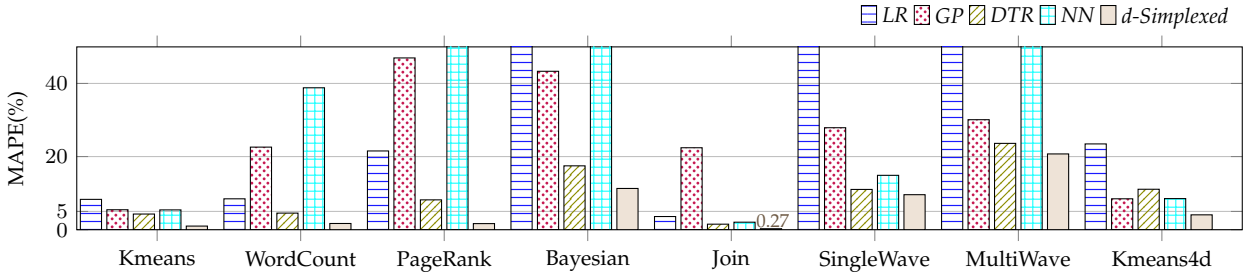


Fig. 7: Overall workload Mean Absolute Percentage Error (MAPE) of d-Simplexed agianst multi-variable Linear regression (LR), Gaussian Process (GP), Decision Tree regression (DTR), and multi-layter Neural network (NN). 2% of samples are trained. (The lower the better.) More than 50% MAPE is cut for better presentation.

TABLE 3: Settings and evaluations of Hibench Workloads, i.e., Kmeans, WordCount, PageRank, Bayesian, and Join with multi-variable Linear regression (LR), Gaussian Process (GP), Decision Tree regression (DTR), multi-layter Neural network (NN), Delaunay Triangulation (DT) with random sampling (DT-RD), DT with grid sampling (DT-GD), and DT with adaptive sampling (d-Simplexed). The results are the Mean Absolute Precision Error (MAPE) with 1% of samples. d-Simplexed achieves the best performance against other methods. Configuration space: start value - step size - end value

Workload	Data size	Config space	LR	GP	DTR	NN	DT-RD	DT-GD	d-Simplexed
Kmeans	80G	M:40-2-240 V:60-1-160	9.379	28.017	5.645	27.218	2.91	2.51	**1.58**
WordCount	80G	M:40-2-240 V:60-1-160	8.93	10.256	5.085	40.772	2.086	4.665	**2.071**
PageRank	80G	M:40-2-240 V:60-1-160	23.914	50.076	14.765	73.218	4.354	10.763	**3.053**
Bayesian	14G	M:10-1-120 V:10-1-120	85.36	66.111	26.646	68.481	20.091	345.045	**15.768**
Join	17.3G	M:10-1-120 V:10-1-120	3.278	20.357	1.833	2.391	2.674	0.954	**0.367**

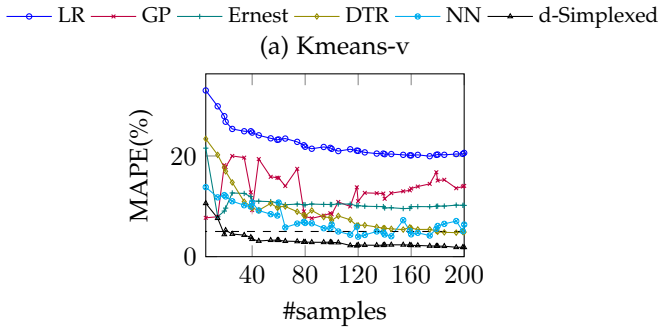


Fig. 8: Comparison of the d-Simplexed against Multivariate Linear Regression (LR), Gaussian Process (GP), Ernest, Decision Tree regression (DTR), multi-layter Neural network (NN) for the Kmeans workload. d-Simplexed achieves the best MAPE with 200 samples.

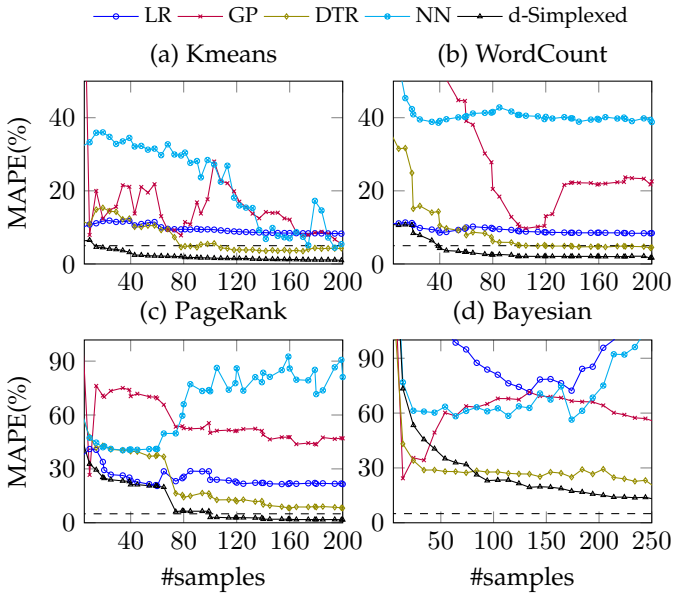


Fig. 9: Comparison of the d-Simplexed against Multivariate Linear Regression (LR), Gaussian Process (GP), Decision Tree regression (DTR), multi-layter Neural network (NN) for Kmeans, WordCount, PageRank, and Bayesian workloads. d-Simplexed achieves the best MAPE.

6.2.2 Model Evaluation

The first set of experiments compares d-Simplexed with adaptive sampling (algorithm 3), against other state-of-the-art methods, namely Multivariate Linear Regression (LR) and Gaussian Process (GP), Ernest, Decision Tree regression (DTR), and multi-layter Neural network (NN).

We first consider input data size and vcore as the features, since Ernest only tolerates these two features. We compare all the models with variable data sizes (10GB-40GB) and numbers of vcores (20-50v). The unit size of each step is 1GB data and 1 vcore. Figure 8 shows the experimental results against Kmeans workload. d-Simplexed is the best among these six models in terms of Mean Absolute Percentage Error (MAPE) Equation 2 and convergence because of its property of local and piece-wise modeling and prediction. Ernest is better than LR because Ernest consists of not only linear but also logarithm terms which better capture the performance properties. GP and NN flip around, which is typical behavior of over-fitting globally, as the samples increase. Since Ernest does not include other features, we exclude it in the rest cases.

We now select memory and vcore as the features, since these two parameters are common yet challenging decision to make in the cloud environment (e.g., Amazon on-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

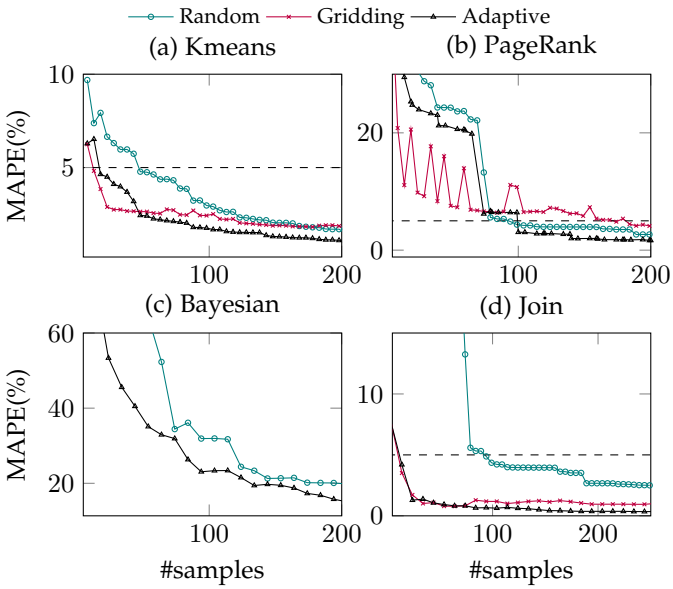


Fig. 10: Comparison of sampling techniques with Delaunay Triangulation; Random Sampling, Grid Sampling, and Adaptive Sampling. Adaptive sampling excels in discovering the samples to improve the model. The dashed line is the 5% MAPE line.

demand price). Table 3 shows the detailed evaluations of HiBench workloads against state-of-the-art methods in terms of MAPE. 1% of the training samples are used in this evaluation. d-Simplexed is the best against other methods concerning MAPE. It achieves less than 5% MAPE in Kmeans, WordCount, PageRank, and Join workloads and 15.768% MAPE in the Bayesian workload.

Figure 9 depicts the more detailed MAPEs for methods with workloads, as sample size increases. Specifically, LR performs poorly in (a) Kmeans and (d) Bayesian workloads since a linear model faces difficulty in capturing non-linear behavior. In contrast, GP and NN models work relatively better than LR in these two workloads. However, they have the problem of over-fitting for the data as more feature points are loaded into the model. However, GP and NN perform unsatisfactorily for (b) WordCount and (c) PageRank workload. DTR splits configuration space piece-wisely, resulting in satisfactory MAPE. Nevertheless, DTR uses an average value to represent the local model instead of a linear regressor like in d-Simplexed, and therefore, poorly catches the surface of local configuration space. Unfortunately, we found d-Simplexed sometimes does not yield the best at the beginning phase. For example, GP and DTR outperform d-Simplexed in Figure 9(d) with fewer samples ($\leq$40). This is because non-linear regressors have a better regression than the fewer linear surfaces in d-Simplexed.

As the samples increase, d-Simplexed outperforms them. There are two reasons. First, when new sample points are added, the d-Simplexed model prefers to fit them locally, that is, their placement only impacts a few adjacent simplexes, rather than the entire model (in the case of LR, GP and NN). Second, d-Simplexed uses utility function (in Equation 6) to judiciously pick the next point, which continuously improves the model as the sample size increases.

6.2.3 Sampling Evaluation

The second set of experiments compares the performance of Delaunay Triangulation (DT) with random and grid sampling. In each sampling approach, initially, we select 4 boundary points and 4 seed samples retrieved using LHS sampling, giving 8 samples to create the initial model with. For each sampling technique, the model is then iteratively improved by adding new samples 1-by-1 and evaluating the MAPE at each step.

Table 3 shows the detailed evaluations of d-Simplexed against DT with random sampling (DT-RD) and DT with grid sampling (DT-GD) in terms of MAPE. d-Simplexed achieves the best MAPE among three samplings. Figure 10 plots more detailed steps for three sampling techniques with the Kmeans, PageRank, Bayesian, and Join workloads. It shows that adaptive sampling achieves the relatively better MAPE among three techniques, given the same number of samples after certain points, meaning that the adaptive approach can be used to build a more accurate performance model. Particularly in the Bayesian workload, grid sampling achieves more than 60% MAPE with even more than 200 samples. This is because the performance surface of the Bayesian changes rapidly in small value area while grid sampling keeps sampling in the global space.

The performance of random sampling is unsatisfactory, especially when the model has a few sampling points. This is because, with a few points in the model, there is a high probability that the next selected random point will fall in a region which has a small performance change, i.e, a small contribution to predicting the critical turning point of performance. Grid sampling is better than random sampling, and it performs well at the beginning phase, as it evenly distributes the chance of picking points for the model. In contrast, adaptive sampling selects the points that continuously improve the accuracy of the model. The adaptive sampling technique avoids picking points in the regions where there are little changes in performance (by Equation 6). This behavior is especially pronounced, where adaptive sampling zooms-in on the critical region of the model which has a steep performance change, while grid and random sampling use a static strategy to pick points randomly or uniformly.

6.2.4 More Evaluation Results

The third set of experiments challenges d-Simplexed with more complex synthetic workloads and with more input features. We describe in detail as follows.

Synthetic Workload In this set of experiments, we sought to analyze the performance of models against a synthetic workload. The purpose of these experiments is to simulate more complex or arbitrary workloads from Spark or any other systems. Specifically, we create a synthetic 120×120 =14400 point $\langle f_1, f_2 \rangle$ surface to demonstrate the model's flexibility under a hypothetical runtime condition, where the runtimes have massive turbulence as shown in Figure 11. We assume that some workloads (e.g., with dependent parameters [1], [11]) may exhibit such behavior, and it is an interesting surface to challenge with our model.

We evaluated d-Simplexed for synthetic surfaces against the state-of-the-art models and presented the results in Figure 12. LR perform unsatisfactory, as its linear fitting is not

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2019.2948338, IEEE Transactions on Big Data

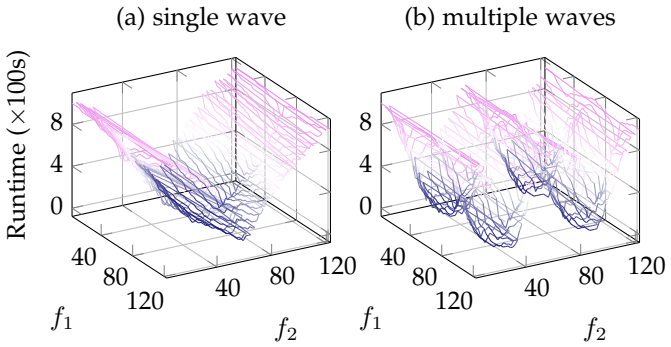


Fig. 11: Synthetic workload with a massive turbulent surface illustrated in three dimensions.

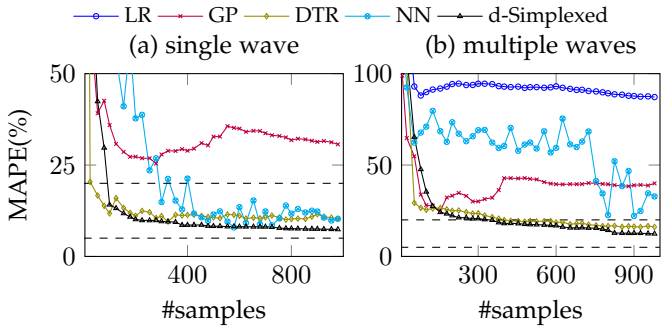


Fig. 12: Comparison of the models for synthetic turbulent surface, d-Simplexed achieves satisfactory and convergent accuracy as samples increase while LR and GP make the model overfit. The dashed lines are in 5% and 20% MAPE line.

suitable for the turbulent non-linear surface. GP and NN are much better than LR, nevertheless, they still cannot achieve a huge error, especially in a more complex multi-wave workload. The reason is that GP and NN consider the surface as a whole and cannot adapt to the local flexibility. What's worst, with additional samples, LR and GP may overfit the samples, thus deteriorate the models. On the contrary, DTR and d-Simplexed keeps discovering the unknown regions and continuously improving the model as they split configuration space piece-wisely, avoiding overfitting. Thanks to the utility function in adaptive sampling and a better regressor in d-Simplexed, d-Simplexed outperforms DTR. Due to the complex surface, we finally achieve less than 10% and 20% MAPE with 2.36% samples for the synthetic data with the single wave and multiple waves, respectively.

More Input Features In this part, we additionally investigate the performance of d-Simplexed with 3 input features. We setup 3-parameter (i.e., vcore (21-50), memory(21-50G), and data size(11-40G)) with 30·30·30 experiments. Figure 13 shows that d-Simplexed outperforms the state-of-the-art methods in a higher dimension situation too.

It is noteworthy that the number of required experiments grows exponentially [42], as the number of features increases. For example, when the total dimension is three and we have two feature parameters (e.g., CPU and memory) each with 30 values, then we need 900(=30·30) ground truths to verify the model. However, adding one more dimension,

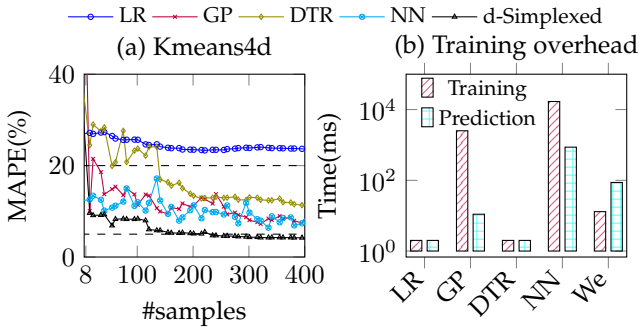


Fig. 13: (a) Comparison of the prediction models for Kmeans workload with 4 dimension model; Multivariate Linear Regression (LR), Gaussian Process (GP), and d-Simplexed. d-Simplexed reaches less than 5% error by 0.80% configuration spaces. The dashed lines are in 5% and 20% MAPE line. (b) Training cost for one round modeling and prediction for 1.5% samples. d-Simplexed requires less than one second for both training the model and making prediction.

e.g., data size with a granularity of 30, will need 27000(=30^3) ground truths to verify the model. Since running all 27000 experiments of workload would be a nightmare, we only run partially needed experiments, i.e., 400 samples run and 320 (0.5% of all) test results. It is worth noting that only 238 (0.80% of all) samples are required for DT to achieve less than 5% MAPE.

Training overhead Figure 13(b) shows the training overhead of all the models for running Kmeans workload in Figure 13(a). LR and DTR require relatively low cost for training model and making prediction due to its simpler model in nature. GP and NN need more significant time on training the model parameters. Although d-Simplexed requires more training and prediction time than LR and DTR,it yields better performance in accuracy and still just requires less than 1 second for both training the model and making prediction from 400 samples.

6.3 Evaluation Summary

We highlight our main findings in the experiments as follows:

- d-Simplexed, by using 1% of samples, achieves less than 5% MAPE in Kmeans, WordCount, PageRank, and Join workloads and 15.768% MAPE in the Bayesian workload, outperforming all the state-of-the-art models (i.e., Ernest, LR, GP, DT, and NN).
- The proposed adaptive sampling method enables us to use fewer samples to train the model, outperforming basic sampling techniques, i.e., random and gridding sampling.
- d-Simplexed outperforms the state-of-the-art models (i.e., LR, GP, DT, and NN)) both in more challenging synthetic workloads and with more (3) input features.

7 Related Work

MapReduce and Hadoop Performance Prediction. Recently, there have been emerging efforts to predict the performance of MapReduce and Hadoop jobs. The basic cost models for these two frameworks are introduced in [24], [39]. These approaches require complex instrumentation

of the frameworks for fine-grained modeling of the jobs. Among such methods, StarFish [17] and MRTuner [30] use the derived models to set the optimal values of the features. ARIA [36] models job execution time concerning the number of mappers and reducers from historical traces. HP [37] extends this model with a scaling factor to estimate the job execution time using simple linear regression. Khan et al. [21] proposed a non-parametric prediction model based on Locally Weighted Linear Regression and built on a large set of historical Hadoop job traces. AROMA [23] depends on historical traces and applies clustering to identify the jobs with similar behavior, and then applies pattern matching to find the optimal resources for a job. ALOJA-ML [4] is another machine learning-based framework for predicting the execution time of Hadoop machine learning jobs. The framework maintains a large collection of job execution time and resource configurations. Ernest [35] constructs prediction models by executing jobs with a fraction of the input data. Our method proposed in this paper relies on a small amount of historical job execution traces to derive the prediction models with Delaunay Triangulation. Also, our method is applicable to Hadoop and MapReduce frameworks.

Database Performance Prediction. Database systems have many mature tools and guidelines for physical design tuning (e.g., index selection, materialized view generation). However, they do not provide a performance model with the varied configuration parameters, because they depend on the query optimizer's cost models, which do not capture the effects of many important parameters. There a few works have been done on the prediction model with various configuration parameters in modern database systems. For example, MDN [22] (Mixture Density Network) is a black-box model that feeds the historical traces of Hive queries with features and performance to a neural network which can predict the execution time of a new job. ParaTimer [27] predicts the progress of running parallel database queries that are expressed as Pig Scripts which together resembles a DAG of MapReduce jobs running in parallel.

Database Sampling Techniques. Traditional database sampling deals with the problem of sampling from a large dataset, and their main purpose is to estimate the cardinality of query results (or intermediate results). In contrast, our sampling strategy in § 5 aims to draw samples from a response surface that is never materialized fully. Another related paper iTuned [11] relies on Gaussian Process (GP) to decide the next sampling points. While our method *d*-Simplexed shares the main goal for adaptive sampling to build an accurate performance model, the application scenarios and algorithms differ.

8 Conclusions and Future Work

In this paper, we studied the problem of predicting the performance of Big Data analytics platforms with a small amount of historical traces. We demonstrated how a geometric interpolation method: *Delaunay Triangulation* could be used to model the feature configurations for predicting the runtime of the analytics jobs. We proposed a sampling strategy to select data points judiciously for faster and accurate model construction. Finally, we performed empirical experiments to demonstrate the superiority of our method over the-state-of-art approaches.

Exciting follow-up research can be centered around the implementation of parameter tuning algorithms based on the prediction model proposed in this paper. In addition, we intend to explore the extension of the *d*-Simplexed framework to handle various large-scale machine learning platforms. Consider, for instance, the problem of building an accurate performance model on TensorFlow [12] or SystemML [14].

References

[1] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *SIGMOD Conference*, pages 1009–1024. ACM, 2017.

[2] S. J. Anderson, S. Karumanchi, and K. Iagnemma. Constraint-based planning and control for safe, semi-autonomous operation of vehicles. In *Intelligent Vehicles Symposium*, pages 383–388. IEEE, 2012.

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[4] J. L. Berral, N. Poggi, D. Carrera, A. Call, R. Reinauer, and D. Green. Aloja-ml: A framework for automating characterization and knowledge discovery in hadoop deployments. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1701–1710, New York, NY, USA, 2015. ACM.

[5] V. Bryant. Linear programming duality; an introduction to oriented matroids, by a. bachem and w. kem. pp. 216. dm 68. 1992 isbn 3-540-55417-3 (springer). *The Mathematical Gazette*, 77(480):387–387, 1993.

[6] H. Chen and J. Bishop. Delaunay triangulation for curved surfaces. *Meshing Roundtable*, pages 115–127, 1997.

[7] K. Chen, J. Powers, S. Guo, and F. Tian. CRESP: towards optimal resource provisioning for mapreduce computing in public clouds. *IEEE Trans. Parallel Distrib. Syst.*, 25(6):1403–1412, 2014.

[8] J. A. De Loera, J. Rambau, and F. Santos. *Triangulations Structures for algorithms and applications*. Springer, 2010.

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

[10] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, pages 793–800, 1934.

[11] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. *Proc. VLDB Endow.*, 2(1):1246–1257, 2009.

[12] M. A. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015.

[13] P.-L. George and H. Borouchaki. Delaunay triangulation and meshing. 1998.

[14] A. e. a. Ghoting. Systemml: Declarative machine learning on mapreduce. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 231–242. IEEE, 2011.

[15] A. Gounaris, G. Kougka, R. Tous, C. T. Montes, and J. Torres. Dynamic configuration of partitioning in spark applications. *IEEE Trans. Parallel Distrib. Syst.*, 28(7):1891–1904, 2017.

[16] Á. B. Hernández, M. S. Perez, S. Gupta, and V. Muntés-Mulero. Using Machine Learning to Optimize Parallelism in Big Data Applications. *Future Generation Computer Systems*, pages 1–12, 2017.

[17] 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*, volume 11, pages 261–272, 2011.

[18] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

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

[20] R. L. Iman. Latin hypercube sampling. *Wiley StatsRef: Statistics Reference Online*, 2008.

[21] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang. Hadoop performance modeling for job estimation and resource provisioning. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):441–454, Feb 2016.

[22] A. Khoshkbarforoushha and R. Ranjan. Resource and performance distribution prediction for large scale analytics queries. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, pages 49–54, New York, NY, USA, 2016. ACM.

[23] P. Lama and X. Zhou. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 63–72, New York, NY, USA, 2012. ACM.

[24] X. Lin, Z. Meng, C. Xu, and M. Wang. A practical performance model for hadoop mapreduce. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing Workshops*, CLUSTERW '12, pages 231–239, Washington, DC, USA, 2012. IEEE Computer Society.

[25] M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[26] C. C. Morris and R. M. Stark. *Finite Mathematics: Models and Applications*. John Wiley & Sons, 2015.

[27] K. Morton, M. Balazinska, and D. Grossman. Paratimer: A progress indicator for mapreduce dags. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 507–518, New York, NY, USA, 2010. ACM.

[28] N. M. Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[29] C. E. Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[30] J. Shi, J. Zou, J. Lu, Z. Cao, S. Li, and C. Wang. Mrtuner: a toolkit to enable holistic optimization for mapreduce jobs. *Proceedings of the VLDB Endowment*, 7(13):1319–1330, 2014.

[31] R. Singhal and P. Singh. Performance assurance model for applications on spark platform. In R. Nambiar and M. Poess, editors, *Performance Evaluation and Benchmarking for the Analytics Era*, pages 131–146, Cham, 2018. Springer International Publishing.

[32] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *SIGMOD Conference*, pages 953–966. ACM, 2008.

[33] D. G. Sullivan, M. I. Seltzer, and A. Pfeffer. Using probabilistic reasoning to automate software tuning. In *SIGMETRICS*, pages 404–405. ACM, 2004.

[34] 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*, page 5. ACM, 2013.

[35] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *NSDI*, pages 363–378, 2016.

[36] A. Verma, L. Cherkasova, and R. H. Campbell. Aria: Automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 235–244, New York, NY, USA, 2011. ACM.

[37] A. Verma, L. Cherkasova, and R. H. Campbell. Resource provisioning framework for mapreduce jobs with performance goals. In F. Kon and A.-M. Kermarrec, editors, *Middleware 2011*, pages 165–186, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

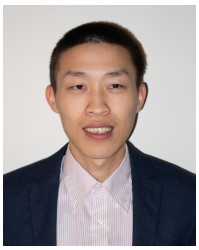
[38] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[39] N. Yigitbasi, T. L. Willke, G. Liao, and D. H. J. Epema. Towards machine learning-based auto-tuning of mapreduce. In *MASCOTS*, pages 11–20. IEEE Computer Society, 2013.

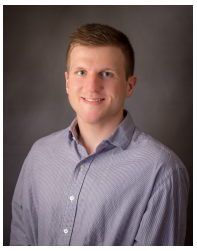
[40] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[41] M. e. a. Zaharia. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[42] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *SIGMOD Conference*, pages 415–432. ACM, 2019.



Yuxing Chen is a doctoral student in the Department of Computer Science at University of Helsinki, Finland. He obtained (2016) MSc in Computer Science and Engineering from Politecnico di Milano, Italy, and (2014) BSc in Computing Information Science from Guangdong University of Technology, China, respectively.



Peter Goetsch is a graduate student working in the UDBMS group at the University of Helsinki. His research interests include Big Data performance and evaluation, as well as Distributed System design. He has a BSc in Computer Science from the University of Wisconsin - Madison.

Mohammad A. Hoque is a postdoctoral researcher at the University of Helsinki. He obtained his M.Sc degree in Computer Science and Engineering in 2010, and Ph.D in 2013 from Aalto University. His research interests include energy efficient mobile computing, data analysis, distributed computing, and resource-aware scheduling.

Jiaheng Lu is an Associate Professor at the University of Helsinki, Finland. His main research interests lie in the Big Data management and database systems. He has published more than eighty journal and conference papers. He has published several books, on XML, Hadoop and NoSQL databases. His book on Hadoop is one of the top-10 best-selling books in the category of computer software in China in 2013. He has frequently served as a PC member for conferences including SIGMOD, VLDB, ICDE, EDBT, CIKM etc.

Sasu Tarkoma is a Professor of Computer Science at the University of Helsinki, and Head of the Department of Computer Science. He has authored 4 textbooks and has published over 160 scientific articles. His research interests are Internet technology, distributed systems, data analytics, and mobile and ubiquitous computing. He has seven granted US Patents. His research has received several Best Paper awards and mentions, for example at IEEE PerCom, ACM CCR, and ACM OSR.